A Review on Software Requirements Validation and Consistency Management

Massila Kamalrudin¹ and Safiah Sidek²

Innovative Software System & Services Group
University Teknikal Malaysia Melaka,
76100 Durian Tunggal, Melaka, Malaysia
¹massila@utem.edu.my, ²safiahsidek@utem.edu.my

Abstract

Requirements need to be validated at the early phase of the software development to avoid errors such as inconsistency, incompleteness and incorrectness. Drawn from this argument, a requirements validation process needs to consider Consistency, Completeness and Correctness (“3 Cs”) for the production of a quality software specifications. This paper provides a review of requirements validation and consistency management based on the existing literature in order to identify the gaps in the existing knowledge on the process of software requirements specifications. This paper begins with a review of the definitions of the 3Cs, upon which the understanding of the 3Cs is derived. Next comprehensive review of related works on the identified consistency management techniques: traceability and analysis approaches are then presented. This is supported with a heat map representations of the related research on the types of contributions, techniques, specifications and semantics used in consistency management. Since semi-formal specifications were found as the most common representation of the requirements, the types of models used as semi-formal specifications to represent the requirements were also discussed. Overall, this paper identifies the various gaps existing within the process of validating and managing the consistency of requirements to avoid re-inventing the wheels in the diverse and comprehensive knowledge of requirements engineering.

Keywords: Software Requirement Validation, Requirements Engineering, Consistency Management

1. Introduction

Software requirement specifications elaborate the functional and non-functional requirements, design artifacts, business processes and other aspects of a software system. Software requirement specifications that are complete and accepted by developers and clients provide a shared understanding and agreement of what a software system should do and why. Since requirement documents form the basis of development processes and this agreement, they should be correct, complete, and unambiguous [1] and need to be validated with respect to Consistency, Completeness and Correctness (“3 Cs”) to detect errors such as inconsistency and incompleteness [2].

For ensuring the 3Cs to be fulfilled in the requirements, a requirements validation process needs to be conducted in the software development cycle. Requirement validation is a process executed throughout the system life cycle [3]. It ensures the correctness, completeness and consistency of a requirement [3]. The validation process also helps to determine that the end product is correct and complete as well as to guarantee that the system developed satisfies the stakeholders’ original requirements [3]. Late validation of requirements could cause requirement quality to suffer [4]. In order to make sure that the original requirements of stakeholders are met, the requirements captured by the requirement engineer/analyst need to be entirely precise and consistent from the early
stage of the RE process. The inconsistencies of requirements also have the potential of creating an adverse effect in comparison to other errors since it can disrupt the whole development process of RE. Specifically, in cases when the requirement needs of the clients cannot be met, any attempts to do rectify the inconsistencies may cause delay, increase the cost of the system development process, put the properties related to the quality of a system at risk and make the maintenance process of a system cumbersome [5, 6]. Therefore, inconsistencies need to be avoided [7] and the 3Cs’, namely Consistency, Completeness and Correctness need to be taken into consideration in the requirements validation process. In fact the importance of ensuring consistency, completeness and correctness in the requirements validation process has been documented in the existing literature. Table 1 shows the definition of consistency; completeness and correctness from different points of view.

Based on the definitions stated in Table 1, we sum up our understanding of consistency, completeness and correctness (“3Cs”) in requirements. Consistency happens when any of the requirements components are intended to be equivalent. The requirements components also should have the same naming and the sequence of the requirements need to be in the same order throughout the software requirements specification. In addition, consistency happens when the requirements captured by a requirements engineer are confirmed as satisfying the clients’ intended need. We assume all the requirements are complete when there are no missing key definitions or constraints for the software system. We also assume all the requirements are correct when the requirements are captured accurately, without any redundancies and they reflect the actual requirements and needs of the clients.

Table 1. The Definitions of Correctness, Completeness and Accuracy

<table>
<thead>
<tr>
<th>Type of Requirement Quality</th>
<th>Description</th>
</tr>
</thead>
</table>
| Correctness                | ● “Describes the correspondence of that specification with the real needs of the intended users in much the same way that correctness of a piece of software refers to the agreement of the software part with its specification.” [2]  
● “A program is considered correct if it behaves as expected on each element of its input domain” [8].  
● "An SRS is correct if and only if every requirement represents something required of the system to be built” [9] |
| Completeness               | ● “Implies that all customer’s needs will be met when the system is constructed.” [2]  
● "A requirement must have all relevant components” [10]  
● “A requirement’s document should include requirements that define all functions and the constraints intended by the system user” [11]  
● “It specifies required behaviour and output for all possible states under all possible constraints.” [12]  
● “Responses of the software to all realizable classes of input data in all realizable classes of situations is included” [13] |
| Consistency                | ● “No two or more requirements in a specification contradict each other and the case where words and terms have the same meaning throughout the requirement’s specifications (consistent use of terminology)” [2]  
● “Requirement uses terms in a manner consistent with their specified meanings.” [10]  
● “Requirement should be understood precisely in the same way by every person who reads it.” [10]  
● “Consistency is also referring to situations where there is no internal (logical) contradiction in a specification of a system.” [14]  
● “Consistent specification exists when there is a computational model for its implementation and the specification is valid when it satisfies the user requirements.” [15]  
● "An SRS is internally consistent if and only if no subset of individual requirements stated therein conflict” [13] |
2. Consistency/Inconsistency Management

As discussed in our previous work [81] only few requirements tools are available in the market to provide facilities for the validation process especially for verifying and checking the requirements qualities, such as the completeness and consistency [17]. Thus, for further understanding on the technique and method used in managing the consistency, we investigate rigorously the related research done by others in validating requirements quality, especially the consistency. Other researchers have devoted their studies to investigate how to manage the consistency of requirements. There are efforts to check the existence of inconsistencies in either the informal specifications, semi-formal specifications or formal specifications [19, 20]. There is also work on managing the consistency in the architecture model and other design models [21, 22, 23]. In addition, there are heavyweight or lightweight approaches used to check for the inconsistencies. Further, there is also work on repairing inconsistencies and tolerating their presence [7, 24, 25]. There are many techniques to check for inconsistencies and to maintain the consistency of requirements [26]. Techniques used to check for consistency or to handle the inconsistency based errors include traceability and analysis approaches. Analysis approaches can be categorised as either formal analysis or heuristic analysis. Different types of specifications are also used to represent the requirements before consistency checking is conducted. Semantics is sometimes applied to the requirements to assist the validation process. We will explore work relating to these issues in the subsequent sections of this paper.

2.1. Traceability

Traceability is defined as the “ability to describe and follow the life of an artifact which is developed during software lifecycle in both forward and backwards directions” [27]. Traceability is an important approach to manage requirements effectively [28] and a vital practice in an organisation [29]. Traceability must also cover all aspects in terms of scope and coverage, including system level scope and all of the four types of coverage, as defined by Bashir et al. [30]. First, is the traceability between an origin of a requirement inclusive of source, stakeholders and requirements. Next, is the traceability between the requirements and other requirements, such as the functional and non-functional requirements. Then, is the traceability between the requirements and other artifacts which provides a trace between different requirements forms, such as specifications, designs and test cases. Finally, is the traceability between other artifacts and other artifacts, such as the possible links and dependencies among artifacts.

Cysneiros and Zisman (2008) assert that traceability relations help in a number of activities in software development [31], for example the evolution of software systems, compliance verification of code, requirements validation, aspect identification and any design decisions. Traceability is often informally practised in tracing requirements to and from a software design [28]. Some traceability techniques are assisted by information retrieval (IR), a derived technique to support identifying traceability links. However, IR is unable to identify all links [31,32. Although traceability is important, it is sometimes not applied in practice as it is too difficult and costly [29].

Many approaches have been proposed to maintain consistency and check inconsistency. One of them is traceability. The traceability technique is divided into two categories; forward/backward trace and derived. Olsson and Grundy developed a Web-based tool to summarise artifact data and to support basic explicit linking of elements in different representational models [22]. The method uses traceability and manages fuzzy relationships between high-level software artifacts (requirements), uses case models and black box test plans. The aim of this tool is to assist the inconsistency management for all changes made to artifacts. However, automation is impossible and that is needed to create
a relationship. Further, “high level natural language often lacks well-defined formal abstraction for all software artifacts representation” [31].

Cysneiro and Zisman implemented the automatic generation of traceability relations among various types of models generated during the development of agent-oriented systems and identification of missing elements in the Prometheus model and JACK code specification [31]. Its purpose is to check completeness in order to ensure the consistency between model and code specification is maintained, especially in a huge and complex system which involves different stakeholders. Rule-based approaches and Prometheus methodology are used with an extended version of XQuery to represent rules in traceability. However, this is still considered as a preliminary work and enhanced verification is needed.

Another technique to reduce inconsistencies among product lines was developed by Jirapanthong and Zisman [32]. XtraQue supports the generation of traceability relations in different types of documents that are capable of representing different levels of the development lifecycle of a product line [32]. It can define the semantics between the artifacts being compared and can also be used to bridge various activities and stakeholders taking part in the product line engineering [32]. It generates nine traceability relations, such as satisfiable, ability, dependency, overlaps, evolutions, implements, refinements, containment, similar and different features based on OO documents created during development [32]. An extension of XQuery is used to represent the traceability rules and consider the semantics of documents, the traceability relation of various types of traceability with the product line domain and the grammatical roles of the words in textual parts of document, together with the synonyms and distance of words being compared [32]. A Rule-based approach is also applied to automatically generate the traceability relations among elements of documents that are created during the development of the product line system. Nevertheless, the “existing rules failed to identify between requirements and object-oriented specification, besides changes in the documents require the traceability to be re-executed” [32].

Goknil et al., [33] proposed an approach together with a tool for defining requirement relations using traceability. They cater for issues of consistency, change management and inference of requirements. First order logic is used to support the consistency checking of relations and to infer new relations. However, their approach only supports textual requirements and lacks consistency management between textual and other requirement artifacts, such as the use of case and activity diagrams. There is also no automation provided for modeling requirements. The visualised result of either inferred relations or inconsistencies needs to be interpreted manually by the requirements engineer, which can lead to errors [33].

There is also a “technique to recover traceability links between source code and free text documentation” [34] using information retrieval which applies to both the IR techniques, namely the probabilistic and vector space. This technique is applied to trace C++ and Java source classes to manual pages as well as the functional requirements. However, the effectiveness of this technique becomes less prominent when the number of familiar words between the source code component identifiers and the documentation item decreases [34].

2.2. Analysis Approach

There are two types of analysis identified for checking consistency/inconsistency: heuristic analysis and formal analysis: line before, and one after.

2.1.1. Heuristic Analysis: “The term heuristic means a method which, on the basis of experience or judgement, seems likely to yield a reasonable solution to a problem, but which cannot be guaranteed to produce the mathematically optimal solution”[35]. Heuristic analysis is used without the structure of a mathematical model for making
decisions [35] and it is believed that it could assist in specifying the essential process for achieving the goal state [36]. It can also be used in a particular situation to specify the process involved in detecting an exception and taking corrective action [36]. Often, an heuristic algorithm is applied as it helps to provide a close right answer or solution for a specific instance of a problem [37].

As described earlier, analysis approaches are divided into two categories; heuristic analysis and formal analysis. The analysis approach is used together with requirement specifications and semantics. Heuristic analysis is one of the common techniques used to check for consistency or inconsistency of requirements. For example, Koth et al., [38] developed a technique to check the inconsistency of XML documents from a semantic point of view using an incremental attribute evaluation approach. This technique introduces incremental facilities and evaluates the attributes associated to XML semantics by “adding an incremental strategy to XML semantic checker evaluator” [38]. It also uses the Propagate algorithm and checks the consistency of documents repeatedly until a consistent document is produced. The efficiency of the evaluator is improved by lessening the re-evaluation process and evaluating the affected area only in the XML and not the entire document [38].

In addition, Chitchyan et al. implemented an automation support for requirements annotation, which is the extension of the WMatrix natural language processing tool suite called RDL. “RDL is a tool enriched with the existing natural language requirements specification with semantic information derived from the semantics of the natural language itself” [39] and MRAT tools (Multidimensional Requirements Analysis Tool) by Waters [40] which is an Eclipse plug-in is used to facilitate the composition and analysis. MRAT is used to analyse the temporal relationship of RDL composition and it is believed that finding the temporal dependencies is useful to determine the points of sequencing conflicts and to avoid the conflicts and inconsistencies from happening.

Kroha et al., [41] investigated the use of semantic web technology to check the consistency of requirement specifications. They transform the static part of UML models that illustrate requirements into a problem ontology and attempt to discover inconsistencies by using ontological reasoning to uncover contradictions [41]. This work does not, however, check for behavioural consistency as it cannot represent dynamic aspects of UML specifications in the ontology.

Much research has been devoted to checking inconsistency and consistency using semi-formal specifications and heuristic algorithms. Egyed[21] implemented a UML-based transformation framework to check inconsistency and help in comparison. The author introduced an automated checking tool called VIEWINTEGRA which used consistent transformation to translate diagrams into interpretations and used the consistency comparison to compare those interpretations with those of other diagrams [21]. This technique can check inconsistencies without the help of third party or intermediate languages. The limitation of this tool exists when checking the consistency between an object diagram and stating the chart diagram or vice versa, as they cannot be transformed directly and need to be changed to a class diagram in order to obtain the consistency results [21].

Sabetzadeh et al., [42] proposed a tool-supported approach for checking the consistency of a distributed model and enabling the checking for the inter-model properties of a set of models. This is done by checking of properties that merge within the intra-models [42]. A set of generic expressions is also developed to characterise the recurrent patterns in a structural constraint of a conceptual model. This approach currently works in the homogeneous model only as the merger cannot be defined at a notational level and this leads to a challenge in implementing this approach in a heterogeneous model [42].

In addition, Groher et al., presented an incremental consistency checker which allows one to define and redefine constraints [43]. This approach allows engineers to define and
change the meta model and model any time “without manual annotation or restriction on
the constraint of the language used” [43]. This approach is implemented in a tool called
“Model analyzer” which highlights the inconsistency of models in red [43].

Sinha et al., introduced a modeling environment called “Archetest” which adapts a
unique bi-layer approach for precise modeling and automates the analysis [44]. It also
analyses consistency and completeness. This approach accepts vague use case
descriptions and helps to provide accuracy to them through a wizard-driven process [44].
However, more case studies are needed to test this work in order to prove its early results
[44].

Kim [45] implemented a technique to assist the verification of user requirements
expressed in natural language. This technique verifies the discrete event simulation model
using a DEVS formalism together with a prototype tool called VERIDEV [45]. The
verification consists of “consistency verification between user requirements specification
and class diagram, consistency verification between user requirement specification and
sequence diagram of UML and the consistency verification between sequence diagram
and DEVS diagram” [45]. This technique is hard to apply because of the difficulties faced
in expressing it in the DEVS graph [45]. As a result, future enhancement is needed to
automate the technique of integrity checking for the text base used in describing a model
[45].

Chanda et al., proposed a formalisation methodology for the three most common uses
of a UML diagram in capturing the static and dynamic aspects of an object oriented
system: use case diagram, activity diagram and class diagram in order to emphasise inter-
diagram consistency, syntactic correctness and traceability of requirement by using
several formal rules [46]. A regular expression featuring (e. g., +, *) is used to enhance
the simplicity and understanding [46] of the grammar.

Jurack et al., presented a criteria for checking the consistency of refined activity
diagrams which includes pre- and post- conditions [47]. The work used graph
transformation rule sequences to define the behaviour of the refined activity diagrams to
check consistency. This allows the analysis of a set of sequence to be conducted in a static
manner [47]. However, the graph transformation rule cannot be checked with the static
analysis and it needs stronger reduction mechanisms to allow consistency analysis for a
wider range of activity diagrams. For now, a restriction and assumption is applied to deal
with the problem [47].

Litvak proposed an algorithmic approach to check the consistency of UML sequence
diagrams and state diagrams [48]. An automation tool called BVUML is used to
implement the consistency check algorithm [48]. The proposed algorithm helps in
handling “complex state diagrams such as fork, join and concurrent composite states”
[48]. The algorithm uses a breadth first search over the state diagram and a hybrid
sequence state diagram is introduced to visualise the process with which the diagram state
is associated to the sequence diagram [48]. This tool is sufficient for checking the
consistency of a UML dynamic diagram, suitable for the standard UML and is
demonstrated to be fast error detector. It is easy to use and does not require first or second
order logic knowledge to generate or to understand the tool. In contrast, “BVUML do not
support purely syntactical states such as the stub states” [48].

Whittle and Schumann presented an algorithm that works with a prototype tool in Java
to automate the generation of a UML state-chart from a scenario in a form of sequence
diagram [49]. Semantics information is added to the sequence diagram to detect and
report inconsistencies [49]. The concept of hierarchy and structure in a form of class
diagrams is used to show the merging of multiple sequence diagrams in a single state-
chart [49]. However, the generated state-chart is just a skeleton and can be a substitute for
manual refinement and modification [49].

Likewise, Li et al., [50] have also conducted research into the consistency checking of
UML diagrams. The research proposed a technique for checking the consistency of a
UML requirement model which comprises the use of cases and conceptual class models with system constraint [50]. Together with this, the consistency of the requirements can be checked logically using semantics. However, this proposed technique only focuses on the aspect of formal model of requirement consistency. In order to validate the functional aspect of a requirement, a prototype generator tool is developed. It helps to automatically generate Java source code from the formal model of a requirement [50].

Zapata et al., detected consistency problems in UML diagrams by implementing a novel approach using XPath and Xquery together with a rule-based system [51]. The reason for using them is because of “their strange mix of suitability and standardization” [51] that they can achieve. The main focus is to assess the “consistency rules between UML class diagram and use case diagram”[51]. These diagrams are integrated with OCL to avoid the ambiguities and to guarantee the well-formed models in a formal way [51].

Alternatively, Satyajit et al., [14] suggested finding and specifying consistency conditions (CCs) for the domain in the initial abstract formal specification with the aim of recovering logical errors during the early phase of development. The RAISE Specification Language (RSL) [14] is used in writing the formal specification for this purpose. This tool combines the inspection of a specification and testing the executable specification of a prototype using test cases [14]. The intention is to validate the specification against requirements and to ensure the specified CCs are respected and maintained by the operation defined in the specification [14]. However, CCs are not used for checking the consistency of a requirement specification.

Blanc et al., proposed an approach to deal with inconsistency based on model construction operations, which uses logical constraints to define inconsistency rules and it is also a meta model independent, which allows both intra-model and inter-model inconsistency rules to be defined and checked [52]. The consistency check is performed in a batch mode where the whole model is loaded into the memory and the verification starts by running the rules successively on the entire model [52].

Engels et al., presented a technique to specify and analyse consistency [53]. In order to conduct the checking process, models are mapped to semantic domains and behavioural constraints are analysed [53]. The problem of state-chart inheritance is demonstrated for this methodology [53]. A hybrid, rule-based notation is used. This rule combines textual styles of an attribute grammar with the queries of the meta model expressed as visual patterns [53]. The limitation faced by this technique is that it only supports partial resolution, although complete mapping is supported. It also needs a tool support to generate a model compiler for the rule-based description provided [53].

Ha and Kang proposed several verification rules to check for consistency between UML static and dynamic diagrams, such as class diagram, component diagram, state-chart diagram, sequence diagram, activity diagram, use case diagram, deployment diagram, collaboration diagram and object diagram [54]. A relation graph is used to show the relationship between diagrams. Consistency rules are developed from the relationships of both the object and dynamic diagrams [54]. However, these rules need help from the OCL (Object Constraint Language) as the rules need to first be transformed to formal language if the consistency checking is to be conducted automatically [54].

Ryndina et al., proposed a technique to establish the consistency between the business process model and object life cycles [55]. They defined two consistency notions for a process model called “life cycle compliance and coverage” [55] expressed in terms of conditions. A prototype tool that acts as an extension to the IBM WebSphere Business modeller was developed to help capture the existence of object states in the business process model, generating the life cycle from the process model and checking the consistency of consistency conditions [55]. This technique still has to be evaluated using a larger case study [55].

El-Attar and Miller proposed a structure presented in the use of case models called a Simple Structure Use Case Description (SSUCD) with a tool support called SAREUCD.
which helps automate the detection and elimination of possible defects caused by inconsistencies [56]. The authors invented a technique called Reverse Engineering of Use Case Diagrams (REUCD) to generate use case diagrams from the SSUCD [106]. The SSUCD and REUCD processes allow the use diagram to be generated systematically and helps to guarantee the “consistency between the descriptions and their diagrams” [56]. However, the tool support still requires human intervention to fill in the details in each use of case description before the tool can detect the inconsistencies. It also requires manual inspection for inconsistency if the segments are written in an unstructured natural language [56].

Another approach is presented by Perrouin et al. for managing the inconsistencies amongst heterogeneous models by using a model composition mechanism [57]. The information of the heterogeneous models is translated to a set of model fragments [57]. Fusion is applied to build a global model which allows various inconsistencies to be detected, resulting in the global model [57]. Automation is applied to compute traceability links between the input model and the global one and thus supports the reporting of the inconsistencies on the original model and helps to resolve the cause of the inconsistencies [57]. However, the classification of which inconsistencies need to be resolved is not provided [57].

Mehner et al., proposed an approach for analysing the interaction and the possible inconsistencies that might exist in the requirement modeling phase [58]. A variant of UML with the use of case-driven approach using use case diagrams, activity diagrams and class diagrams is applied [58]. The concept of pre- and post-condition using the UML variant of an activity is defined [58]. This requires more effort and it is recommended that there is an early formal analysis to overcome this problem [58]. The approach uses a formal technique called graph transformation with a tool support, AGG, in order to provide the chosen UML variant with formal semantics and allow a thorough and automatic analysis to be conducted [58]. This approach also allows the analysis of the interactions between the functional and non-functional aspects to be conducted automatically [58].

El-Mahded and Maibaum developed a tool called GOPSD to develop aspect-based process control and check for the consistency and completeness of a requirement [59]. The tool adapts the concept of goal-driven analysis which was adapted from the KAOS technique for addressing process control systems [59]. The tool offers an animation utility which helps to reason about the taken actions in terms of “aspect goals, cycle by cycle and during the symbolic execution” [59]. Further evaluation is needed for these purposes. GOPSD also covers the early stage of development and it “refines the abstract user’s needs to functional and formal specification” [59]. The tool also transforms the requirements automatically to B specification but the requirement needs to be corrected and validated first by the user before the transformation can be conducted [59].

In addition, Grundy et al., introduced a methodology of aspect-oriented component engineering to overcome problems related to component requirement engineering [60]. Their methodology analyses and characterises the component based on the “different aspects of the overall application component addresses” [60]. The authors developed tool support which helps to specify aspects of a component in a component based software development environment. The tool is equipped with basic validation checking in order to make sure all aspects of a requirement are met correctly [60]. This tool also provides a basic inconsistency management technique to help to manage the evolving aspect-oriented requirements including a highlighting of the change facility for all types of views and consistency checking via the matching of required links between components [60].

Another work related to checking the consistency using an aspect-oriented paradigm, this time for web applications, is by Yu [61]. The author presents a tool called HILA which was designed as an extension of UML state machines to model the adaptation rules for web application [61]. However, this work is believed not to be limited to web
engineering applications only but may also be applicable to various other areas [61]. HILA could be helpful in improving the modularity of models and helps to automate the consistency checking of aspects to ensure rules are always in a consistent state [61]. However, HILA is likely to be useful to model the content and presentation only if it is modeled in a base machine [61].

Kamalrudin et al., [62], develop a lightweight approach using Essential Use Cases (EUCs) model and traceability to check for the consistency of the requirements. Here, an EUC library pattern and a visual differencing technique are employed to automatically visualize the inconsistency, incompleteness and incorrectness. For this, a tool called MaramaAI is also developed. However, the tool is still in need of improvements especially on the EUC library pattern for supporting wider domain of applications.

To sum up, there are many approaches and techniques used to check or manage consistency and inconsistency of requirements by using heuristic analysis. Some of the techniques [48, 21, 41, 42, 44, 47, 49, 55] are well generated but most are still immature and need further enhancements. We have identified that most work needs tool support for the checking process. However, it is also true that most work integrate with other available tools and are not purely built for consistency checking, especially when this needs to deal with processing natural language. Most tools or approaches do not support rigorous checking for consistency, but only support partial solution for checking or identifying inconsistency and with a homogeneous model of a set of requirements. It is also identified that the tools developed still need human intervention to interpret the consistency results or invoke the action to check for the inconsistency although impressive and high level form of techniques are applied.

2.2.2. Formal Analysis: “Formal analysis helps to detect many types of errors in a requirements specification either manually or automatically” [63]. Formal analysis uses formal notation which can be used to analyse and manipulate mathematical operators and mathematical proof procedures [63]. It also provides benefits in testing and proving the internal consistency including data conservation and syntactic correctness of the specification”[64].

There are many researchers dedicated to checking the consistency of requirements using formal analysis together with the use of different types of requirement specifications and semantics. For example, Nenwitch et al. presented a lightweight framework called Xlinkit in order to check the consistency of distributed and heterogeneous documents using first order logic and lightweight mechanisms [65]. The main contribution of this framework is the definition of an extended semantics based on first order logic and producing hyperlink which diagnose inconsistencies across the specifications at different stages. The incremental checking technique used can also decrease checking time. However, XLinkit’s limitation is that it lacks discovery of problems if the inconsistencies are recognised.

Other work by Nentwich et al. proposes a repair framework for inconsistent distributed documents [24]. They generate interactive repairs from an input of a first order logic formula that constrains the documents. Their repair system provides a correct repair action for each inconsistency together with available choices to handle the problem. However, they face problems when the repair actions interact with the grammar in a document, and also actions generated by other constraints [24]. Their approach also fails to identify a single inconsistency that may lead to other inconsistencies [24].

Other than that, Chen and Ghose developed an automated tool using the semantic web technology called SC-CHECK. This tool mainly focuses on the consistency management in distributed requirements engineering, especially in detecting inconsistency and “supporting resolution in the context of industry-standard requirements specification notations” [66]. Prolog is used to identify the violated consistency rule and possible errors or elements that need to be repaired. It uses an informal requirement specification and
semi-formal representation in a form of sequence diagram for abstracting the formal representation to detect and resolve the inconsistencies [66]. It also provides the user with guidance to attempt to correct the inconsistencies. This tool has been tested via a medium scale case study and the results seem beneficial.

Zowghi et al., [18] proposed a technique to detect inconsistencies in requirements and the way to deal with them in a formal manner. A prototype tool named CARL is used to test the technique and it can perform an exhaustive search for plausible scenarios which cause latent inconsistencies to emerge [18]. A reasoning engine called CARET is applied to natural language requirements in order to analyse the inconsistencies within the translated logical statements of requirements. A simple engine for natural language parsing called Ciccois used as a generic syntax-based parser by taking a subset of the English grammar as its domain [18]. It uses an application of the fuzzy rewriting system to a text and uses heuristic optimisation strategies and backtracking too. It is also believed that it could help in identifying and handling inconsistencies in Natural language requirements. Although it is useful in identifying, analysing and handling inconsistencies, a more expressive logic is needed to hold the extended logic, the NL translation needs to be refined. Then, Gervasi and Zowghi [67] used the tool in detecting, analysing and handling inconsistencies in requirements for various stakeholders. It extends the tool with employing the theorem-proving and model-checking techniques in the context of default logic and it shows how to deal with the problems in a formal manner. The limitation of this tool is that the propositional logic used is not powerful enough to model adequate detail and accurate complex system behaviour. Propositional logic is not meant to detail the way the system should behave, but it is only suitable for high-level requirements [67].

Taibi et al., [68] implemented an algorithm for self-checking consistency for the classes using Object-Z specification. Verification utilises a test of specification, model abstraction and model checking. This algorithm conducts self-consistency only for each class and does not ensure consistency for the whole specification [68].

Kaneiwa and Satoh introduced an approach for conducting well-mannered consistency checking of UML class diagrams by translating the identified inconsistencies to first-order predicate logic [69]. They introduced an optimised algorithm with respect to the size of the class diagram to calculate “the respective consistencies of class diagrams of different expressive powers in P, NP, PSPACE, or EXPTIME” [69]. This work also helps to confirm the restrictions’ existence for the class diagram in order to avoid any logical inconsistency.

Lamsweerde et al., [70] proposed a framework which is based on both formal and heuristic techniques to discover the conflicts and divergences of the goals or requirements of a domain property from a specification. The notion of boundary condition and domain knowledge plays an important role in this technique and the KAOS language is chosen as the specification language. Here, model checking is used to detect divergence among goal assertion 70. It helps to capture the existence of different types of concept during the elaboration of requirements [70].

Kozlenkov and Zisman manage the consistency between natural language requirements and software artifacts that are generated during different phases of software system development, using a specific tool which embodies a goal driven and formal reasoning approach inclusive of “goal elaboration, ordered abduction and morphing of path” [20]. This is applied together with the use of knowledge-based and rule-based approaches. The weaknesses of this tool are that the inconsistencies discovered are limited to those related to the structure that has been recognised grammatically in natural language sentences only, and that the type of structures used needs to be expanded in order to allow the approach to be used in a large scale application [20].

Mu et al., [71] presented a merging-based approach to handling inconsistency by prioritising software requirements locally using the Viewpoints framework, which
The authors choose to use categorisation of the priority - High, Medium and Low. The first order logic is the best suited for this approach. Priority of requirements is measured to be a beneficial clue in resolving conflicts and making trade-off decisions. Conversely, there are problems occur while presenting the merging process. In a few cases, the user may not obtain a stratified merge requirements collection and the introduced model-based merging operators could lead to difficulty in explaining the additional formulas in terms of the viewpoint demands of the merge result [71].

Weitl et al., implemented an approach based on user support with the combination of pattern-based specification, temporal logic and ontology [72]. A Description Logic (DL) specification is used to represent the ontology and the content of the documents [72]. The verification framework is knowledge-based and the technique to support user specification is based on example- and a pattern-based approaches which are themselves based on the concrete examples of both correct and incorrect documents. This approach aims to check the consistency of high level structure with the content document and to check semantic consistency criteria on the context-dependent documents with high expressiveness, flexibility, applicability and high degree usability [72]. However, a broad knowledge of the logic used is needed in order to interpret or create the diagram and to apply the temporal formalism for solving the verification problems [72].

Scheffczyk et al., propose “formalizing the temporal consistency rules and generating a few domain specific repairs for inconsistencies” [73] of an industrial specification, using specification examples which focus on the functional requirements of a specification, such as business processes, use cases and dialogues [73]. Therefore, a semi-formal consistency management toolkit called CDET is used to improve the quality of the industrial requirement specification. CDET can be used as a tool to “check the semantics at different granularity levels and integrates fully with established practices” [73] and daily project work. CDET is also integrated smoothly with the arbitrary revision control system (RCS) with the aim to establish a work process. CDET uses derivation of temporal predicate logic to facilitate consistency checking across the document revisions and it is suitable for checking any property of a document that is computable [73]. Although this tool is profitable for heterogeneous documents, experts in the field of logic are required to formalise the consistency rules [73].

Sousa et al., [74] presented an approach of using formal specification to check for inconsistency in a requirement. They used the B specification as a formal language derived from a controlled natural language [74] in the form of use case descriptions or scenarios together with the B method - a well-known formal method based on “first order logic, a set of theory, integer arithmetic and generalized substitutions” [74]. The work automates the analysis of requirement consistency against constraints (safety property) with the B method tool to reveal the inconsistencies in the specification. However, the work still lacks supports in terms of quality dimensions, such as correctness and timeliness, lack of automatic consistency recovery such as a suggestion for changes and lack of support for a complex scenario and definition of grammar rules for use case scenarios and properties [74].

3. Analysis of Consistency / Inconsistency Management Research

From the discussion of the related works, we present a heat map in Figure 1 to show a categorization of the corpus work based on the type of specifications used, the type of contributions from all the researchers, type of semantics applied to each work and the type of techniques used to manage the consistency of requirements. The categorisation is mapped using colours (multiple tones of dark orange to light yellow) whereby mapping towards dark orange includes a higher percentage of papers following into this category.
To sum up, there are three types of specifications used to represent the requirements in a form of formal, semi-formal or informal specification. The specification most often used is semi-formal. The types of semi-formal specification used are UML models, structured diagram, scenario/textual description, description logics and other components. Most of the semi-formal specifications used are UML models. A UML model is described as the design representation of the source code and this diagram is useful in making the source code understandable [75]. Semi-formal specifications also receive great interest here because the models are easily decomposed into smaller parts and this allows them to be better understood [76]. This feature encourages much works done to check the consistency between models although some studies concluded that maintaining consistency between models is not important but expensive [76].

Another specification used to represent requirements in performing consistency checking is informal specification written in natural language. The least used representation is formal specification. This is because the use of formal specification is challenging and hard for the beginner to use for fast results [77].

There are also five types of contributions for conducting consistency checking work; tool, methodology, algorithm, framework and rules. Tools seem to receive the most interest from researchers to perform the checking for inconsistency. This is followed by the development of methodology and algorithm. Quantitative evidence proves Yu’s 17) point of view, discussed in the previous section. It was found that
there was a very limited research that focuses on developing frameworks and rules to handle the consistency. Most of the research to date has not applied any semantics. However, there are some work which apply the semantics of an artifact, followed by the semantics of natural language as well as the semantics of the web and of XML.

Techniques are categorised into two types; traceability and analysis approaches. The analysis approach is used more than the traceability because the latter has been identified by several researchers as being complicated and costly to use and it is also seen to have no proper method to conduct [29, 28]. Further, the current automated approaches of traceability do not allow engineers to have proper means to visualise the result [78, 79]. Both techniques are then divided into several sub-techniques. Traceability is divided into forward- and derived techniques. Here, the forward technique is used more than the derived to present the trace. The analysis approach is divided into formal analysis and heuristic analysis techniques. Figure 2 shows that heuristic analysis is applied more by researchers than the formal analysis when performing consistency checking of requirements. For the formal analysis technique, first-order logic is chosen most by researchers, followed by the model checking, temporal logic, formal reasoning, fuzzy logic, prolog and OCL. Theorem proving is least used in this work. For heuristic analysis, the use of constraints to perform the consistency checking has received enormous interest by researchers; followed by a descending popularity on the use of graph transformation, reverse engineering, relationship, rule-based, goal driven, model merging, condition and hybrid rule. The least used techniques are algorithm, early aspects, model composition, ontology, breadth first search, transformation, regular expression and Bi-layer.

![Figure 2. Heat Map representation: Classification of the Model Used as a Semi-Formal Specification Approaches](image-url)

As shown in Figure 2, most researchers use semi-formal specifications to represent the requirements to check for inconsistency. To investigate in more detail on the types of model used, we further classified the types of model using a heat map similar to the approach in Figure 1. As shown in Figure 2, from the 43 works discussed, more than half used the semi-formal specification. Based on the classification of model used in a semi-formal specification approach in Figure 2, the most used model in consistency checking work is the UML model. The case diagram and class diagram are used in most of the works, followed by the sequence
diagram, state chart diagram, activity diagram and object diagram. Models other than the UML, such as Scenario or textual requirement description, are also used in checking consistency. Models, such as task model, Essential Use Cases and Conventional Use Cases showed less and almost no interest by the researchers for checking the consistency of requirements, although Biddle et al., [80] found that Essential Use Cases open fruitful research of consistency issues between the responsibility concept in the requirements and their related designs as they help to improve the traceability support [80].

4. Conclusion

In this paper, we discussed the idea of 3 C’s: Consistency, correctness and completeness. Then we provide a discussion of requirement validation in general, highlighting on the consistency or inconsistency management. The different types of techniques used in consistency management or inconsistency were also discussed.

We conducted a literature review of related works based on the type of techniques used to manage consistency. In this section, we simplified and categorised the types of contributions, specifications, semantics and techniques used in the field of consistency management of requirements. We also compared the existing works and approaches and identified their strengths and weaknesses. A heat map was used to present graphically the data of the types of contributions, specifications, semantics and techniques used in the consistency management simplified from the related works. Here, the frequency of the usage was categorised based on different shades of orange colour. The higher the value of usage, the darker the colour of the squares. A similar approach was applied to represent the type of models used as a semi-formal specification to represent the requirements.

The techniques used were categorised into traceability and analysis approaches. Each technique was then divided into smaller categories. Traceability was divided into forward and derived techniques whilst analysis approach was divided into formal and heuristic analysis techniques. Each formal and heuristic analysis has its own sub-categories as described in previous sections. Most research used a combination of techniques. The semi-formal specifications technique was widely used by most researchers. We then analysed the model used for the works, applying the semi-formal specification technique through the heat map representation in Figure 2. In addition, from the related works as well as the analysis, we found that traceability techniques are less often used for consistency checking work due to the difficulties mentioned. The use of UML diagrams for checking the consistency have gained more interest among researchers. Other models such as Essential Use Cases were not thoroughly explored, although they were recognised as beneficial in checking the consistency of requirements and designs and have the ability to improve the traceability support. Further, most of the research did not have full coverage of the consistency checking of the requirements but tend to employ partial consistency checking, which focuses either only on the consistency of the natural language requirement or the models, or consistency between the natural language requirement and the models. Visual capability, such as highlighting the inconsistency to detect the inconsistency was used the least. There was a dearth of research that provides a complete end-to-end consistency checking support that is, from the natural language requirement to models and then to the prototype. Most of the researches conducted were mainly for the understanding and responsibility of requirement engineers and there were very limited research that provides confirming consistency and validating requirements from the clients’ side.

Acknowledgements

We acknowledge Professor John Hosking and John Grundy for their help and supervision. We also would like to thank the financial support from Ministry of Higher
Education Malaysia and Universiti Teknikal Malaysia Melaka for their assistance in this research. All authors acknowledge the support of the FRGS grant:FRGS/2/2013/ICT01/FTMK/0202/F00185.

References


Authors

Dr Massila Kamalrudin, born in Melaka, Malaysia. She is a Senior Lecturer of Software Engineering department and a research group leader of Innovative Software system & Services group of UTeM. Her first Degree was Computer Science majoring in Software Engineering from Universiti Putra Malaysia in 2003, then completed her MSc in Computing and Software Technology in 2006 from University of Wales, Swansea. In 2011 she gained her PhD from University of Auckland, New Zealand in the area of Requirements Engineering within 3 years and did a post-doctorate study at Swinburne University of Technology, Melbourne Australia.

She publishes in numbers of ISI/Scopus journals, IEEE/ACM conference proceedings and book chapters especially in the area of automated software engineering and requirements engineering. Her main interests of research are requirements engineering, security requirements, global software development and software tool and methodology.

Dr Kamalrudin is also a member of IEEE and Malaysia Software Engineering Group (MYSEIG).

Safiah Sidek, is a senior lecturer at the Centre for Languages and Human Development, Universiti Teknikal Malaysia Melaka. She currently holds the position of Deputy Dean of Academic at the center. She has a degree in Business Economics from Brock University, Canada and Masters in Science (TESL) from Universiti Putra Malaysia (UPM). She completed her PhD at Deakin University, Australia focusing on the use of online technology and changes in academics’ identities.

Her research interests include issues in globalisation, innovation and change, the use of online technology and research products commercialisation. She has been actively involved in research related to the use of technology. Besides being a research member for several research projects, she has led several research projects at the university level as well as the national level. At present, she is an active and key member of a research group, named Innovative
Software System and Services (IS3). Her significant contribution in this research group is drawn from her expertise in the social aspect of the use of technology. She has been actively involved in several ICT product innovations by investigating users’ acceptance and user’s behaviour on the newly developed technology, particularly related to the Internet of Things. Among the research projects that she is currently involved are the investigation of a sustainable usage of health information systems in Malaysian hospitals and the investigation of users’ behaviour on the use of new application in smart phones.

Safiah is also a prolific writer and has written and co-authored several articles in journals indexed ISI/Scopus and proceedings as well as book chapters. She has co-authored two books: a book on the identities of distinguished professors in Malaysia and another book on the real experience of Malaysian researchers in commercialising their research product.