Automatic Code Review for SmartThings application using Static Analysis

Janine Cassandra Son¹ Byeong-Mo Chang² Kwanghoon Choi³
¹²Division of Computer Science, Sookmyung Women’s University
³Dept. of Electronics and Computer Engineering, Chonnam National University

Abstract

SmartApps are IoT applications that run in the cloud through SmartThings hub and are bounded by the features available in the SmartThings environment. SmartThings has provided documentation for the purpose of code review of SmartApps. Instead of manual code review, violations of the specified rules can be detected automatically through static analysis tools. Automatic code review through a rule-based static analysis tool can also be used to produce metrics to evaluate the characteristics of SmartApps. This study aims to automate the code review process based on the specifications provided by SmartThings and express it as metrics for a measureable evaluation of SmartApp characteristics.

1. Introduction

Interest in the Internet of Things (IoT) application development is continuously growing along with the development of smart homes, devices, and other automations. IoT source code differs in certain ways from general program source code [1]. SmartThings application, called SmartApp, in particular has a different structure compared to regular applications. The attributes unique to SmartApps and default attributes of programming languages can be used to identify metrics measureable through static analysis. The metrics are developed based on the guidelines provided by SmartThings that can be implemented using a rule-based static analysis tool.

This study aims to automate code review of SmartApps through static analysis. Traditional code review process is usually done manually by a team but studies conducted have shown that this process can be automated using static analysis of source code [2]. Although it cannot fully automate the code review process because of certain limitations, it allows faster and more efficient way of analyzing source codes compared to manual code review. After all, code review and static analysis tools both serve the same purpose: to detect errors and violations.

The rules provided in the SmartApp code review guidelines can be implemented in CodeNarc¹, which is a rule-based static analysis tool for Groovy². The number of violations for each rule can be expressed as metrics, which can be used to evaluate the characteristics unique to SmartApps.

This system can be divided into two parts: static analysis and evaluation tool. The static analysis information will be the input to the evaluation tool, which will be developed.

In this paper, we designed and implemented a static analysis tool of SmartApps for automated code review based on CodeNarc. We also show some metrics from the analysis to evaluate characteristics of SmartApps.

2. SmartApp Code Review Guidelines and Best Practices

SmartThings has provided documentation such as Code Review Guidelines and Best Practices and they contain rules on how to develop applications correctly for personal use or for distribution. SmartApps are written using a restricted subset of Groovy programming language [3]. The unique structure of SmartApps implies that there are rules applicable to SmartApps but not to general Groovy programs. Fig. 1 shows a typical structure of a SmartApp.

Fig. 1. SmartApp structure

¹http://codenarc.sourceforge.net
²http://groovy-lang.org/
One of the use cases for smart applications is to schedule a job to run on a specific schedule. Fig. 2 shows a violation of a rule from code review guidelines and best practices. Avoid chained runIn() calls involves the use of runIn() method which executes a specified handler method after a given number of seconds have elapsed. It states that chained runIn calls must be avoided since it is prone to failure. When a scheduled execution in handler() fails, it will not be able to reschedule itself thus, causing the whole chain to collapse. Instead, a predefined scheduling function such as runEvery5Minutes(), must be used to specify a recurring schedule [3].

```groovy
def initialize() {
    runIn(60, handler)
}

def handler() {
    // do something here
    // schedule to run again in one minute - this is an antipattern
    runIn(60, handler)
}
```

Fig. 2. Sample rule violation – AvoidChainedRunInCall

However, not all of the guidelines can be implemented in CodeNarc due to some limitations. For example, Use Groovy truth correctly states that the code must be consistent with what Groovy considers true and false. An example is “Empty strings are considered false; non-empty strings are considered true”. This rule is impossible to implement using static analysis since we do not know exactly the intention of the programmer for writing that code. Therefore, this type of rule is beyond the capacity of static analysis because coordination with the developer is needed in order to know the real purpose of the code.

3. Characteristics of SmartApps

SmartApps possess some characteristics which make them different from conventional programs. The metrics which we will define can be categorized under these characteristics.

1) Sandboxed Groovy environment: SmartApps are developed in a restricted form of Groovy which means that creation of new classes or calling certain methods are not allowed, among other restrictions. Also, it includes predefined functions as part of the SmartThings standard environment. Most of the necessary function calls for developing a smart application have already been provided by SmartThings and they are available in the documentation [3]. They can be invoked right away as they are already built in with the framework.

2) Subscription model: The most common type of SmartApp is an Event Handler Smart App. It operates using a subscription model which allows devices to subscribe to some Event and take action when the Event happens. Subscriptions are declared in the predefined callbacks section and they invoke the event handlers (see Fig. 1).

3) External system access (WebService and other APIs): SmartApps may need to call external web services. WebService SmartApps allow exposure to Web service endpoints and requests from external applications using an authentication service [4].

4) Default programming language characteristics: Since SmartApps are written using Groovy, default attributes of the Groovy programming language such as basic rules and style conventions may also be applied to SmartApps.

4. Static Analysis using CodeNarc

Static analysis is closely related with code review or inspection since they have the same purpose: to detect software defects without executing it [2]. This study utilizes CodeNarc, a static analysis tool for analyzing Groovy code that checks violations based on over 300 defined rules. It is customizable with available plugins that can run on major IDEs. However, CodeNarc with its default rules is not suitable enough for SmartApps since the latter has a different structure compared to general applications. Therefore, new rules must be added to CodeNarc so it will be appropriate for checking rule violations based on the documentation by SmartThings.

```
override
void visitMethodDef(Expression.MethodDef methodDef) {
    if (methodDefprecated || methodDef.name == "runIn";
        runArguments: 2)) {
        if (methodDef.arguments.size() == 1 && methodDef.name == "runIn") {
            addViolation(msg: "Avoid chained runIn() calls."
            methodDef.arguments[0].variable.name == methodDef.name
        }
```}

Fig. 3. CodeNarc rule implementation for AvoidChainedRunInCall

We selected 38 out of 348 default CodeNarc rules and added 22 new rules based on the code review guidelines into CodeNarc. For example, Fig. 3 shows the implementation of a new rule AvoidChainedRunInCall. Since CodeNarc uses static analysis, it relies heavily on Abstract Syntax Tree (AST) traversal to inspect the code structure and check violations without having to run the program [4].

Fig. 4. Sample CodeNarc report (excerpt)

Fig. 4 displays an excerpt of the HTML report generated by CodeNarc after analyzing multiple SmartApp source codes at once. The information generated from the report will be integrated
into an evaluation tool which will be developed.

5. Metrics
The rules are converted into metrics for measuring source code violations and other attributes. List 1 shows 20 out of 60 metrics both from SmartApp guidelines and CodeNarc default rules. The CodeNarc default rules include basic and convention rules for Groovy and also size and complexity rules. Fig. 5 shows the metrics categorized according to the SmartApp characteristics discussed in Section 3.

A total of 60 SmartApp metrics can be used to evaluate some quality attributes such as Reliability, Security, and Maintainability. Even though this study does not involve the proposal of a quality model, the metrics can still be used to evaluate measurable quality characteristics of SmartApps. For instance, the number of missing event handlers as shown in Fig. 5 can be used to measure reliability since the violation suggests the code being prone to faults if a certain event handler is called but was not defined at all. Another example is the hard-coded SMS message violation count under the category external system access. It suggests issues related to security since hard-coded phone numbers put the system to risk if the value is wrong and cannot be updated using the SmartApp preferences and settings. The guideline suggests a safe and proper way to implement it by using the contact input so it will be subjected to validation and can be updated. In this way, we are able to show that SmartApp quality characteristics can be evaluated through expressing the rules into metrics which are taken from the output of the automatic code review tool.

6. Conclusion and Future Work
This research proposes automatic code review using static analysis, which can be used to evaluate the characteristics of SmartApp source code. First, the code review guidelines for SmartApps are implemented into rules for CodeNarc. Next, metrics are defined to evaluate the characteristics unique to SmartApps. Finally, static analysis of source code is performed through CodeNarc where it generates an HTML report providing details of the violations, source of error and priority level. The information from the report can be used as the source or input to the evaluation tool to be developed.

However, static analysis has some limitations. It cannot fully automate the code review process since some rules need coordination with the developer, making them impossible to be implemented by checking the source code only. Although limitations exist, it makes the process of code review way more efficient than having to do it manually. In addition, the tool developed was able to provide information regarding which quality characteristics need to be improved in the system.

For the future work, a scoring system for each of the characteristics – reliability, security, and maintainability, must be developed in order to calculate and produce measurable results. Determining the score rating for each of the attributes allows quantifiable ways to measure the quality characteristics of SmartApps.

7. References