Description of Consistency Rules in Software Design Models

1 Sadia Sahar, 2 Tasleem Mustafa, 3 Farnaz Usman, 4 Aasma Khalid, 5 Nadia Aslam, 6 Sidra Hafeez
1, 2, 3, 4, 5, 6 Dept. of Computer Science, University of Agriculture Faisalabad, Pakistan

1 sadiasahar@ymail.com, 2 tasleemmustafa@hotmail.com, 3 rhusmaas123ro@yahoo.com, 4 aasmaakhalid89@yahoo.com.
5 nadiaslam7@yahoo.com, 6 pinkdreamflower@gmail.com

INTRODUCTION

Design is considered as the important phase of software development because of the demand of quality and complexity of software. One of the greatest challenges faced by software engineers and researchers is the identification and resolution of inconsistencies within a system’s design. If inconsistencies are detected in design phase then the problem that might occur at later stages of software development can be reduced. As inconsistency is the violation of the rules of software development, the rules must be clearly defined so that one can easily detect the inconsistencies in his software design and can propose a better solution to resolve them. In this paper emphasis is made on the types of inconsistencies and the consistency rules.

Keywords: Component; software design; consistency; inconsistency; UML

1. INTRODUCTION

Software system is gaining hold on our daily activities. May it be a grocery store, building designing and architecture, vehicle manufacturing, communication technology, online commerce, local administration, defense or any other aspects of society software system is enhancing the functioning of all aspects of our society. The software systems become critical to all of them. Day to day increasing demands for software system to ease working of nearby every field of life is giving growth to many functional requirements such marketing the stuff in minimum available time, friendly user interface, seamless operating environment and so on. This leads increased complexity on software development (Olsan and Grundy, 2001). To master this complexity software engineers build and employ well defined development process such as waterfall model and spiral model (Boehm, 1988). The developers tackle software problems in four main phases. These are typically identified as Requirement elicitation and analysis, Architectural and detailed design, Implementation and Testing (Ibrahim et al., 2011). But in development process a large number of different descriptions are used by software engineers. These descriptions include different analysis modes, design, specification, program codes, user guides, style guides, test plans, change requests, schedules and process models (Jackson, 1995).

Design is becoming the important phase of software development process due to demand of quality and complexity of software (Zaretska et al., 2012). The industries are trying to search out the techniques which automatically produce software and can also improve quality with reduction of cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. The increase in demands of software and its quality leads to the complexity of software. This enhances the scope of project and hence the time. Organizations also inquire about techniques to manage the complexity of systems. They also need to solve physical allocation, concurrency, duplication, safety problem, load balancing and fault tolerance. The Unified Modeling Language (UML) is the best design based solution to counter these needs (Larman, 2004).

UML is de facto standard for modeling languages used for software design (Zaretska et al., 2012). Most of the software designs consist of thousands of UML models so a greater chance of inconsistency between software designs is present. As Inconsistency is the violation of rules or relationships that must be obeyed by different description in software development process (Nuseibeh et al., 2001), the rules must be clearly defined so that one can easily detect the inconsistencies in his software design and can propose a better solution to resolve them. In this paper emphasis is made on the types of inconsistencies and the consistency rules.

2. CONSISTENCY RULES IN UML

Use case diagram is not used in design phase of software development (Jacobson, 1994) so it is not included it into research work.

2.1 Consistency Rules for Class Diagram

Rule 1: Concrete Class must have operation and attribute
Rule 2: Operations and attributes present in class must have data type (Liu et al., 2002).
Rule 3: Abstract class used in class diagram must have at-least one concrete class that must have implementation of it (Streaten and Brussel, 2011).
Rule 4: A concrete class cannot define an abstract method (Streaten and Brussel, 2011).
Rule 5: A method in a class cannot have more than one return attribute (Streaten and Brussel, 2011).
Rule 6: A cyclic inheritance of a class in not allowed (Streaten and Brussel, 2011).
Rule 7: Two classes in a class diagram cannot hold the same name (Pabitha et al., 2012).
Rule 8: Two or more classes cannot define an attribute and a function with the same name (Pabitha et al., 2012).
Rule 9: A subclass A of a super class B in one UML model is always behave as a subclass of B in other UML model.
Rule 10: A method in a class cannot have more than one return type (Streaten and Brussel, 2011).
Rule 11: Parameter of method in a class must have a type (Streaten and Brussel, 2011).

2.2 Consistency Rule for Object Diagram
Rule 12: An Object diagram has same classes as mentioned in class diagram
Rule 13: Attribute and operation used in object diagram must be defined in the relevant class.
Rule 14: Attribute value must match with the attribute type defined in class diagram
Rule 15: The number of parameter pass in operation in object diagram must be as in class diagram
Rule 16: The parameter pass in operation in object diagram must be of that type as declared in class diagram
Rule 17: Association between classes must be the same as in class diagram.

2.3 Consistency Rule for Activity Diagram
Rule 18: A use case has at-least one activity diagram (Ibrahim et al. 2011).
Rule 19: An actor who associates with use case will be declared as an activity partition in activity diagram (Ibrahim et al. 2011).
Rule 20: If a use case, (including use case) include a use case, (included use case), then in the associated activity diagram for the including use case there exist action node that refer to the associated activity diagram for the included use case (Ibrahim et al. 2011).
Rule 21: An initial node has no incoming edge (Ibrahim et al. 2011).

2.4 Consistency Rules for Sequence Diagram
Rule 22: Class uses in sequence diagram must be present in class diagram (Liu, 2002).
Rule 23: Operation used as message must be present in relevant class diagram (Pabitha et al., 2012).
Rule 24: Parameter of operation in sequence diagram must be matched with operation used in class diagram (Pabitha et al., 2012).
Rule 25: Recursion must have stop criteria.
Rule 26: Abstract class must not be used in sequence diagram (Streaten and Brussel, 2011).
Rule 27: Navigation direction between classes must be correct (Streaten and Brussel, 2011).
Rule 28: When a class X in sequence diagram sends the message to the second class Y then second class Y must be visible to the class X (Zaretska, 2012).

Rule 29: When a class X in sequence diagram sends the message to the second class Y then operation second class Y must be of static type (Zaretska, 2012).
Rule 30: When a class X in sequence diagram sends the message to the second class Y (without creating its object) then attribute of second class Y must be public (Zaretska, 2012).
Rule 31: When a class X in sequence diagram sends the message to the second class Y then operation of second class Y must be public (Zaretska, 2012).
Rule 32: When a class X in sequence diagram sends the message to the second class Y then second class Y must be visible to the class X (Zaretska, 2012).

2.5 Consistency Rules for Communication Diagram
Rule 33: Class uses in communication diagram must be present in class diagram.
Rule 34: Operation used as message must be present in relevant class diagram.
Rule 35: Parameter of operation in communication diagram must be matched with operation used in class diagram.
Rule 36: Recursion must have stop criteria.
Rule 37: Abstract class must not be used in Communication diagram.
Rule 38: Navigation direction between classes must be correct.
Rule 39: When a class X in communication diagram sends the message to the second class Y then second class Y must be visible to the class X.
Rule 40: When a class X in communication diagram sends the message to the second class Y then operation second class Y must be of static type.
Rule 41: When a class X in communication diagram sends the message to the second class Y then attribute of second class Y must be public.
Rule 42: When a class X in communication diagram sends the message to the second class Y operation second class Y must be public.
Rule 43: When a class X in communication diagram sends the message to the second class Y then second class Y must be visible to the class X.

2.6 Consistency Rules for State Diagram
Rule 44: A state not has too much transition that difficult to evaluate any guard condition first as a result may be erratic.
Rule 45: Only one guard condition can be true. No two or more conditions are allowed to be true.
Rule 46: Guard conditions must be visible.
Rule 47: Every state in the diagram has meaningful implementation no state can be an Invalid state (Nimiya, 2010).
Rule 48: A state must point out the right state.
Rule 49: Every state has at least one incoming or outgoing transition, no state can be unreachable state or impossible (Zaretska, 2012).
Rule 50: Sequence of state must be correct
Rule 51: Transition must have a condition that can be fulfilled
Rule 52: Action defined in state diagram must be matched with relevant class diagram.
Rule 53: No duplication of a state is acceptable
Rule 54: Condition that changes state position into in other state must be written as guard condition
Rule 55: A state of a class X, in class diagram, changes into other state by calling method of class X; the method must be present in class X (Zaretska, 2012).

2.7 Consistency Rules for Component Diagram
Rule 56: Every file used in component diagram must be present as a class in a class diagram.
Rule 57: Association between classes in component diagrams is the same as in class diagram.

2.8 Consistency Rules for Package Diagram
Rule 58: Each package name must be unique
Rule 59: Each package that contains version, that version must be a unique identifier
Rule 60: Each package must contain dictionary and narratives
Rule 61: Imported package must be visible to the importer
Rule 62: Importer cannot delete any data from imported package
Rule 63: Importer can add any data from imported package
Rule 64: Element marked as final cannot be extended
Rule 65: Cyclic inheritance is not allowed in extension relation
Rule 66: Cyclic inheritance is not allowed in usage relation

3. CLASSIFICATION OF INCONSISTENCIES
When the above mentioned rules are violated it will be an inconsistency. Inconsistency can be an intra inconsistency, within a single UML model, or an ultra inconsistency, between two or more UML models.
Inconsistencies can be classified as

A. Dualistic Inconsistency
B. Type Inconsistency
C. Navigational Inconsistency
D. Relational Inconsistency
E. Domain inconsistency
F. Missing object

The following table shows all types of consistencies with their examples and the rules that, if violated, create inconsistencies.

3.1 Dualistic Inconsistency
This type of inconsistency occurs when two or more objects or attribute or properties of the objects have same name.

Example 1:
The two classes in the class diagram having the same name is an example of dualistic inconsistency.

3.2 Type Inconsistency
It is an intra or ultra type inconsistency. It occurs when an object calls a function or attribute of any object and type mismatches or class has attribute and operation without any data type or one operation or attribute have more than one data type. Operations not define with its parameter type or they have no data type or no return type.

Example:
In the following example methods turnon() and turnoff() in class light controlling system has no type.

3.3 Missing Object
Object calls other objects or their elements that are not declared in their right places are called missing object inconsistency.

Example:
Object diagram having an attribute which is not present in relevant class diagram. In the following
diagram object everyday record contain attribute year that is not present in class diagram.

![Class Diagram]

**Fig 3.3:** Object diagram shows missing object

**Example 1:**
When an instance of class A in sequence diagram sends a message to the instance of class B which is not present in class B.

**Example 2:**
In a sequence diagram an instance of one class X sends message to the second class Y with missing parameter also shows inconsistency.

### 3.4 Navigational Inconsistency

**Example 1:** In a sequence diagram an instance of one class X sends message to the second class Y with wrong parameter.

**Example 2:** In a sequence diagram an instance of one class X sends message to the second class Y but navigational direction is wrong.

### 3.5 Domain Inconsistency

Every software problem has its own domain requirement that can be a consistency rule and when this rule is violated is called domain inconsistency.

**Example:**
Start date cannot be greater than the last date

### 3.6 Rapport or Relational Inconsistency

Inconsistencies created due to violation of relationship rules are called rapport inconsistency.

**Example:**
The two classes being used in the class diagrams having the same name A is an inconsistency.

### Table 1: Type of Inconsistencies and rules violates that inconsistency

<table>
<thead>
<tr>
<th>Types of Inconsistencies</th>
<th>Rule Violates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dualistic</td>
<td>15,18</td>
</tr>
<tr>
<td>Type Inconsistency</td>
<td>9,12, 36, 46</td>
</tr>
<tr>
<td>Navigational</td>
<td>39, 26, 32, 37, 43, 44, 57, 56, 61</td>
</tr>
<tr>
<td>Missing Object</td>
<td>24, 27, 28, 33, 35, 40,47, 51, 58, 62</td>
</tr>
<tr>
<td>Domain</td>
<td>None</td>
</tr>
<tr>
<td>Rapport</td>
<td>13,19,21, 22,23,30</td>
</tr>
</tbody>
</table>

### 4. CONCLUSION

Most of the research has been done on inconsistency management but there are no more than two or three rules defined between one or two diagrams of unified modeling language. This research highlights more than fifty consistency rules between eight diagrams of UML so one can easily detect the inconsistencies in his software design and can propose a better solution to resolve them.

### REFERENCES


