A Model Driven Framework for Automatic Detection and Tracking Inconsistencies

A. Ananda Rao¹, T. V. Rajini Kanth², G. Ramesh³*

¹ Director Academic and Planning, JNT University Anantapur, Ananthapuramu, Andhra Pradesh, India.
² CSE Department, Sreenidhi Institute of Science and Technology, Ghatkesar, Hyderabad, Telangana, India.
³ Research Scholar, CSE Department, JNT University Anantapur, Ananthapuramu, Andhra Pradesh, India.

* Corresponding author. Tel.: 09440862112; email: ramesh680@gmail.com
Manuscript submitted January 28, 2016; accepted March 28, 2016.
doi: 10.17706/jsw.11.6.538-553

Abstract: Software design model inconsistencies precipitate into flaws in system that can be avoided at the time of design of the system. Recent contributions in the software engineering domain confirmed this fact clearly. Obstructions in software development and delivery can lead to economic and time-to-market attributes of the software. The consequences of model inconsistencies will have ripple effect in three areas such as cost, development time and delivery. Though UML provides unified and common notations across the globe, the developers may use the models inappropriately leading to model inconsistencies. Many researches came into existence to handle model inconstancies. However, a comprehensive, flexible and extensible framework that caters to the needs of developers with diverse tool usage and skill set is desired. Towards this end, in our previous work, we proposed a framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) for checking software design inconsistencies. The framework was made flexible with placeholders to support different modeling tools, rule detectors and visualizations. In this paper we focus on providing more features for all placeholders to realize a truly flexible and extensible design inconsistency checker. This improved framework and the implemented prototype can help software engineers to build more consistent software design models that can avoid cost and budget overruns as the application can provide early detection of inconstancies. Moreover, the application can support certain degree of tolerance for inconsistencies and help software engineers to switch between UML tools, consistency checkers and visualization mechanisms. The empirical evaluation shows that our framework is flexible and reveals significant performance improvement over other state-of-the-art inconsistency checkers in terms of accuracy, speed and scalability.

Key words: UML tools, design inconsistencies, consistency rules, visualization.

1. Introduction

The utility of Computer Aided Software Engineering (CASE) has been around for last two decades and it is growing consistency. Towards this, many modelling tools such as Rational Rose [1] came into existence. This was due to the UML specifications provided by OMG as it helped many vendors to build modelling tools. However, the modelling tools do little about software design model inconsistencies. When inconsistencies are not identified in the design phase, they can reflect in other phases of SDLC. This will lead to defects or flaws in software and that can eventually affect cost and time and time-to-market parameters of the software. It has its consequences which are not desirable. As we explored in our previous paper [2] there
exists many tools that focused on Model Driven Engineering (MDE). They are Andro MDA, BoUML, BluAge, Enterprise Architect, and Md Workbench to mention few.

The existing tools used for checking model inconsistencies follow different approaches in terms using modelling tool, consistency rule language and visualization techniques. In the literature it was found that the tools were built with different approaches. An important insight in the literature is that there was little research on having a comprehensive tool that can support different modelling tools, consistency checking languages and visualization techniques. The rationale behind this kind of need is that software engineers across the globe do have different skill sets and experience in using certain modelling tools, visualization and consistency rules. A comprehensive tool can provide them choices for selecting modelling tool, consistency checking language and visualization mechanism prior to building software design models. This however is a challenging problem to be addressed.

In our previous work we proposed a comprehensive, flexible and extensible framework known as Extensible Real Time Software Design Inconsistency Checker (XRTSDIC). This framework has two broad parts. First one is personalized configuration while the second one is execution model. The personalized configuration helps users to choose any modelling tool, consistency rule language and visualization mechanism while the execution model takes care of model checking for inconsistencies in a live environment. To our knowledge, it is the first of its kind with such features. More details of the framework can be found in our previous papers [3], [4].

In this paper, our contributions are described here. We improved the framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) proposed by us in [3]. We provided different approaches for rule detection and visualization. These contributions made the framework and the corresponding application to allow software engineers to select modelling tool, consistency rule language and visualization based on their knowledge and choice. This kind of flexibility in the tool can help users to adapt to the tool with ease. The remainder of the paper is structured as follows. Section 2 reviews literature on prior works on consistency checking. Section 3 provides preliminaries. Section 4 presents the proposed framework. Section 5 provides experimental results while Section 6 concludes the paper besides providing directions for future work.

2. Related Works

This section provides prior works on consistency checking models. Some researchers compared design models for discovering inconsistencies while others transformed models and compared. In [5] a graph structure is used to find the difference between class and sequence diagrams. In [6], [7] also transformation approach is followed. Groher et al. [8] makes use of description logic in order to find discrepancies between state chart and sequence diagrams. Campbell et al. [9] explored a model named SPIN for consistency checking in UML. Knowledge base and discovery of patterns is another approach for expressing consistency rules [10]. Incremental consistency checking approach was found in [11]-[13] which is used when transformation is expensive. ArgoUML also supports incremental consistency checking provided annotated consistency rules [14]. Warm queue and hot queue are the two methods of consistency checking in ArgoUML. There were many consistency checking methods that directly use UML models for comparison without the need for transformation [15]-[17].

Consistency of documents is done using xLinkIt [17] which is an XML based solution for consistency checking. Another approach which is similar to that of xLinkIt is in [18] which rely on SQL queries. Research on tolerating inconsistencies is found in [6], [19] that allow inconsistencies in models as they are allowed intentionally. In [20] lazy consistency checking was explored which is something close to the tolerance of inconsistencies. However, the tolerated inconsistencies are to be resolved ultimately as explored in [21] -
Upstream modelling technique and viewpoints were explored in [21], [25]-[27] for consistency checking.

In [28] the researcher presented an automated approach for finding and tracking inconsistencies. It allows software engineers to define rules to detect inconsistencies. It automatically detects inconsistencies when model is changed. Reder and Egyed [29] presented a tool named Model/Anlyzer which is practically a plug-in for modelling software named Rational Software Modeler (RSM). Based on the context of model, the tool could provide feedback on the rules defined by software engineers. The tool is incremental in nature in detecting and providing feedback on model inconsistencies. Costa et al. [30] explored the detection of semantic conflicts with respect to UML class diagrams. Their tool can detect conflicts between two versions of class diagrams and help software engineers to resolve them. In [31] a tool was presented for instance checking of inconsistencies in UML models that are used as part of software engineering. The tool could keep track of errors if any.

Ebeid et al. [32] explored source code generation from sequence diagrams of UML/MARTE. To this effect they proposed a methodology that can help in systematic approach in generating code. Ahmed et al. [33] explored automatic checking of non-functional and functional requirements with an integrated approach. Their method focused on self-adaptive systems using a tool named OMEGA/IFx. Marco et al. [34] combined Open Service for Lifecycle Collaboration (OSLC) and Xtext in order to have an integrated model-based solution for requirements engineering. They have explored tool chaining in order to achieve best results in requirements engineering. Han et al. [35] explored model-based analysis of IEEE 802.11 systems that are space-aware. They used BeSpaceD tool in order to verify spatial constraints. They worked on autonomous robots in their experiments in wireless environment.

Lytra et al. [36] explored the notion of component models that helped in making well informed architectural decisions. Their research helped in easing the difficulties in making architectural decisions by harmonizing them. Wawrzik et al. [37] explored modelling and simulation of cyber-physical systems. They built a framework to achieve this follows certain methodology. Abdul Ganiyi [38] focused on UML class diagrams with vertical semantic consistency rules. Thus their research helped in refining class diagrams. Soltana [39] explored model-based solution for legal compliance checking with respect to legal policies. Sun et al. [40] explored a novel approach based on slicing for improving model checking. Especially they focused on model invariant checking as part of model driven development (MDD). They found that slicing could help reduce model checking time significantly. Buchmann and Karagiannis [41] explored modelling of mobile application requirements with the facility known as semantic traceability.

Sporer et al. [42] explored the gaps between software engineering tools and model-driven engineering tools and tried to bridge gap between them in order to improve consistency, completeness and correctness. Tran et al. [43] provided a graph based solution for checking model inconsistencies. They proposed a light weight approach for containment checking which is essential in UML modelling. Their focus on UML activity diagrams. Gargiulo et al. [44] focused on consistency verification of requirements and reviewed several solutions for the same. Lorber [45] used mutation testing for checking inconsistencies in real world systems. Their solution was model-based. Zurowska and Dingel [46] explored a tool named UML-RT provided by IBM. They followed a dedicated approach in verification of models using MDD approach that made use of algorithms. Swaminathan Jayaraman and Bharat Jayaraman [47] explored design-time specification in finding consistency of Java run-time. They could compare and map runtime behaviour with design time specifications.

In this paper we implemented a framework known as Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) for checking software design inconsistencies. The framework was made flexible with placeholders to support different modelling tools, rule detectors and visualizations. In this
3. Our Framework

This section provides the overview of our framework explored in [3]-[4] for automatic detection and tracking of inconsistencies in UML design models. The framework is flexible and extensible with placeholders that can help in adding new approaches for every possible functionality in future. The framework is named as eXtensible Real Time Software Design Inconsistency Checker (XRTSDIC). The framework allows software engineers to choose modelling tool, consistency rule language and visualization technique. Thus the framework can help software engineers with diverse skill set and preferences. The personalized preferences part of the framework can leverage flexibility as it can facilitate software engineers to have a choice in the aforesaid options. Once the preferences are registered with tool and associated with the user profile, the execution model comes into picture. Fig. 1 shows an overview of the framework.

![Fig. 1. Overview of proposed framework.](image)

As the software development is done by team of individuals who have diversified skills in designing, the framework helps them to choose different UML notations, different language for consistency rules and different visualization based on their choice. Thus the architecture is made flexible and extensible. Based on the framework shown in Fig. 1, we built a prototype application using Java programming language. The application can demonstrate the proof of concept and helps developers to have personalized configurations with respect to design models.

Personalized configuration is the feature that lets software engineers to choose different aspects of modelling as said earlier. These choices or preferences are personalized so as to associate choices with user and their models. Thus the tool is made very flexible and useful for software engineers. When an engineer draws UML diagrams using the selected modelling tool, instance checking of inconsistencies is possible. This is done automatically as user draws model diagrams. The execution model section below provides more insights into this. The preferences are made available to execution model so that it works accordingly. When a model is built or modified, the model dynamics tracker is responsible to record changes. Then the rule detector is responsible to know the rules that are affected by the model changes. Only those rules are identified and given to consistency checker. The consistency checker will check those rules to know whether model has violated them. Then the modelling tool will provide feedback to end user based on the visualization preference that was made by the user prior to drawing model.
4. Execution Model

Let the selected modelling tools be MT, consistency checker CC, rule detector RD, model dynamic tracker MDT, and visualize V. The design model is denoted as M. Change in the model is denoted as MC. The user preferences are denoted as UP. These are considered global variables available to all routines. The pseudo code provided here provides the dynamics of execution model. The execution model comes into picture once user selects her preferences.

4.1. Pseudo Code for the Flow of Execution Model

The pseudo code provides the details of the proposed execution model which takes care of runtime model change and detection of inconstancies. It makes use of three different algorithms to achieve this. Based on the user drawing of model, it follows an iterative approach to detect corresponding rules and identify inconsistencies. Once inconsistencies are identified, they are visualized based on the preferences chosen by end user of the application.

4.2. Rule Detector Algorithm

Rule detection plays a vital role in the inconsistency checking. Rules are used based on the user preferences. The rule selection language is part of the preferences based on the rules are taken from rules database. The rule detector is responsible to detect all the possible rules that are to be applied against the model change occurred when model is drawn or modified.

Algorithm 1 – Rule Detection

```
1 Initialize context vector C
2 Initialize rules vector R
3 Initialize model dynamics vector MD
4 Initialize inconsistencies vector INC
5 Do
6 IF MC is true THEN
7 Notify MDT
8 END IF
9 IF MDT has fresh notification THEN
10 MD = model dynamics
11 R = RuleDetector(MD)
12 END IF
13 IF R1=NULL THEN
14 INC = ConsistencyChecker(R, MD)
15 IF INC !NULL THEN
16 Update C with INC data and metadata
17 Visualizer(C)
18 END IF
19 While(drawning==true)
```

The pseudo code provides the details of the proposed execution model which takes care of runtime model change and detection of inconstancies. It makes use of three different algorithms to achieve this. Based on the user drawing of model, it follows an iterative approach to detect corresponding rules and identify inconsistencies. Once inconsistencies are identified, they are visualized based on the preferences chosen by end user of the application.

```sql
RuleDetector(ModelDynamics MD)
1 Initialize rules vector R
2 Initialize violated rules VR
3 Initialize choice to 0
4 choice = UP.ruleChoice
5 IF choice =1 THEN
6 R = RulesDB(1)
7 ELSE IF choice=2 THEN
8 R = RulesDB(2)
9 ELSE IF choice=3 THEN
10 R = RulesDB(3)
11 END IF
12 FOR each md in MD
13 FOR each r in R
14 IF md is related to r THEN
15 add r to VR
16 END IF
17 END FOR
18 END FOR
19 return VR
```
This algorithm takes model dynamics as input. The model dynamics refer to the changes that have been made to model. It applies the rules based on user preferences with respect to rule selection. Then it applies all rules to each model element that has been subjected to changes. In each iteration, it identified whether a rule is to be considered for verification. Ultimately it returns the set of rules that are to be verified against model violations if any.

4.3. Consistency Checker Algorithm

Algorithm 2 – Consistency Rule Checking

This algorithm takes rules considered for verification and the model dynamics as input. Each rule is verified against each model change and the inconsistency vector is build. Then the inconsistency vector is returned back to its caller.

4.4. Visualization Algorithm

Algorithm 3 – Visualization

This algorithm takes the context vector which contains the model violations and its related meta data containing application, module, and other details pertaining to the model element in which violation occurred. It visualizes the inconsistencies based on user preferences. It has different visualization capabilities such as textual visualization, graphical visualization, structural visualization and so on.

5. Consistency Rules

There are many notations to model software systems using UML. They include Use Case, Sequence, Collaboration, Class, Object, State Chart, Activity, and so on. UML modelling tools may not be able to show inconsistencies. Therefore it is important to have support for automatic consistency checking. Early detection of inconsistencies in software systems in design models can help reduce time and cost of software development. Consistency rules can help in achieving this as explored in [48]. Fig. 2 shows inconsistencies in design.
When the inconsistencies are not identified at design level, it goes to subsequent phases in software development causing unnecessary waste and time and money. The consequences may be severe when deadlines are not met. Consistency rules as shown in Table 1 can help in automatic checking of
inconsistencies in software design models.

Table 1. Consistency Rules

<table>
<thead>
<tr>
<th>RULE</th>
<th>DESCRIPTION</th>
<th>CONTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 01</td>
<td>An object in the sequence diagram should exist as a concrete class in class diagram.</td>
<td>Class vs. Sequence</td>
</tr>
<tr>
<td>Rule 02</td>
<td>When a class name is modified in class diagram, it should reflect in all instance of sequence diagram synchronously.</td>
<td>Class vs. Sequence</td>
</tr>
<tr>
<td>Rule 03</td>
<td>When an object sends message to another object, there must be dependency relationship between them and there must be at least one message between such classes.</td>
<td>Class vs. Sequence</td>
</tr>
<tr>
<td>Rule 04</td>
<td>In sequence diagram a message should have corresponding operation in the receiver and it should be visible to sender.</td>
<td>Class vs. Sequence</td>
</tr>
<tr>
<td>Rule 05</td>
<td>When an object is deleted from a class diagram, its instances should be removed automatically from sequence diagrams.</td>
<td>Class vs. Sequence</td>
</tr>
<tr>
<td>Rule 06</td>
<td>An object represented in sequence and collaboration diagrams should correspond to same class in class diagram.</td>
<td>Sequence vs. Collaboration</td>
</tr>
<tr>
<td>Rule 07</td>
<td>An object represented in state machine must be an instance of concrete class in class diagram.</td>
<td>Sequence vs. Collaboration</td>
</tr>
<tr>
<td>Rule 08</td>
<td>When a class is deleted from class diagram, corresponding state machine diagrams should be deleted automatically.</td>
<td>Sequence vs. Collaboration</td>
</tr>
<tr>
<td>Rule 09</td>
<td>A state represented in state machine diagram should be a legitimate value of an attribute of corresponding class in class diagram.</td>
<td>Sequence vs. Collaboration</td>
</tr>
<tr>
<td>Rule 10</td>
<td>The operation used in the state machine diagram should be consistent with the operation in the class diagram in all aspects.</td>
<td>Class vs. State Machine</td>
</tr>
<tr>
<td>Rule 11</td>
<td>An activity in state machine diagram must be a message represented in the sequence diagram.</td>
<td>Sequence vs. State Machine</td>
</tr>
<tr>
<td>Rule 12</td>
<td>Use cases represented in use case diagram should be reflected in the operations of class diagrams.</td>
<td>Use case vs. Class</td>
</tr>
<tr>
<td>Rule 13</td>
<td>Activities and swim lanes in an activity diagram must have corresponding operations in respective classes.</td>
<td>Activity vs. Class</td>
</tr>
</tbody>
</table>

Four methods exist for inconsistency checking. They are manual check, dynamic check, automatic maintenance and compulsory restriction. The relationship between the rules and the methods is presented in Table 2.

Table 2. Consistency Checking Methods and Rules

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Best Applicable Rules (as shown in Table1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual check</td>
<td>Software engineer checks inconsistencies manually</td>
<td>7, 9, 12, 13</td>
</tr>
<tr>
<td>Dynamic check</td>
<td>Real time checking of inconsistencies against changes</td>
<td>6, 10, 11</td>
</tr>
<tr>
<td>Automatic maintenance</td>
<td>Modelling tool makes required changes to user initiated ones</td>
<td>2, 4, 5, 8</td>
</tr>
</tbody>
</table>
The relationship dynamics provided in Table 2 provide suitability of methods for applying different rules. However, in practice it is possible to apply more than a rule for rule enforcement.

6. Case Study and Prototype Evaluation

As shown in Table 3, 10 UML models are taken for experiments. The prototype application we built is used to check consistency of the models. However, the results of the evaluation of one model named “University Portal” with administration use case. Every employee belongs to a department. The employee and department classes are used to check inconsistencies in UML design models. The inconsistencies are visualized with different vitalization approaches.

As shown in Fig. 3, the duplicate attribute rule is violated in the model. Therefore the execution model has utilized the proposed algorithms and finally the detected violation is presented with different visualization approaches such as textual, graphical and structural.

As shown in Fig. 4, the duplicate method rule is violated in the model. Therefore the execution model has utilized the proposed algorithms and finally the detected violation is presented with different visualization approaches such as textual, graphical and structural.

Fig. 4. Detection of duplicate method rule violation.
As shown in Fig. 5, the invalid method rule is violated in the model. Therefore the execution model has utilized the proposed algorithms and finally the detected violation is presented with different visualization approaches such as textual, graphical and structural.

![Fig. 5. Detection of invalid method rule violation in sequence diagram.](image)

### 7. Experimental Results

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Class Diagram</th>
<th>Sequence Diagram</th>
<th>State chart Diagram</th>
<th># Model Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>145</td>
</tr>
<tr>
<td>Video on Demand</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>46</td>
</tr>
<tr>
<td>Online Courses</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>185</td>
</tr>
<tr>
<td>Billing System</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>230</td>
</tr>
<tr>
<td>Hospital Management</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>540</td>
</tr>
<tr>
<td>Hotel Management</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>890</td>
</tr>
<tr>
<td>University Portal</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>1230</td>
</tr>
<tr>
<td>Defect Tracking System</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>450</td>
</tr>
<tr>
<td>Valuation Portal</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>1125</td>
</tr>
<tr>
<td>School Management</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>1500</td>
</tr>
</tbody>
</table>
We used our prototype application for experiments. As many as 10 models shown in Table 3 are used for testing our application and the underlying framework on which the application is built. User preferences and the personalization of them is the important feature of the application. Prior to the drawing models, user can choose his preferences pertaining to modelling tool, consistency rule language and visualization method. Then the preferences are saved and associated with user profile. Afterwards, user can draw models or modify them.

The 10 models are used to find the feasibility of the proposed solution as it needs to check the computational cost and resource utilization for optimizing the application performance. The models are used to evaluate them based on the consistency rules defined by the developer. The rule detector is very important here to know the corresponding rule based on the model change.

The model evaluation time decreases significantly due to the approach used. This approach makes use of model elements that have been modified. The entire model is not verified for inconsistencies. This will save time and thus the proposed model can reduce evaluation time. It is an incremental and hierarchical approach that makes it intelligent to effectively delegate consistency checking to components in the architecture. It also provides scalability and accuracy. As the approach is heuristic that can make well informed decisions besides getting rid of unnecessary verifications.
As seen in Fig. 7, it is evident that the percentage of model change has its influence on the evaluation time taken. The horizontal axis represents the percentage of changes while the vertical axis provides average model evaluation time in milliseconds. We made 10 experiments for each percentage change and the average model evaluation time is recorded. The results reveal that there is performance improvement when our approach is compared with an existing approach. The evaluation time is very less as shown in results and thus the system is scalable to large models as well. With respect to memory cost, the observations show that the cost is increased when model size increases in linear fashion.

As can be seen in Fig. 8, model size has its influence in the consumption of main memory. As the modern computers have plenty of RAM, the memory consumption is not a big issue. However, it is a good practice to have resource efficient consistency checking models for optimal performance. Since the approach is heuristic in nature, memory is obviously consumed. Though it affects scalability, its effects are less and the system remains scalable.

8. Conclusion and Future Work

In this paper we focused on the framework proposed by us in our previous work. It is known as Extensible Real Time Software Design Inconsistency Checker (XRTSDIC). It provides a comprehensive, flexible and extensible architecture that can adapt to new UML modeling tools, consistency rule languages, and visualization mechanisms. However, in the previous paper the implementation had minimal features. In this paper we improved the framework and the application to realize the intended features of it. Thus our consistency checker has got more flexibility and offers different choices in terms of modeling tool, consistency rules, and visualization to software engineers while modeling their systems. This will make software engineers to utilize their skills in most productive way. As there was little research found in such comprehensive framework, we believe that our framework can help in building models with high accuracy thus avoiding unnecessary wastage of time and money. XRTSDIC also helps developers to tolerate model inconsistencies to certain extent. This will help them to skip certain unimportant aspect intentionally and move on to complete the design. The ability to choose different modeling tools, consistency rule languages and visualization mechanisms besides providing significant performance in terms of speed, accuracy and scalability makes it very useful in software engineering domain. Our empirical results with the prototype application reveal this fact. In future, we intend to focus on model transformations and improving our framework to support Software Product Lines (SPLs) for checking model inconsistencies.

References


Ananda Rao Akepogu received the B.Tech degree in computer science and engineering from University of Hyderabad, Andhra Pradesh, India and the M.Tech degree in A.I & robotics from University of Hyderabad, Andhra Pradesh, India. He received his PhD degree from Indian Institute of Technology Madras, Chennai, India. He is a professor of Computer Science and Engineering Department and currently working as a director academic and planning, of JNTUA College of Engineering, Anantapur, Jawaharlal Nehru Technological University, Andhra Pradesh, India. Dr. Rao has published more than 100 publications in various national and international journals/conferences. He received the best research paper award for the paper titled: An approach to test case design for cost effective software testing, ” International Conference on Software Engineering, Hong Kong, 2009. He received the best paper award: “Design and analysis of novel similarity measure for clustering and classification of high dimensional text documents,” in Proc. 15th ACM-International Conference on Computer Systems and Technologies, pp. 1-8, 2014, Ruse, Bulgaria, Europe. He also received best educationist award, Bharat Vidya Shiromani Award, Rashtriya Vidya Gaurav Gold Medal Award, Best Computer Teacher Award and Best Teacher Award from the Andhra Pradesh chief minister for the year 2014. His main research interest includes software engineering and data mining.

T. V. Rajinikanth received his M. Tech degree in computer science and engineering from Osmania University Hyderabad, Andhra Pradesh, India and he received his PhD degree from Osmania University Hyderabad, Andhra Pradesh, India. He is a professor of Computer Science and Engineering Department, SNIST, Hyderabad, Andhra Pradesh, India. He has published more than 50 publications in various national and international journals/conferences. He has Organized and Program Chaired 2 International Conferences, 2 grants received from UGC, AICTE. He is an editorial board member for several International Journals. He received his best paper award: Design and analysis of novel similarity measure for clustering and classification of high dimensional text documents,” in Proc. 15th ACM-International Conference on Computer Systems and Technologies, pp. 1-8, 2014, Ruse, Bulgaria, Europe. His main research interest includes image processing, data mining, machine learning.
G. Ramesh received his B. Tech degree in information technology from RGM CET, Nandyal, Kurnool Dist. Andhra Pradesh, and he received his M. Tech degree in software engineering from JNTUA College of Engineering, Ananthapuramu, Andhra Pradesh, India. He is perusing his Ph. D at JNTUA, Anapatramu, Andhra Pradesh, India. His main research interest includes software engineering and big data. He has published a several papers in various international journals/conferences.