Applying Aspect-Oriented Change Realization in the Mobile Application Domain

Sandra Kostova
Institute of Informatics, Information Systems and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Slovakia
kostova.sandra@gmail.com

Valentino Vranić
Institute of Informatics, Information Systems and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Slovakia
vranic@stuba.sk

ABSTRACT

Aspect-oriented programming makes possible to express changes in a modular way directly at the level of programming language constructs. This is particularly useful in situations that require explicit change manipulation, such as change reapplication to another version branch in application customization. However, it may be tricky to employ appropriate aspect-oriented constructs in a correct way that accommodates well the change to be implemented. This has been successfully addressed by an approach to aspect-oriented change realization based on a two-level change type model, which features a catalog of specification and implementation change types and their relationships, primarily targeting the web application domain. In this paper, we explore the applicability of the change types gathered in this catalog for the mobile application domain. For this, we performed a study that involved a set of hypothetical scenarios and two real mobile applications for Android. The study revealed that the change types known from the web application domain are in their essence applicable to the mobile application domain. It also lead to the discovery of four new specification change types in the mobile application domain and the corresponding specification–implementation change type relationships along with further relationships between known change types.

CCS CONCEPTS

• **Human-centered computing** → **Smartphones**; **Mobile devices**; • **Social and professional topics** → **Software maintenance**; • **Software and its engineering** → **Abstraction, modeling and modularity**; **Software configuration management and version control systems**; **Software version control**; **Maintaining software**;

KEYWORDS

aspect-oriented programming, change modularization, mobile applications, Android, maintenance

ACM Reference Format:
Sandra Kostova and Valentino Vranić. 2018. Applying Aspect-Oriented Change Realization in the Mobile Application Domain. In *Proceedings of 2nd International Conference on the Art, Science, and Engineering of Programming (<Programming'18> Companion).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3191697.3213806

<Programming'18> Companion, April 9–12, 2018, Nice, France

ACM ISBN 978-1-4503-5513-1/18/04…$15.00
https://doi.org/10.1145/3191697.3213806

1 INTRODUCTION

Every software system, from the moment of being conceived, is subject to changes. Most of the work in the area of change management and version control focuses either on tracking changes and versions in code (with the most prominent system for this being Git) or models [29] as realization artifacts, or on tracking changes in requirements [2, 10] as specification artifacts. Aspect-oriented change realization [36] interconnects these two areas by dealing with changes in code as with conceptual units in terms of specification providing means for a quick transition from a change request to its implementation, which, moreover, remains in one module.

Aspect-oriented programming can be used to express non-corrective changes[1] in a modular way directly at the level of programming language constructs [11]. This is particularly useful in situations that require explicit change manipulation, such as change reapplication to another version branch in application customization [35, 36]. However, it may be tricky to employ appropriate aspect-oriented constructs in a correct way that accommodates well the change to be implemented. This has been successfully addressed by an approach to aspect-oriented change realization based on a two-level change type model, which features a catalog of specification and implementation change types and their relationships (further referred to simply as *catalog of change types*), primarily targeting the web application domain [3].

Aspect-oriented programming has not yet come to be widely accepted on mobile platforms, but it is possible to apply it [22, 30]. Since mobile applications are not less than desktop applications subject to numerous changes, mobile application development could benefit from using aspect-oriented change realization.

The rest of the paper is structured as follows. Section 2 revisits the catalog of known change types in the context of aspect-oriented

[1] Corrective changes can be realized using aspect-oriented programming, too, but that would cause unnecessary version proliferation, as there is little use of keeping the possibility of restoring incorrect versions.

change realization in the mobile application domain. Section 3 presents the implementation change types from this catalog that we applied in our study. Section 4 explains newly identified specification change types for the mobile application domain. Section 5 discusses change interaction. Section 6 reflects on related work. Section 7 concludes the paper.

2 APPLYING KNOWN ASPECT-ORIENTED CHANGE TYPES TO MOBILE APPLICATIONS

As we mentioned in the introduction, in the two-level aspect-oriented change realization model [34], one level contains implementation change types, which are generally applicable regardless of the application domain, while the other one contains specification change types, which are domain specific. Given a change request, it is first decomposed into individual changes, each of which is then separately analyzed and generalized and then looked for a similar phrasing among the specification change types in the catalog of change types. Each specification change type points to the corresponding implementation type, which is specified by a code scheme.

Upon identifying the specification change type that suits the needs of the required change, the corresponding implementation change type, which includes a ready to be be applied code scheme, is simply read from the cataloged relationships. Implementation change types recall design patterns and some of them actually are design patterns. Sometimes, several implementation types can be used alternatively or may have to be combined in order to implement the given specification change type.

Consider an example. One of the functions of the contact management mobile application is to provide telephone numbers owned by the company employees. The application requires registration. There are two account types: administrator and common employee. In addition to common employee rights, administrators can see the contact details of all employees, change the position of other employees, delete contacts, etc. Assume that a change request comes stating that common employees should be prevented from accessing the options for deleting or changing the employee information. They are already prevented from performing these operations in the application logic, but the corresponding options are still available to them in the user interface. What we need to do is to restrict common employees in accessing the user interface accordingly. Going through specification change types, we may find that the User Interface Restriction specification change type, for which the catalog reads, "It is quite annoying when a user sees, but can't access some options due to user rights restrictions," quite corresponds to this situation. Strictly speaking, this is rather a generalized example than a definition, but this format seems to suit matching to change specifications.

The existing catalog of change types for the web application domain contains the following change types and relationships (specification change types given first) [36]:

- One Way Integration: Performing Action After Event
- Two Way Integration: Performing Action After Event
- Adding Column to Grid: Performing Action After Event
- Removing Column from Grid: Method Substitution
- Altering Column Presentation in Grid: Method Substitution
- Adding Fields to Form: Enumeration Modification with Additional Return Value Checking/Modification
- Removing Fields from Form: Additional Return Value Checking/Modification
- Introducing Additional Constraint on Fields: Additional Parameter Checking or Performing Action After Event
- Introducing User Rights Management: Border Control with Method Substitution
- User Interface Restriction: Additional Return Value Checking/Modifications
- Introducing Resource Backup: Class Exchange

According to the catalog, the Additional Return Value Checking/-Modification is an implementation change type that corresponds to the User Interface Restriction specification change type. In our situation, the menu consists of the list of the options which are consequently rendered by the user interface part of the application. The menu is returned by the logon() method. To restrict the user interface for a common user then means simply to leave out the options not allowed for this type of users. By applying the corresponding code scheme included in the catalog, we can write the following aspect:

```
public aspect CommonUserMenuRestriction {
    private Menu menu;
    pointcut logonmethod(User user): call(* User.logon(..))
        && target(user);

    Menu around(User user): logonMethods(user) {
        menu = proceed(user);
        restrictMenu(user);
        return menu;
    }
    private void restrictMenu(User user) {
        if (user.type == UserTypes.common) {
            ... // leave out non−administrative options from the menu
        }
    }
}
```

Let us consider few other change requests (code omitted for space reasons). One of them might state that enhancing the restrictions to the user interface by introducing a full-fledged user rights management is needed. In effect, this would concern limiting access to sensitive parts of the application when a common user is in charge, which brings us to the Introducing User Rights Management specification change type. According to the catalog, this change is implemented by the Border Control aspect-oriented design pattern [23] combined with the Method Substitution implementation change type. Border Control specifies the restricted region in terms of method calls prohibited for common users. Method Substitution empowers this pointcut by a piece of around advice to decide upon the execution of these calls.

Another change request might be to introduce telephone number validation consisting of checking whether the telephone number already exists in the application and whether it is in a correct format. This is as if we would need to integrate our application with the telephone number validation. It is not necessary for the validation

part to be aware of the rest of the application, i.e., the integration is one way only. The catalog contains a specification change type named exactly like that: One Way Integration. The corresponding implementation change type is Performing Action After Event, with the action being the telephone number validation, and the event being the entering of the telephone number.

Consider one more change request that demonstrated applicability of known aspect-oriented change types to mobile applications: calling prefixes should be appended automatically to telephone numbers entered into forms if they contain no calling prefix themselves. Adding a calling prefix means adding constraints to the telephone number field, which brings us to the Adding Constraint to the Field specification change type, to be implemented as the Additional Parameter Checking implementation change type.

3 APPLIED IMPLEMENTATION CHANGE TYPES IN MOBILE APPLICATIONS

To assess the two-level aspect-oriented change realization approach in the mobile application domain, we performed a study involving implementing a series of changes in two mobile applications. Diabetes Monitoring System (DMS)[2] is intended to help diabetics to better manage their disease and keep it under control. Expense Tracker (ET)[3] helps users manage their money by providing weekly, monthly, and yearly reports about their expenses.

Since we were unable to find the corresponding specification change types in the existing catalog of change types, we resorted directly to implementation change types. This indicates the need for establishing specification change types specific for mobile applications, which we address in Section 4. Sections 3.1–3.5 describe the experience of implementing selected implementation change types from the known catalog of change types for the web application domain.

3.1 Performing Action After Event

We successfully used the Performing Action After Event implementation change to implement sound notification, vibration notification, and logging in DMS and ET. An example of logging screen changes in DMS is shown below:

```
public aspect ActivityAspect {
    private pointcut getViewPointcut(int i):
        call(* setContentView(int)) && args(i);

    after(int i): getViewPointcut(i) {
        getActivityLayoutName(i);
    }
    public void getActivityLayoutName(int i) {
        if (R.layout.activity_main == i) {
            Log.i("INFO", "Main User Activity is on");
            return;
        } else if (R.layout.add_new_user == i) {
        ...
        } else if (R.layout.login == i){
            Log.i("INFO", "Login Activity is on" );
            return;
        }
    }
}
```

The calls to the setContentView() method, which actually sets the main layout (activity), along with the layout number, an argument to this method, are being captured. The layout number is then interpreted in terms of existing layouts with the corresponding information being logged.

3.2 Additional Parameter Checking

In DMS, the Additional Parameter Checking implementation change type was used to introduce checking the entered doctor number value and checking the entered name values:

```
public privileged aspect NameValidatorAspect {
    String userName, doctorName;
    boolean ok, ok1;
    protected pointcut myClass(): within(AddNewUser);
    private pointcut getName(): myClass() && execution(* save(..));

    void around(): getName() {
        userName = ((AddNewUser) (thisJoinPoint.getThis())).
            userName.getText().toString();
        doctorName = ... // analogously as userName
        ok = nameValidation(userName);
        ok1 = nameValidation(doctorName);

        if (!ok) {
            Toast.makeText(AddNewUser.getContext(), ...).show();
            return;
        }
        if (!ok1) { ... // analogously as with ok
        }
        proceed();
    }
    private boolean nameValidation() {
        ...
    }
}
```

One of the difficulties we encountered was in accessing the names. They were defined as private attributes and no get methods were implemented either. In AspectJ, an aspect can access a member of a class even though it is private if the aspect is declared as privileged, which is what we used. In effect, this means breaking encapsulation, so privileged aspects have to be used with caution. In ET, the Additional Parameter Checking implementation change type was applied to impose checking the number of unfinished expense entries performed upon adding or deleting an expense.

3.3 Additional Return Value Checking/Modification

Assume that in ET we would like to round the amount of the expense entry to two decimal places whenever a user inserts a new expense entry. To do this, the return value of the method that saves the entry data must be changed:

```
Object around(EditAbstract o): EntryPoint(o) {
    Entry list = new Entry();
```

[2] https://github.com/arsikj/Diabetes-Monitoring-System-Android
[3] https://github.com/vinsol/expense-tracker

```
    list.id = o.entry.id;
    o.entry.amount = o.editAmount.getText().toString();

    if (!o.entry.amount.equals(".") && !o.entry.amount.equals("")) {
        Double mAmount = Double.parseDouble(o.entry.amount);
        mAmount = (double) ((int) ((mAmount + 0.005) * 100.0) / 100.0);
        list.amount = mAmount + "";
        h.put("Amount Digits", new StringProcessing().
            getStringDoubleDecimal(list.amount).length()+"");
    } else {
        list.amount = "";
    }
    return list;
}
```

In DMS, the same implementation change type was used to introduce telephone number formatting in terms of introducing the current calling prefix if it is missing.

3.4 Class Exchange

The amount rounding we dealt with in the previous section actually requires using yet another implementation change type, Class Exchange:

```
public aspect AmountValidation {
    HashMap<String,String> h;
    private pointcut HashPoint():
        withincode(Entry getSaveEntryData(..))
            && call(HashMap.new(..));

    Object around(): HashPoint() {
        h = new HashMap<String, String>();
        return h;
    }
    Object around(EditAbstract o): EntryPoint(o) {
        ...
    }
}
```

A pointcut captures each new HashMap instance creation within the getSaveEntryData() method. In a piece of the around advice, that class instance is exchanged with another one. What follows is the around advice from the previous section that implements Additional Return Value Checking/Modification. Consequently, this was a very special implementation because we came to combine two different implementation change types in one aspect. This is actually a new way of implementing the Introducing Additional Constraint specification change type that needs to be cataloged (see Section 4).

3.5 Method Substitution

While in the Class Exchange implementation change type we are exchanging one class with another one, in the Method Substitution implementation change type it is the execution of a method that is being exchanged. Sometimes, the original method execution is completely disabled [36]. We used this change type to achieve duplicity checking in DMS. This means that after the Save button is pressed while registering a user, DMS checks whether a user with the same name already exists. If this is so, the application attempts to modify the name by preceding it with *NEW* and checks once more whether the user with this name already exists in the user list. For this, one method execution was substituted with another method execution. If there is such a name, the method execution is completely disabled (code omitted for space reasons). A very similar functionality was implemented in ET, too: duplicity check of the entered expense entry.

4 NEW SPECIFICATION CHANGE TYPES FOR THE MOBILE APPLICATION DOMAIN

In previous sections, we demonstrated that the two-level aspect-oriented change realization approach and the original catalog of change types for the web application domain is applicable to the mobile domain applications. Out of six implementation change types in this catalog, we successfully applied five. This was expected, since implementation change types in the two-level aspect-oriented change realization approach are expressed so that they do not depend on the application domain. This is why they were originally called generally applicable change types [3].

The situation was different with the specification change types, originally known as application specific change types [3]. Although some of the specification change types for the web application domain could be applied to the mobile application domain, there were situations in which no existing specification change types could be applied. Also, we observed further relationships between change types.

One of the new specification changes is Introducing Sound Notification, which was implemented in both DMS and ET to introduce a sound upon registering a new user. Another one is Introducing Vibration Notification, which was implemented in both DMS and ET to introduce a vibration upon pressing the back button.

We also discovered the Logging Activity Changes specification change. A typical activity change that one might want to log in Android applications is a change to the main activity, and this is what we implemented in both DMS and ET applications.

Introducing Display Constraints, the last specification change type we discovered, was applied to ET to introduce checking the number of unfinished expense entries. For this, the Additional Parameter Checking implementation change type was used. Actually, this change was a result of code refactoring. The existing functionality was rephrased as an aspect to enable its easy inclusion and exclusion. Table 1 summarizes the changes implemented in DMS and ET.

In order to be incorporated into the process of aspect-oriented change realization based on a two-level change type model, newly identified specification change types for mobile application domain have to be bound to corresponding implementation change types. The previous section indicated some of these relationships, along with some new relationships between already known change types. An example of this is amount rounding (recall Sections 3.3 and 3.4), which was identified to be of the Introducing Additional Constraint specification change type, and which was implemented using both Class Exchange and Add Return Value Checking Modification implementation change types. In the rightmost two columns, Table 1 summarizes the new relationships between change types we discovered in our study.

Table 1: Implemented change types in mobile applications.

Change	Application	Implementation Change Type	Specification Change Type
Name validation	DMS	Additional Parameter Checking	Introducing Additional Constraints on Fields
Telephone number formatting	DMS	Additional Return Value Checking/Modification	Introducing Additional Constraints on Fields
Telephone number validation	DMS	Additional Parameter Checking	Introducing Additional Constraints on Fields
Sound notification	both	Performing Action After Event	Introducing Sound Notification
Vibration notification	both	Performing Action After Event	Introducing Vibration Notification
Logging screen changes	both	Performing Action After Event	Logging Activity Changes
Duplicity control	both	Method Substitution	Altering Column Presentation in Grid
Amount rounding	ET	Add Return Value Checking/Modifications and Class Exchange	Introducing Additional Constraints on Fields
Checking the number of unfinished expense entries	ET	Additional Parameter Checking	Introducing Display Constraints

5 CHANGE INTERACTION

Interaction of changes realized in aspect-oriented way occurs mainly due to addressing and manipulating the same elements by pieces of advice belonging to different change implementations. Consider the interaction that occurred in DMS between telephone number validation, telephone number formatting, and name validation changes. Each of these changes address the same method named save() (see Section 3.2). The exact order in which the changes are applied can be determined by the rules of advice precedence, which are based on advice type and physical order of code [19]. The order of our changes happens to be as follows: telephone number validation, telephone number formatting, and name validation. Telephone number validation checks whether the entered numbers are valid. If they are not, telephone number formatting and name validation are not executed. This way, some changes may prevent other changes from being executed. In our case, this poses no problem because we want the application to format the number only if the telephone number validation passes successfully.

Another change interaction we noticed in DMS happened between sound notification and duplicity control. Duplicity control occurs around calls to the method of registering a new user. Sound notification occurs after the same method is called. According to advice precedence rules, a piece of the around advice will be executed before a piece of the after advice. Thus, sound notification, implemented as a piece of the after advice, will not accomplish its function if duplicity control prohibits the registration of a new user, as the join point sound notification waits for will never be executed.

6 RELATED WORK

Jalali and Bider [18] presented an idea of using aspect-oriented approach to adaptive case management. For this, they identified particular constraints and proposed their aspect-oriented realization, such as response restriction, precedence restriction, or not chain succession constraint, which are one level above aspect-oriented programming primitives, just like implementation change types in aspect-oriented change realization based on a two-level change type model. However, no specification change types have been proposed there.

Aspect-oriented programming has been identified as interesting and applicable in Android applications [8] and studies of technical issues in affecting applications in Android using aspects have been reported [22, 31] but these efforts cannot be considered to be organized application of aspect-oriented programming in the mobile application domain.

The SuperMod tool [27] aims at interconnecting version control changes with features in software product line development. This is similar to how aspect-oriented change realization interconnects implementation and specification level changes. However, in aspect-oriented change realization, changes remain operated upon directly at the programming language level with no need for additional tools.

Other applications of aspect-oriented programming, such as those in modifying compiled code [4, 5], use case driven modularization [6, 7, 16, 17] refactoring [24], creating programming languages [9, 25], or complex event processing [20, 21] could potentially benefit from aspect-oriented change realization in making these approaches closer to specification artifacts and, consequently, non-expert practitioners, such as end users.

7 CONCLUSIONS AND FURTHER WORK

In order to make the benefits of aspect-oriented change realization available in the mobile application domain, we explored the applicability of aspect-oriented change realization based on a two-level change type model and known change types originally devised for the web application domain for the mobile application domain. For this, we performed a study that involved a set of hypothetical scenarios and two real mobile applications for Android. The study revealed that the change types known from the web application domain are in their essence applicable to the mobile application domain. It also lead to the discovery of four new change types in the mobile application domain and the corresponding specification–implementation change type relationships along with further relationships between known change types. We discussed the problems and limitations and paid particular attention to the issue of change interaction.

By performing further studies, the catalog of change types could be enriched with new change types that would make applying aspect-oriented change realization available in the mobile application domain even more convenient. This could be supported further by expressing newly discovered change types at the modeling level [35]. It has been demonstrated that easier maintenance of aspects compensates for the initial effort necessary to develop

these aspects if they are not affecting the underlying structure, while otherwise, the effect is opposite [12]. We would like to explore this further by assessing the level of maintainability reached by aspect-oriented change realization using some of the available frameworks [1, 26, 28]. Also, we would like to explore how layered 3D visualization of software models [13–15] possibly employing virtual reality [32, 33] applied to aspect-oriented change realization could improve maintainability.

ACKNOWLEDGMENTS

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-15-0508, Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge ITMS 26240220039, co-funded by the ERDF, and internal grant scheme of the Slovak University of Technology in Bratislava in support of the teams preparing proposals for Horizon 2020.

REFERENCES

[1] W. Abdelmoez, Hatem Khater, and Noha El-shoafy. 2012. Comparing Maintainability Evolution of Object-Oriented and Aspect-Oriented Software Product Lines. In *Proceedings of 8th International Conference on INFOrmatics and Systems, INFOS2012, Advances in Software Engineering Track*. IEEE, Cairo, Egypt.

[2] Naveed Ali and Richard Lai. 2016. A Method of Requirements Change Management for Global Software Development. *Information and Software Technology* 70, C (2016), 49–67.

[3] Michal Bebjak, Valentino Vranić, and Peter Dolog. 2007. Evolution of Web Applications with Aspect-Oriented Design Patterns. In *Proceedings of ICWE 2007 Workshops, 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering, AEWSE 2007*. Como, Italy.

[4] Ilona Bluemke and Konrad Billewicz. 2008. Aspects in the Maintenance of Complied Programs. In *Proceedings of 3rd International Conference on Dependability of Computer Systems, DepCoS-RELCOMEX 2008*. IEEE, Szklarska Poręba, Poland.

[5] Ilona Bluemke and Konrad Billewicz. 2009. Aspect Modification of an EAR Application. In *Advanced Techniques in Computing Sciences and Software Engineering, Proceedings of 2008 International Conference on Systems, Computing Sciences and Software Engineering, SCSS 2008, part of International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, CIS^{2}E 08*. Springer, Krakow, Poland.

[6] Michal Bystrický and Valentino Vranić. 2015. Preserving Use Case Flows in Source Code. In *Proceedings of 4th Eastern European Regional Conference on the Engineering of Computer Based Systems, ECBS-EERC 2015*. IEEE Computer Society, Brno, Czech Republic.

[7] Michal Bystrický and Valentino Vranić. 2017. Preserving Use Case Flows in Source Code: Approach, Context, and Challenges. *Computer Science and Information Systems Journal (ComSIS)* 14, 2 (2017), 423–445.

[8] Fernando Cejas. 2014. Aspect-Oriented Programming in Andorid. https://fernandocejas.com/2014/08/03/aspect-oriented-programming-in-android/.

[9] Sergej Chodarev, Dominik Lakatoš, Jaroslav Porubän, and Ján Kollár. 2014. Abstract Syntax Driven Approach for Language Composition. *Central European Journal of Computer Science* 4, 3 (2014), 107–117.

[10] Catarina Costa and Leonardo Murta. 2013. Version Control in Distributed Software Development: A Systematic Mapping Study. In *2013 IEEE 8th International Conference on Global Software Engineering, ICGSE 2013*. IEEE, Bari, Italy.

[11] Peter Dolog, Valentino Vranić, and Mária Bieliková. 2001. Representing Change by Aspect. *ACM SIGPLAN Notices* 36, 12 (Dec. 2001), 77–83.

[12] Stefan Endrikat and Stefan Hanenberg. 2011. Is Aspect-Oriented Programming a Rewarding Investment into Future Code Changes? A Socio-technical Study on Development and Maintenance Time. In *Proceedings of 2011 IEEE 19th International Conference on Program Comprehension, ICPC 2011*. IEEE, Kingston, ON, Canada.

[13] Matej Ferenc, Ivan Polášek, and Juraj Vincúr. 2017. Collaborative Modeling and Visualisation of Software Systems Using Multidimensional UML. In *Proceedings of 5th IEEE Working Conference on Software Visualization, VISSOFT 2017*. IEEE, Shangai, China.

[14] Lukáš Gregorovič and Ivan Polášek. 2015. Analysis and Design of Object-Oriented Software Using Multidimensional UML. In *Proceedings of 15th International Conference on Knowledge Technologies and Data-Driven Business*. ACM, Graz, Austria.

[15] Lukáš Gregorovič, Ivan Polášek, and Branislav Sobota. 2015. Software Model Creation with Multidimensional UML. In *Proceedings of 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, part of WCC 2015 (LNCS 9357)*. Springer, Daejeon, Korea.

[16] Ivar Jacobson. 2003. Use Cases and Aspects – Working Seamlessly Together. *Journal of Object Technology* 2, 4 (2003).

[17] Ivar Jacobson and Pan-Wei Ng. 2004. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley.

[18] Amin Jalali and Ilia Bider. 2014. Towards Aspect Oriented Adaptive Case Management. In *Proceedings of 2014 IEEE 18th International Enterprise Distributed Object Computing Conference, EDOCW 2014, Workshops and Demonstrations*. IEEE, Ulm, Germany.

[19] Ramnivas Laddad. 2010. *AspectJ in Action: Enterprise AOP with Spring Applications* (second ed.). Manning.

[20] Ján Lang, Michal Jantošovič, and Ivan Polášek. 2012. Re-Usability in Complex Event Pattern Monitoring. In *Proceedings of IEEE 10th Jubilee International Symposium on Aplied Machine Intelligence and Informatics, SAMI 2012*. IEEE, Herľany, Slovakia.

[21] Ján Lang and Ján Jánik. 2013. Reactive Distributed System Modeling Supported by Complex Event Processing. In *Proceedings of 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems, ECBS-EERC 2013*. IEEE Computer Society, Budapest, Hungary.

[22] Ivan Martoš and Valentino Vranić. 2017. Affecting Applications in Android Using Aspects. In *Proceedings of 2015 IEEE 1st International Workshop on Consumer Electronics, 2015 CE WS*. IEEE, Novi Sad, Serbia.

[23] Russell Miles. 2004. *AspectJ Cookbook*. O'Reilly.

[24] Roman Pipík and Ivan Polášek. 2013. Semi-Automatic Refactoring to Aspect-Oriented Platform. In *Proceedings of 14th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2013*. IEEE, Budapest, Hungary.

[25] Jaroslav Porubän, Miroslav Sabo, Ján Kollár, and Marjan Mernik. 2010. Abstract Syntax Driven Language Development: Defining Language Semantics Through Aspects. In *Proceedings of International Workshop on Formalization of Modeling Languages, FML '10, part of ECOOP 2010*. ACM, Maribor, Slovenia.

[26] Claudio Sant'anna, Alessandro Garcia, Christina Chavez, Carlos Lucena, and Arndt v. von Staa. 2003. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In *Proceedings of XVII Brazilian Symposium on Software Engineering*. Manaus, Brazil.

[27] Felix Schwägerl, Thomas Buchmann, and Bernhard Westfechtel. 2015. SuperMod—A Model-Driven Tool that Combines Version Control and Software Product Line Engineering. In *2015 10th International Joint Conference on Software Technologies, ICSOFT 2015*. IEEE, Colmar, France.

[28] Ananthi Sheshasaayee and Robi Jose. 2015. A Theoretical Framework for the Maintainability Model of Aspect Oriented Systems. *Procedia Computer Science* 62 (2015), 505–512. Proceedings of 2015 International Conference on Soft Computing and Software Engineering, SCSE 2015.

[29] Jessada Tomyim and Amnart Pohthong. 2016. Requirements Change Management Based on Object-Oriented Software Engineering with Unified Modeling Language. In *2016 7th IEEE International Conference on Software Engineering and Service Science, ICSESS 2016*. IEEE, Beijing, China.

[30] Ľuboš Staráček and Valentino Vranić. 2014. MDA Based Multiplatform Mobile Application Modeling with Platform Compliant User Interfaces. *INFOCOMP Journal of Computer Science* 13, 2 (2014), 34–43.

[31] Erik Šuta, Ivan Martoš, and Valentino Vranić. 2017. Usability of AspectJ from the Performance Perspective. In *Proceedings of 2015 IEEE 1st International Workshop on Consumer Electronics, 2015 CE WS*. IEEE, Novi Sad, Serbia.

[32] Juraj Vincúr, Pavol Návrat, and Ivan Polášek. 2017. VR City: Software Analysis in Virtual Reality Environment. In *IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*. IEEE, Prague, Czech Republic.

[33] Juraj Vincúr, Ivan Polášek, and Pavol Návrat. 2017. Searching and Exploring Software Repositories in Virtual Reality. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology, VRST 2017*. ACM, Gothenburg, Sweden.

[34] Valentino Vranić, Michal Bebjak, Radoslav Menkyna, and Peter Dolog. 2011. Developing Applications with Aspect-Oriented Change Realization. In *Proceedings of 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2008, Revised Selected Papers (LNCS 4980)*. Springer, Brno, Czech Republic.

[35] Valentino Vranić and Branislav Kuliha. 2015. Realizing Changes by Aspects at the Design Level. In *Proceedings of IEEE 19th International Conference on Intelligent Engineering Systems, INES 2015*. IEEE, Bratislava, Slovakia.

[36] Valentino Vranić, Radoslav Menkyna, Michal Bebjak, and Peter Dolog. 2009. Aspect-Oriented Change Realizations and Their Interaction. *e-Informatica Software Engineering Journal* 3, 1 (2009), 43–58.