International Journal on Advanced Science Engineering Information Technology

Vol.8 (2018) No. 6
ISSN: 2088-5334

Blocks Correctness Evaluation Methodology for Block-Based Software Development

Abdullah Mohd Zin[#1], Mustafa Almatary[#2], Marini Abu Bakar[#4], Rodziah Latih[#5], Norleyza Jailani[#6]

[#]*Center for Software Technology and Management, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600 Bangi Selangor, Malaysia*
E-mail: [1]amzftsm@ukm.edu.my, [2]vim4mustafa@yahoo.com, [3]marini@ukm.edu.my, [4]rodziah.latih@ukm.edu.my, [5]njailani@ukm.edu.my

***Abstract*— The term "block" in Block-Based Software Development (BBSD) refers to a software component that has the characteristics of reusable, composition, customizable and configurable. Based on the principles of component-based software development and end-user development, the objective of BBSD is to allow non-programmer known as end-user to build a new application by using a set of blocks by creating composite blocks, configuring and customizing for a specific application domain. In the current implementation, a Domain Initiator is responsible for identifying blocks' specifications, which will be uploaded to the block store repository. Block developers can contribute to developing blocks using the Java programming language. Blocks for a specific domain are bundled as a JAR file. These blocks will be stored in a block store. The block store is a software repository that provides a sharing mechanism for domain driven blocks specification, cataloging, archiving, and distribution. Before the blocks submitted to the block store can be distributed to end-users, they are required to undergo the process of block verification and evaluation to ensure that they conform to the requirement specification. The submitted blocks will also need to be approved by the domain initiator before they are made available to the end users. This paper proposes the block-based evaluation methodology as well as the software tool which helps domain initiator in the process of blocks verification and evaluation. The proposed methodology consists of three types of validation namely Automatic Validation Approach, JSR-303 or JSR-349 standard bean Validation Specification, and the manual testing. The proposed methodology itself was verified through a case study using a list of blocks submitted to the block store repository.**

***Keywords*— software reuse repository; end user development; block-based software development; component-based software development; component evaluation.**

I. INTRODUCTION

The block store repository is a domain driven software blocks sharing mechanism to support the Block-Based Software Development (BBSD). BBSD is a software development approach based on the principles of Component-Based Software Development (CBSD) and End-User Development (EUD) [1]. The main objective of BBSD is to allow end user programmers to develop applications by integrating blocks. End user programmers are software developers who are not trained as professional programmers, such as teachers, accountants, scientists, engineers and parents.

Within the context of BBSD, the term "block" refers to a software component that can be reused, highly composable, customizable and configurable. Blocks can be combined with other blocks to form an application without going through the normal coding process [2].

Apart from end user programmers, there are four other actors in BBSD as shown in Fig. 1. These actors are administrator, visitors, domain initiators and block developers. Administrator is a person responsible for managing the block store. Administrator is responsible for managing users accounts, creating of domains/subdomains, managing users profiles, authentication information and handling communication with all users through inbox messaging. Domain initiator is responsible for identifying a new application domain, creating sub-domains and then identifing blocks required for that particular domain. Block developers are professional programmers who are responsible for the blocks development.

A number of tools and methodologies have been developed to support the BBSD. Two of the methodologies are Blocks Identification Methodology and Block Creation Methodology. Tools that have been developed include Blocks Creation Tool [3] and Blocks Integration Tool [4]. Blocks Creation Tool helps block developers to develop blocks while Blocks Integration Tool helps end user programmers to integrate blocks.

Blocks submitted to the block store repository by block developers need to be managed and verified by project initiator, before they can be published and distributed. This paper describes a methodology and software tools that can be used in the evaluation and verification of software blocks. The proposed methodology consists of three types of validation: Automatic Validation Approach, JSR-303 or JSR-349 standard bean Validation Specification, and the manual testing. The proposed methodology is then validated through a case study on a list of blocks submitted to the block store repository.

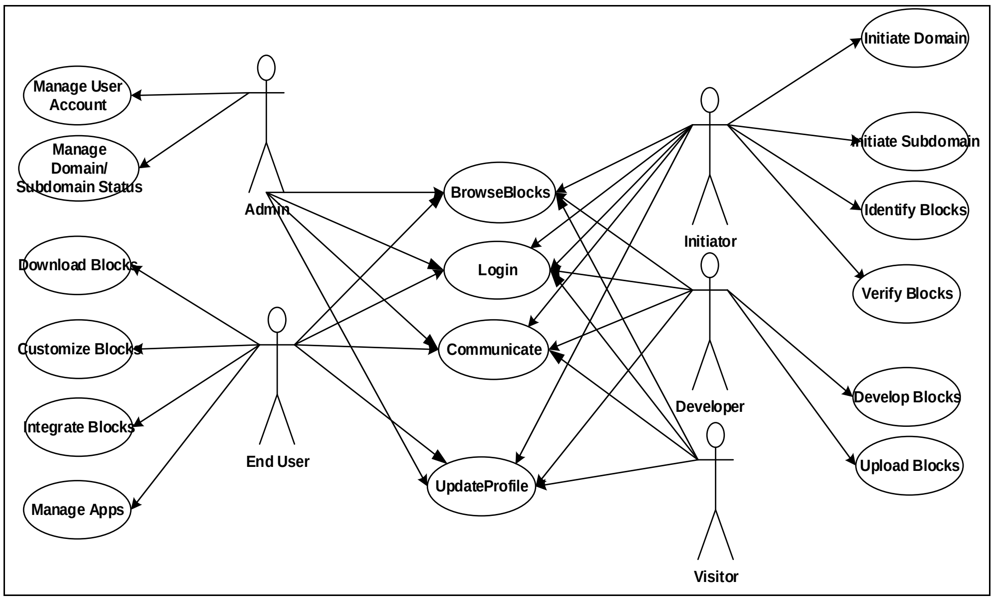


Fig. 1 The Block Store Use Case diagram

II. Material And Method

A block is basically a software component. Currently a block is implemented by using JavaBeans technology. Thus block evaluation is related to software component evaluation. In the following subsections, we will first describe works that have been carried out in component evaluation in general and JavaBeans evaluation in particular. The third subsection describes our proposed methodology for blocks evaluation.

A. Component Evaluation

Component evaluation is performed in order to find the best component that fit a given task and to certify properties of the component [5]. The evaluation can be done in one of the three stages: during requirement analysis, design and implementation, or deployment [6]. However, Alvaro et al [7] proposes that component evaluation can only be done during certification and selection.

Component evaluation is performed based on certain goals. Thus, the mechanism, methods and type of validation is determined based on these goals. Some of the goals for components evaluation are regarding components security, performance, usability, reusability [8, 9] and maintainability [10, 11]. Research in software components evaluation is still immature and further research is required to develop techniques, methods, processes and tools [12].

B. JavaBean Validation

The JavaBean become more popular in recent years since JavaBeans specifications and conventions made it easier to implement changes to properties through setter/getter methods. In order to ensure that properties in JavaBeans have the right values in them, Java Bean Validation (JSR-303) was introduced and approved by Java Community Process (JCP) in Nov 2009. Java Bean Validation 1.1 (JSR-349) is an improved version of JSR-303 and was released in May 2013. Both JSR-303 and JSR-349 specification have made a clear imprint to standardize the dynamic validation among different providers and open the gate for a custom constraints design and implementation. Most of the frameworks for implementing JSR-303 and JSR-349 involve the use of annotations since annotations are easy to use, create and add clarity to the code, and they also provide good type safety and increase reusability [13]. However, this kind of validation is only suitable for a runtime validation and commonly used for data entry validation.

An automatic documentation annotation also can be realized on data sharing inside a program itself [14]. Simultaneously, software engineers are allowed to program the same style used previously. However, the annotations have been used as semantics validation and specification technique. In addition, the JML is designed to specify java modules and tools created to allow users to view the specifications in a convenient documentation manner, such as JMLDoc, javadoc-like, doc++, Doclet, and iDoclet [15].

The validation and evaluation surpass the syntax and semantics of data content to component compatibility issues, especially in BBSD, to help end user programmers compose blocks to form an application. A number of frameworks implement JSR-303 and JSR-349, such as JAX-RS, JAXB, JPA, CDI, Wicket, Spring, and Jface. However, these frameworks are mainly designed to work with JavaBeans using Plain Old Java Object (POJO). Strong assumptions can be made on the type of applications that can utilize the frameworks. In addition, these frameworks should be easy to integrate with any Java project [13].

C. Proposed Methodology

A block is a type of single layer component with several characteristics identified in requirement specification documents. The specification identifies the attributes and behaviours to be verified. The list of behaviours and attributes, such as block input and output attributes, and list of behaviours/methods required are identified in the specification documents. To gain more clarity and to identify the main specification of blocks, we need to emphasize the main characteristics of the disparity between blocks and common components. These characteristics are mainly based on the interfacing and communication among the blocks and other disparities. Differences between blocks and components are shown in Table I.

TABLE I.
DIFFERENCES BETWEEN BLOCKS AND COMPONENTS

Component	Block
Communicate directly with another component	Cannot communicate directly with each other
They need to be designed to fit with a desired environment.	The interface designed to be more flexible to adapt the plugged block.
Complicated (can have nested component)	Single layer type of component
Can act as a complete system	Need to be composed with another block
Can handle more than one intersection process	It complete a single task (no tasks intersection)
Required and provide directly affect the processed result.	No result processed through different blocks
Required and provided result may differ from one to another.	Required and provided result should be standardized for all blocks (exp 0,1,…n)

Blocks are more independent in design and implementation. These blocks have nothing to share with each other directly other than through a connector. A connector is just a piece of code that handles the sequence of blocks execution. Three types of connectors are available: sequential, alternative, and random.

Table II shows the main specifications of block and connector that need to be evaluated during blocks verification. Blocks specifications consist of block number, block type, input, and output. The block type is required to determine the connector type. The input of the block is required value, while the output represents the provided value. The specification for a connector includes connector type, number of blocks, connector number, and connector type.

The block store repository is a distribution mechanism to support the BBSD. Fig. 2 illustrates the block evaluation environment where the block store plays the main role. Domain Initiator is responsible for identifying blocks specification that is then put into the block store repository. A block developer obtains the specification and then submits the developed blocks into the block store repository. The submitted blocks need to be approved by the domain initiator before they are made available to the end users.

The proposed method for evaluating blocks in the block store repository is shown in Fig. 3. It involves three types of evaluation: (i) Type 1: the validation of the standard block specification, (ii) Type 2: the runtime validation by using Standard JSR, and (iii) Type 3: manual method.

TABLE II
LIST OF COMMON BLOCK SPECIFICATIONS

Block Specification			
	Type	**Methods**	**Description**
Attributes	Input	`getRequiredIn()`	Get the block input
	Output	`getProvideOut()`	Get the block output
	Block	`getBlockType()`	Get the block type
	connector	`getConnectorType()`	Get the connector type
Behaviours	Property method	`CheckProList()`	Get the list of properties of the block
	Action method	`getListofEvent-Method()`	The list of methods handle the actions
	Task method	`getListofTask-Method()`	The list of methods achieves some tasks

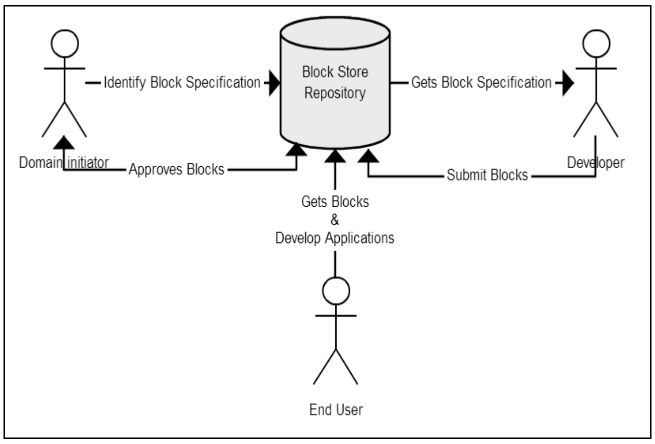


Fig. 2. Block evaluation environment

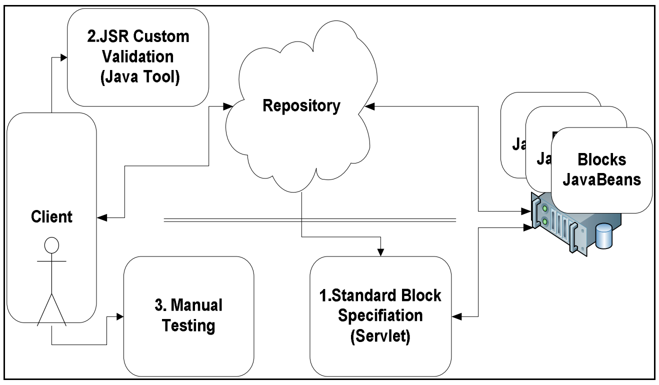

2.JSR Custom Validation (Java Tool)
Repository
Blocks JavaBeans
Client
3. Manual Testing
1.Standard Block Specification (Servlet)

Fig. 3. Block evaluation processes

The first type of evaluation involves the validation of the standard block specification. An example of the standard block specification is shown in Table III. The evaluation of the standard block specification involves validating the main block specifications implemented in the submitted block. It also checks whether all attributes and behaviors identified in a block specification are implemented in the submitted block.

TABLE III
LOGIN BLOCK REQUIREMENT SPECIFICATION

Block Name	**Login Block**
Block Id	B-E-C-10001
Contract Type	Sequential
Actors	User
Attributes	Block ID, Contract Type, Block Input, Block Output
Behaviors	Color, Font, Message, Main Label.
Use Cases	Login.
Remark	The user can change the color, font, main label, and response message

In the implementation of a block, the standard block specification should be grouped in an interface that implements the main attributes and behaviors, as shown in the following code:

```
/** The Standard Block Specifications interface */
package specs;
public interface BlockSpecs
{
  int getBlockID();
  int getBlockContractType();
  int getBlockInput();
  int getBlockOutput();
} // End of BlockSpecs interface
```

Since each block has different behaviours and attributes, each of these behaviours and attributes need to be properly specified. Examples of behaviours and attributes for some blocks are given in Table IV.

The validation of the standard block specification is achieved by using the following steps:

1. Check that the JAR JavaBean main class has implemented the interface BlockSpecs:
   This step will ensure that all methods returning the main specification are implemented, which can be achieved using the following code grouped into two classes: class finder and JAR Manager.
2. Check that all attributes and behaviors of the block specifications are implemented:
   This step will check whether the developed block consists of all the behaviors listed in the specification docs. It involves two sub-steps: parsing through the JAR file followed by identifying all methods and attributes that have been implemented. These methods are then compared with required specifications if available or missing is displayed.

TABLE IV
EXAMPLES OF BEHAVIOURS AND ATTRIBUTES OF DIFFERENT BLOCKS

Block Name	**Behaviors**	**Attributes**
Login	Color, Font, Message, Main Label	`BackID, ContractType, BlockInpput, BlockOutpout`
Capture Deals	Change printer, change text color, change tax schema, and switch to invoice option.	`BackID, ContractType, BlockInpput, BlockOutpout`
Manage Order	Change text color, change background color, change order source, and change customer info	`BackID, ContractType, BlockInpput, BlockOutpout`
Process Payment	Change payment method, change text color, and switch to offline payment	`BackID, ContractType, BlockInpput, BlockOutpout`
Manage Stock	Change text color, change background color, report product shortage, and report nearly expired product.	`BackID, ContractType, BlockInpput, BlockOutpout`

The second type of evaluation is the validation of the custom block specification to ensure that properties in a block have the right values in them. In order to do this type of validation, blocks need to be annotated by using annotation method based on the standard JSR-303 & JSR-349. An example of how to design a custom block specification validator is shown in the following code:

```
/*  @author Mostafa    */
@Target({METHOD, FIELD,ANNOTATION_TYPE})
@Retention(RUNTIME)
@Constraint(validatedBy = BlockProValidator.class)
@Documented
public @interface BlockPro
{
   String message() default "{validator.blockpro}";
   Class<?>[] groups() default {};
   Class<? extends Payload>[] payload() default {};
}
```

The `@interface` keyword is used to define an annotation type where the attributes of an annotation type are declared in a method. According to the API specifications, any constraint annotation defines an attribute "message" that returns the default key for creating custom error messages. In cases where the constraint is violated, an attribute "groups" allows the specification of validation groups to which this constraint belongs. In addition, the annotation type Meta annotations [13] specifies the class name validator to be used for validating elements annotated with `@Documented`.

The third type of evaluation is the manual evaluation where a block is tested for functionality and specified features.

D. Software Tool

For the first type of evaluation, a specific software tool has been designed and implemented. Fig. 4 shows the validation service structure used to verify the submitted blocks into the block repository and illustrates main classes used to validate whether the Block Specification interface is implemented. If the evaluated block has not implemented the required properties, the evaluation mechanism should display the missing properties. Moreover, the approval link shall not be enabled and the block will not be available for selection by end user programmers.

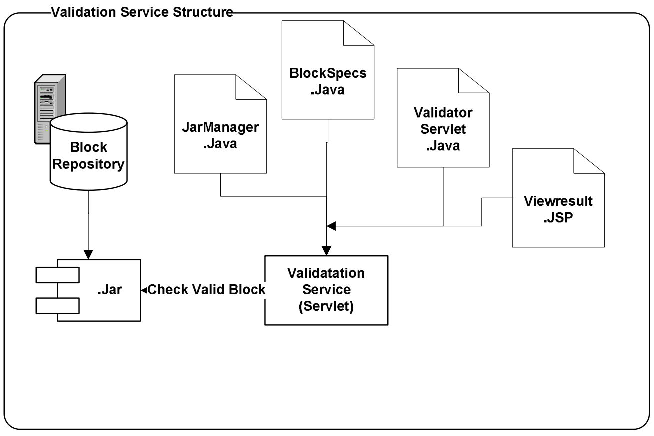


Fig. 4. Block verification architecture

The second type of evaluation is supported by NetBeans. The steps for carrying out the evaluation are listed as follows:

Step 1: Create an application by using NetBeans IDE.
Step 2: Create the block class. The source code should be provided.
Step 3: Create the custom validator if needed.
Step 4: Generate the test cases automatically.
Step 5: Run/compile the targeted test.
Step 6: View the test report.

The third type of evaluation is carried out by executing the block. The output of the execution is manually checked to ensure its correctness.

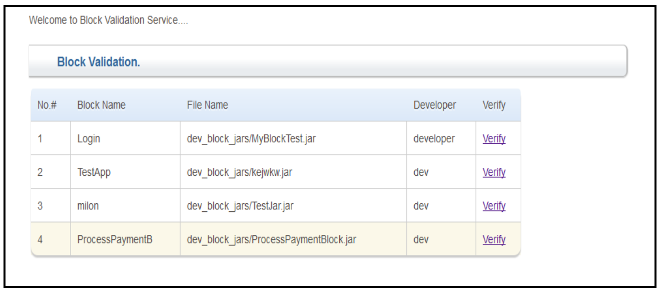
III. Results And Discussion

In this section, we show how the process of block evaluation is carried out. For the purpose of the discussion, we choose a "ProcessPayment" block that has been identified as one of the blocks needed to support online business transaction [16]. The specification of the block is given in Table V.

The first type of evaluation is done by using the software tool. To verify the block, we have to select the block to be verified and click "verify" as shown in Fig. 5. The block is then evaluated against the requirements analyzed in the block specification. The implemented behaviour is stated as "Available" while the behaviour that is not implemented is stated as "Missing", as shown in Fig. 6. In this example "textMethod" and "paymentMethod" are available while "offlineMode" is missing.

TABLE V
PROCESS PAYMENT BLOCK REQUIREMENT SPECIFICATION

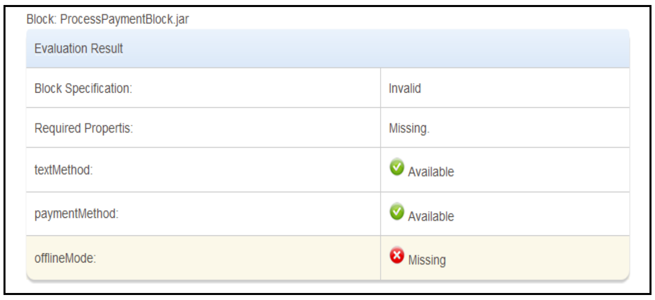
Block Name	**Process Payment**
Block Id	B-E-C-10002
Contract Type	Alternative
Actors	Seller / Customer
Attributes	BackID, ContractType
Behaviors	Change payment method, change text color, and switch to offline payment.
Use Cases	Validate payment, process payment.
Remark	The switch to offline payment should be enabled at run time.


Welcome to Block Validation Service....

Block Validation.

No.#	Block Name	File Name	Developer	Verify
1	Login	dev_block_jars/MyBlockTest.jar	developer	Verify
2	TestApp	dev_block_jars/kejwkw.jar	dev	Verify
3	milon	dev_block_jars/TestJar.jar	dev	Verify
4	ProcessPaymentB	dev_block_jars/ProcessPaymentBlock.jar	dev	Verify

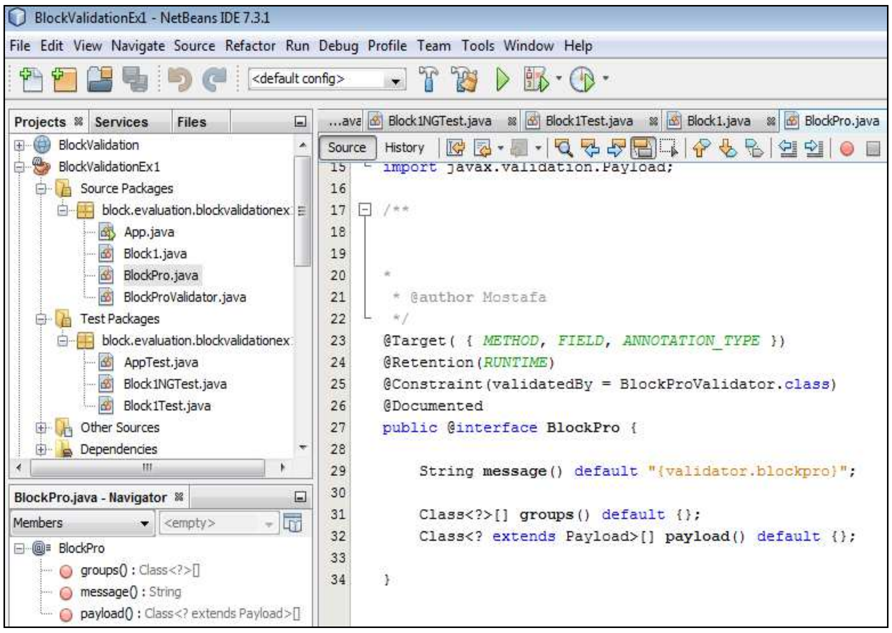
Fig. 5. List of blocks to be verified


Block: ProcessPaymentBlock.jar

Evaluation Result	
Block Specification:	Invalid
Required Propertis:	Missing.
textMethod:	✓ Available
paymentMethod:	✓ Available
offlineMode:	✖ Missing

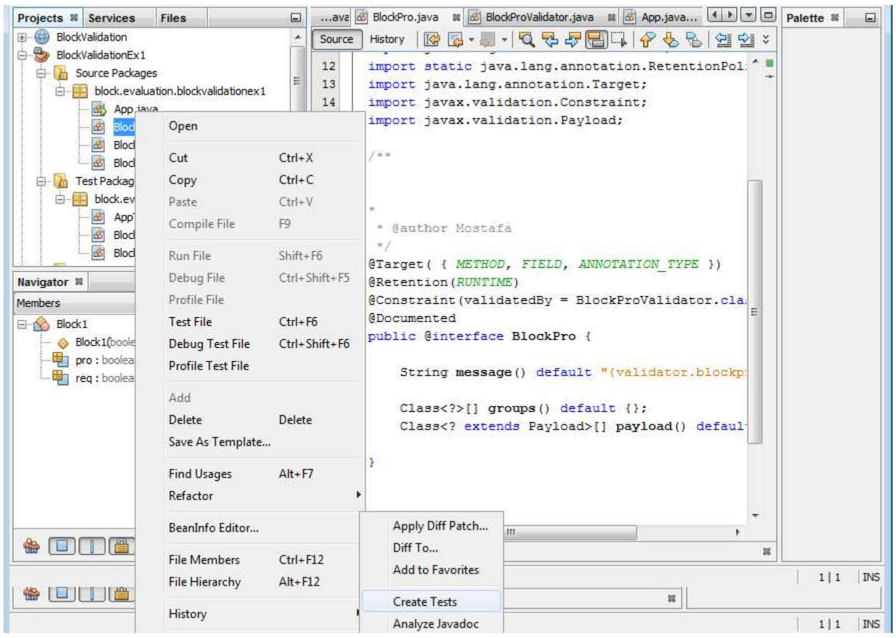
Fig. 6. Result of some specifications implemented

The second type of evaluation is carried out by using six steps described earlier. The custom validator is created as shown in Fig. 7. The test cases are generated automatically as shown in Fig. 8. The result of the evaluation is shown in Fig. 9.


BlockValidationEx1 - NetBeans IDE 7.3.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects Services Files
BlockValidation
BlockValidationEx1
Source Packages
block.evaluation.blockvalidationex
App.java
Block1.java
BlockPro.java
BlockProValidator.java
Test Packages
block.evaluation.blockvalidationex
AppTest.java
Block1NGTest.java
Block1Test.java
Other Sources
Dependencies
BlockPro.java - Navigator
Members
<empty>
BlockPro
groups() : Class<?>[]
message() : String
payload() : Class<? extends Payload>[]
...ava Block1NGTest.java Block1Test.java Block1.java BlockPro.java
Source History

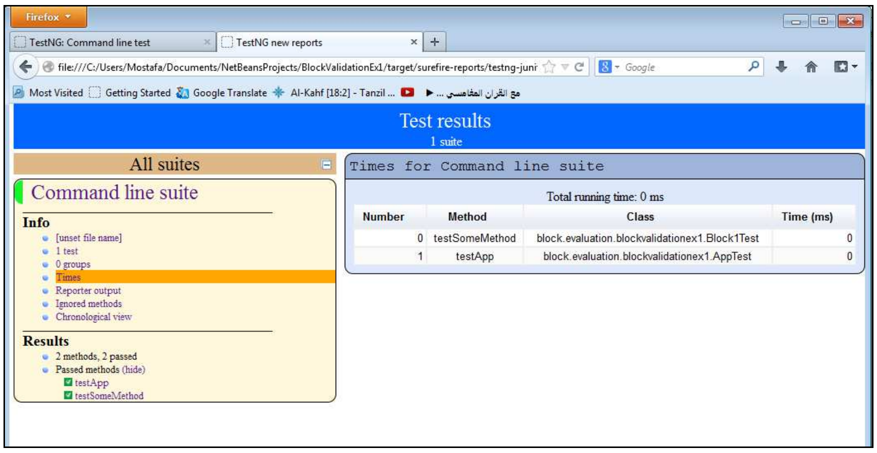
```
import javax.validation.Payload;

/**
 *
 * @author Mostafa
 */
@Target( { METHOD, FIELD, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Constraint(validatedBy = BlockProValidator.class)
@Documented
public @interface BlockPro {

    String message() default "{validator.blockpro}";

    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};

}
```

Fig. 7. Custom JSR-303 annotation specification validator



```
import static java.lang.annotation.RetentionPol
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;

/**
 *
 * @author Mostafa
 */
@Target( { METHOD, FIELD, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Constraint(validatedBy = BlockProValidator.cla
@Documented
public @interface BlockPro {

    String message() default "{validator.blockp

    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() defaul

}
```

Fig. 8. Create an automatic test


Firefox
TestNG: Command line test
TestNG new reports
file:///C:/Users/Mostafa/Documents/NetBeansProjects/BlockValidationEx1/target/surefire-reports/testng-juni
Google
Most Visited Getting Started Google Translate Al-Kahf [18:2] - Tanzil ... مع القرآن المفاهيمي ...

Test results
1 suite

All suites

Command line suite

Info
- [unset file name]
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results
- 2 methods, 2 passed
- Passed methods (hide)
  - testApp
  - testSomeMethod

Times for Command line suite

Total running time: 0 ms

Number	Method	Class	Time (ms)
0	testSomeMethod	block.evaluation.blockvalidationex1.Block1Test	0
1	testApp	block.evaluation.blockvalidationex1.AppTest	0

Fig. 9. Test report result

Fig. 10 illustrates the user interface for the "ProcessPayment" block. The third type of evaluation is done to determine the correctness of the block. The evaluation is done by running the block "ProcessPayment" and check that the block has implemented all required properties.

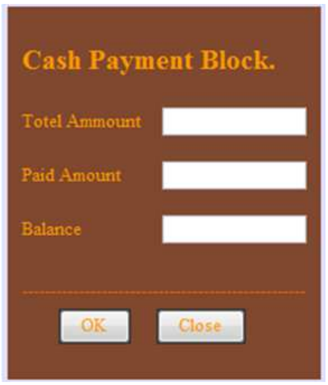


Fig. 10. Result of complete block specifications

IV. Conclusion

Block-Based Software Development (BBSD) approach offers a software development environment that enables end user programmers to make use of the available blocks to develop applications. The Block Store repository is a place where blocks are distributed [17]. The main purpose of the block store is to enables software developers to share and distribute blocks and to allow end users to browse and select required blocks. The availability of the block store enables end user programmers to develop applications that satisfy their requirements.

The BBSD is supported by two methodologies: the block identification methodology and block creation methodology. This paper describes the third methodology needed for BBSD: block correctness evaluation methodology.

There are three types of validation that need to be carried out before a block can be considered acceptable and can then be put into the block store repository. The three types of evaluation are (i) Standard blocks specification validation, (ii) Custom block validation and (iii) manual testing techniques.

This paper has shown the feasibility of the evaluation methodology through a case study. Finally, the verified blocks for a particular subdomain have been approved for distribution and unqualified blocks a fault report is generated for developers to be corrected.

Acknowledgment

We would like to thank Universiti Kebangsaan Malaysia for supporting this work through its research grant fund UKM-GUP-TMK-07-01-032

References

[1] A.M. Zin, "Block-Based Approach for End User Software Development". Asian Journal of Information Technology, 10(6), pp. 249–258, 2011.

[2] S. N. H. Mohamad, A. Patel, Y. Tew, R. Latih, and Q. Qassim, "Principles and Dynamics of Block-based Programming Approach", pp. 340–345, 2011. DOI: 10.1109/ISCI.2011.5958938

[3] M. Djasmir, S. Idris, M.A. Bakar, and A.M. Zin, "An Integrated Development Environment for Blocks Creation". Asian Journal of Information Technology, 11(6), pp. 194–200, 2012. DOI: 10.3923/ajit.2012.194.200

[4] S.N. Sarif, S. Idris, and A.M. Zin, "The Design of Blocks Integration Tool to Support End-User Programming". In 2011 International Conference on Electrical Engineering and Informatics. pp. 1-5, 2011. DOI: 10.1109/ICEEI.2011.6021657

[5] T. Vale, I. Crnkovic, E.S.d. Almeida, P. A. da M. S. Neto, Y. C. Cavalcanti, S.R.d.L. Meira, "Twenty-eight years of component-based software engineering", The Journal of Systems and Software, vol.111, pp.128–14, 2016. http://dx.doi.org/10.1016/j.jss.2015.09.019

[6] I. Crnkovic, M. Chaudron, and S. Larsson, "Component-Based Development Process and Component Lifecycle". In International Conference on Software Engineering Advances, pp. 44–44, 2006. DOI: 10.1109/ICSEA.2006.261300

[7] A. Alvaro, R. Land, and I. Crnkovic, "Software Component Evaluation: A Theoretical Study on Component Selection and Certification". MRTC report. Mälardalen Real-Time Research Centre, Mälardalen University, 2007. ISRN: MDH-MRTC-217/2007-1-SE

[8] A.P. Singh, and P. Tomar, "Rule-based fuzzy model for reusability measurement of a software component". International Journal of Computer Aided Engineering and Technology, 9(4), 2017. DOI: 10.1504/IJCAET.2017.086932

[9] M. Tahir, F. Khan, M. Babar, F. Arif, and S. Khan, "Framework for Better Reusability in Component Based Software Engineering". Journal of Applied Environmental and Biological Sciences, 6(4S), pp. 77-81, 2016.

[10] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek and S. Kounev, "Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-Based Architectures," *IEEE Transactions on Software Engineering*, vol. 41(2), pp.157-175, 2015. DOI: 10.1109/TSE.2014.2362755

[11] de AG Saraiva, J., De França, M. S., Soares, S. C., Fernando Filho, J. C. L., & de Souza, R. M., "Classifying metrics for assessing object-oriented software maintainability: A family of metrics' catalogs". Journal of Systems and Software, 103, pp. 85-101, 2015.

[12] A. Tiwari and P. S. Chakraborty, "Software Component Quality Characteristics Model for Component Based Software Engineering," *2015 IEEE International Conference on Computational Intelligence & Communication Technology*, pp. 47-51, 2015. DOI: 10.1109/CICT.2015.40

[13] de Siqueira J.L., Silveira F.F., Guerra E.M., "An Approach for Code Annotation Validation with Metadata Location Transparency". In: Gervasi O. et al. (eds) Computational Science and Its Applications -- ICCSA 2016. Lecture Notes in Computer Science, Springer, Cham, 2016, vol. 9789.

[14] M. Sulír and M. Nosál', "Sharing developers' mental models through source code annotations," *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 997-1006, 2015. doi: 10.15439/2015F301

[15] Donthala, Arjun Mitra Reddy, "Design of a JMLdoclet for JMLdoc in OpenJML". *Electronic Theses and Dissertations*. 5132, 2016 Retrieved from http://stars.library.ucf.edu/etd/5132

[16] M. Almatary, M.A. Bakar, and A.M. Zin, "Block Identification Methodology: Case Study on Business Domain". Journal of Theoretical and Applied Information Technology, 60(1), pp.47–54, 2014. http://www.jatit.org/volumes/Vol60No1/sixtyth_1_2014.php

[17] M. Almatary, M.A. Bakar, and A.M. Zin, "The Block Store of Block-Based Programming Approach". Journal of Theoretical & Applied Information Technology, 60(2), pp. 237–244, 2014. http://www.jatit.org/volumes/Vol60No2/sixtyth_2_2014.php.