Formal Methods in Requirements Phase of SDLC

S. K. Pandey
Department of Information Technology, Board of Studies,
The Institute of Chartered Accountants of India
(Set up by an Act of Parliament)
NOIDA - 201309, INDIA

Mona Batra
Assistant Professor, Dept. of CSE
International Institute of Management, Engineering & Technology
JAIPUR-302020, INDIA

ABSTRACT
Currently, there is an increasing demand for more rigorous and systematic approaches to develop security critical software systems across the globe. The complexity of the software system is rapidly raising due to the inclusion of properties like security and reliability. The process of software development complicates with the raising complexity of the software system. As a result, formal methods are currently used to model complex security critical systems. Literature reveals that formal methods can be applied at various points through the development process. Their tools can provide automated support, needed for checking completeness, traceability, verifiability, reusability and inconsistency management of requirement specification, which is the backbone of entire SDLC. Accordingly, there appears a need for a critical review of these formal methods. The paper presents a brief discussion on various formal methods particularly Z-method, B-method, VDM, OBJ, Larch and Communicating Sequential Process etc. along with their strengths and weaknesses followed by a comparative study on the basis of the review results. The present research work may help the software developers to provide their recommendations for using formal methods at different stages of software development and particularly for requirements phase, based on the specific requirements of an organization.

Keywords
Formal Methods, Comparative Study of Formal Methods, Requirements Engineering, Z-method, B-method, VDM, OBJ, Larch, Communicating Sequential Process etc.

1. INTRODUCTION
In today's digital era, businesses are facing a challenge of releasing commercial software projects of quality on time and within budget. Most of the software is delivered with some errors, with lack of complete functionality and sometimes with cost overrun. Overrun of budgets occurs due to errors in requirement specifications that can cost up to a hundred times to correct, when detected later in the development life cycle.

Formal methods are one of the feasible solutions to the above stated problems of current IT industries. Formal methods reduce the number of errors in the delivered product and are a cost-effective way of developing high integrity software [1]. These are a particular kind of mathematically based techniques for the specification, implementation, development and testing of software and hardware systems. They benefits in accuracy and testability of the software. These methods have a great potential in specification, early debugging of errors and certification of software [2].

Formal methods can be applied at various phases of development process such as at requirement specification phase (eliciting, articulating and representing requirements), software design phase, implementation phase (code verification), testing and software maintenance phase [3]. They are even becoming integral components of standards.

The representation used in formal methods is called a formal specification language. The formal specification languages are based on set theory and first order predicate calculus, but this mathematical background was initially not fully formalized [4]. Formal methods can detect or prevent the defect densities in specification, designs and code early on and hence can decrease the cost of correction and the development time [5]. In past, formal methods are generally used in real–time, safety critical systems but now these methods are finding their ways in other areas of various applications.

This paper describes different kinds of formal methods in requirements engineering, as found in literature. Beyond this introduction, the organization of paper is as follows: Section 2 presents a brief discussion on the existing formal methods, whereas in Section 3, detailed study of formal methods is done. Section 4 presents 'Strengths and Weaknesses' of each one and in Section 5, a 'Comparative Study' is done on the basis of the critical review. Section 6 presents 'Future Research Directions' in the area. 'Conclusion and Future Work' is reported in Section 7.

2. INTRODUCTION TO FORMAL METHODS
Formal methods are mathematics based languages, techniques and tools that can be applied at any part of the program life-cycle. These mathematics based methods also make use of refinement techniques at any stage to ensure the correctness, completeness and consistency of specification. By providing precise and unambiguous description mechanisms, formal methods facilitate the development of the critical systems. The representation used in formal methods is called a formal specification language. The formal specification languages are based on set theory and first order predicate calculus. The language has a formal semantics that can be used to express specifications in a clear and unequivocal manner. A formal specification language can be used to model the most complex systems using relatively simple mathematical entities, such as sets, relations and functions.

2.1 Types of Formal Specification Styles
There are basically three types of formal specification styles [6]; their pictorial representation is given as follows:

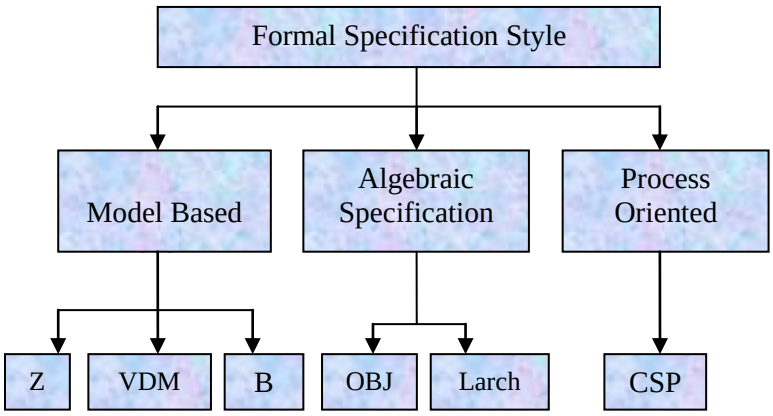


Fig. 1: Formal Specification Styles

Detailed discussion on each of them is given as follows:

A) Model Based Languages

There are various ways to write a requirement specification. One approach is model based languages. These languages specify system behavior by the construction of a mathematical model with an underlying state (data) and a collection of operations on that state [7]. The state model is constructed with the help of mathematical entities such as relations, sets, sequences and functions. Operations of a system are specified by defining how they affect the state of the system model. Operations are also described by the predicates given in terms of pre and post conditions. The most widely used notations for developing model based languages are Vienna Development Method (VDM) [8], Zed (Z) [9] and B [10].

B) Algebraic Specification

Algebraic specification is a technique, used to specify the system behavior. These languages uses methods derived from abstract algebra to specify behavior of information system. Algebraic approach was originally designed for the definition of abstract data types and interface. The most widely used notations for developing algebraic specification languages are LARCH, ASL and OBJ [11].

C) Process Oriented

Process oriented formal specification language is basically used to describe concurrent system. For concurrency, these languages are based on specific implicit model. In these languages processes are denoted by expressions and built up by elementary expressions. Elementary expressions describe simple processes by operations, which combine processes to yield more complex processes. The most widely used process oriented language is Communicating Sequential Processes (CSP) [1].

2.2 Formal Languages in SDLC

In SDLC, formal languages are basically used in two phases; requirements and testing. Their detailed study is given as follows:

A) Specification (Requirements Analysis Phase)

Specification is the major step in development life-cycle. It is the process of describing a system behavior and its desired properties. Formal specification languages describe system properties that might include functional behavior, timing behavior, performance characteristics and internal structure etc [3].

For specifying the behavior of sequential systems formal methods such as Z, VDM and Larch are used while other formal methods such as CSP, CCS, State charts, Temporal Logic, Lamport and I/O automata, focus on specifying the behavior of concurrent systems [5]. RAISE is used for handling rich state spaces and LOTOS is one of the languages for handling complexity due to concurrency.

B) Verification (Testing Phase)

Verification is the process to prove or disprove the correctness of a system with respect to the formal specification or property. For the verification of the code, there are two important forms: Model Checking and Theorem proving [5]. In model checking, a finite state model of the system is build and its state space is mechanically investigated. Two well-known and equivalent model checkers are NuSMV and SPIN.

Theorem proving is another approach for verification of a specification or program is correct. A model of the system is described in a mathematical language and desired properties of the model can be mechanically proven by a theorem prover. It is mechanization of a logical proof. The specification to be checked by a theorem prover is written in a mathematical notation, Z (pronounced 'Zed') being a well-known example.

3.A SURVEY OF EXISTING METHODS

Various formal methods are reported in the existing literature. Some methods are significantly used in requirements phase and some at software testing phase. A detailed discussion on each of them relating to requirement phase with reference to each category as discussed in earlier section is given as follows:

3.1 Model- Based Languages

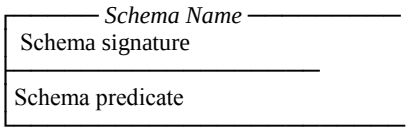
There are three major model based language reported in the literature, which are given as follows:

(a) Z-method

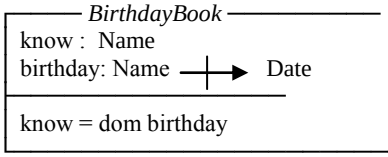
Pronounced "Zed", developed in 1977 by Jean-Raymond Abrail, Steve Schuman and Betrand Meyer; it was later further developed at the programming research group at Oxford University. This is a model based language used in requirement specification and verification phase, based on Zermelo-Frankel set-theory, lambda-calculus and first-order predicate logic.

The specification of an application using Z [9] is constructed basically by the definition of schemas as shown in Fig. 3.1(a). The schemas are used to describe both static and dynamic aspects of a system. A Z schema consists of a name, the signature part and predicate part. Attributes and the respective types of the entities being specified specifies in signature part. predicate part contains the possible values that the attributes can take. It is used to specify the state and operations of a system. Fig. 3.1(b) illustrates the example of the Z schema, that is, the BirthdayBook schema. Z specification usually made up of six main parts: the declaration of types and global variables, the state space, the initial state, the operations, the error handling schemas and the

robust versions of operations. Each of them is formed in terms of the schema.



(a) The Structure of Z Schema



(b) The *BirthdayBook* Schema

Fig. 2: A Z Schema

The declaration of types and global variable consists of basic types, free types, and axiomatic description. The state space presents the state variables and the relationships between the state variables. The initial state assigns initial values to the state variables. The operation schemas can able to change the values of the state variables or remain it unchanged. Their predicate part consists of the pre-condition and post-condition. The error handling schemas specify what happens if the pre-condition in operation schema does not hold. Lastly, the robust versions of operations define the complete operations in the sense that they are able to handle errors which might occur when invalid date is input.

(b) VDM

The Vienna Development Method (VDM) is a formal language developed at the IBM laboratories in Vienna in the mid-1970s. It is a collection of techniques for the modeling, specification and design of computer-based system. It is a model based language used in specification phase. The most popular tool for VDM today (VDMTools) is a rather useful tool for development of formal models in VDM++ or VDM-SL.

VDM describes software systems and other systems as models [12]. Models are described as objects and operations on objects. The objects represent input, output and internal state of the system. It is based on abstraction to develop top-down development of systems. Requirement specification is typically given as a rather abstract model. The objects capture only necessary properties for expressing the essential concepts of the operation of the intended software system.

A VDM model has a specific role in application areas, such as semantics of programming language, databases and construction of compilers. For expressing the models Meta-IV, which is the specification language of VDM, is used. The models are defined using a number of type definitions (for the objects) and function definitions (for the operations). Meta-IV is aimed at supporting abstraction in writing specifications. Abstraction is obtained through mathematical concepts, such as sets and functions. The abstraction provided by Meta-IV offers a set of mathematically based primitives that allow the construction of application-specific models.

When using VDM, an abstract model traditionally contains the following components [12]:

• **Semantic domains:** These types describe the objects to be operated on.

• **Invariants:** Invariants are the boolean functions that define a set of condition on the objects that is described by semantic domains.

• **Syntactic domains:** Types that define a "language" in which to express commands for manipulating the objects defined by the semantic domains.

• **Well-formedness conditions:** These are the functions that define when the commands, which is defined by the syntactic domains have a well-defined effect.

• **Semantic functions:** These functions provide the effect of commands on the objects defined by the semantic domains.

(c) B-method

The B method is developed by Jean- Raymond Abrail in 1985. It is based on Abstract Machine notation. This method is basically used for specifying, refining and implementing software [13]. A tool set is available supporting specification, design, proof and code generation [4].

The main concept follows in B- method is to initiate with an abstract model of the system under development, which roughly corresponds to the modules in many programming languages. A development process creates a number of proof obligations, which guarantee the correctness. Proof obligations can then be proven by automatic or interactive prover [14].

An abstract MACHINE of a given name contains information of CONSTANTS that describe the structure of the problem [14]. It includes PROPERTIES that describe the constants and ASSERTIONS which are the list of theorems that help in proving. The machine also includes VARIABLES, their INITIALIZATION and an INVARIANT, which describe the unchanging properties of one or more variables. A variable in an abstract machine is defined to be a variable passed into the machine from outside and which the machine modifies. The OPERATIONS clause depicts the procedures which the machine is intended to perform. The format for a machine can be described as follows:

MACHINE m

CONSTANTS k

PROPERTIES T

VARIABLES v

INVARIANT I

INITIALIZATION L

OPERATIONS

$$y \leftarrow op(x) \triangleq$$

PRE P THEN S

END;

...

END

When the machine is a refinement of the original abstract machine, then a new clause is introduced called REFINES [10]. The refined abstract machine very closely follows the original machine; however it adds more details to the design. It takes the specifications through a sequence of design steps towards implementation. The most important step in the B method is proving that the algorithm in the refined machine is correct.

3.2 Algebraic Specification

There are two basic languages relating to 'Algebraic Specification', which are given as follows:

(a)OBJ

OBJ is an algebraic programming and specification languages family introduced by Joseph Goguen in 1976. Object-oriented specification languages are those which provide at least the three basic object-oriented features (i.e. objects, classes and inheritance) and are based on a semantic model that allows formal manipulation of the specification [11].

The OBJ languages are based on algebra, first-order logic, temporal logic and set theory etc [15]. In most cases variations of different formalisms are used. A summary of approaches and their formal basis is shown in Table 1. Object-oriented specification approaches have been classified in two main groups. These are classified as follows:

A) The first one is formed by new languages that have been designed from inception following the paradigm [11].

TABLE 1. Summary of object-oriented specification languages and their formal underlying model

Name	Formal Method
TROLL	Logical framework called Object Specification Logic
CSML	Algebraic semantics. Development method based on Jackson System Development
EAM	Entity-Relationships model and Abstract Machines
Disco	Joint Actions and Statecharts
MONDEL	Operational semantics. Formal verification method based on Coloured Petri Nets
OOST	Set theory and Relational Neutral Systems ('Rest stays unchanged' semantics)
OOZE	Algebraic semantics derived from the underlying OBJ system
COLD	Operational semantics
Z++	Z-based semantics for objects and operations. Algebraic for classes and inheritance
MooZ	Z (transformational semantics)
LOOPN	Based on Coloured Petri Nets
CO-OPN	Timed Petri Nets and Distributed Transition Systems
VDM++	VDM (transformational semantics)
SDL92	SDL (transformational semantics)

B) Incorporation of object orientation into existing languages

The second group of languages reviewed is formed by object-oriented dialects of well known formal description techniques. The language Z [9] has received a lot of attention but other techniques such as Lotos, Petri Nets and VDM [8] have also been 'affected' by the object-oriented paradigm. Three different approaches are possible:

- Re-interpretation of the language by giving a set of guidelines that allow us to specify the system in an object-oriented fashion. The advantage of this approach is that proof systems and tool support available for the original languages can still be used. However, on the other hand, the extent to which they follow the object oriented paradigm may be very limited.
- Extension of its syntax with additional constructs. The formal semantics of the new constructs are normally mapped to the semantics of the non object-oriented version (transformational semantics).
- Full transformation into an object-oriented language. This involves the definition of a new language based on its precursor, but not necessarily compatible with it. Syntax and semantics have to be redefined.

(b)LARCH

The Larch family of languages supports a two-tiered, definitional style of specification. One language that is designed for a specific programming language, Larch Interface Language (LIL) and another language that is independent of any programming language, Larch Shared Language (LSL) [16].

Interface languages are used to specify the interfaces between program components. Each specification provides the information needed to use an interface. Interface helps the components to communicate. Communication mechanisms differ from programming language to programming language. For example, some languages have mechanisms for signaling exceptional conditions, others do not. Specifications written in such interface languages are generally shorter. They are also clearer to programmers who use components and to programmers who implement them.

The top level specifications are written in LSL specification language which is based on first order logic with equality and induction. The second level is a behavioural interface specification language (BISL) which is used to specify the details of a particular implementation such as procedure side-effects and aliasing using primitives defined in LSL. The tiered system gives Larch great flexibility since each target programming language uses a specifically designed BISL [17].

The basic unit of specification in LSL is a trait. A trait introduces operator names, signatures (involving type, or sort, names), and a set of axioms which define properties of the operators. Traits can be combined with other traits to structured specifications.

3.3 Process Oriented

Literature reveals only one language in this category, which is given as follows:

Communicating Sequential Process

Communicating Sequential Processes (CSP) was first described in 1978 by C. A. R. Hoare. The version of CSP presented in Hoare's original 1978 paper was essentially a concurrent programming language. It is formal language for describing patterns of interaction in concurrent systems. Programs in the original CSP were written as a parallel composition of a fixed number of sequential processes communicating with each other strictly through synchronous message-passing [1]. In contrast to later versions of CSP, each process was assigned an explicit name, and the source or destination of a message was defined by specifying the name of the intended sending or receiving process.

- **Syntax**

CSP operates over events and processes. Events are simply actions which take place while processes are made up of multiple events. Events are generally written in lower-case, while processes are generally written (at least beginning in) upper-case. CSP has two basic processes: SKIP and STOP. SKIP is used to represent successful termination while STOP represents a deadlocked process which can no longer engage in any events.

The syntax of CSP defines the "legal" ways in which processes and events may be combined. Let e be an event, and X be a set of events, then the basic syntax of CSP can be defined as:

$$
\begin{array}{lll}
Proc & ::= & STOP \\
 & | & SKIP \\
 & | & e \rightarrow Proc \qquad (prefixing) \\
 & | & Proc \,\Box\, Proc \qquad (external\ choice) \\
 & | & Proc \,\sqcap\, Proc \qquad (nondeterministic\ choice) \\
 & | & Proc \,|||\, Proc \qquad (interleaving) \\
 & | & Proc \,|[\{X\}]|\, Proc \qquad (interface\ parallel) \\
 & | & Proc \setminus X \qquad (hiding) \\
 & | & Proc; Proc \qquad (sequential\ composition) \\
 & | & \text{if } b \text{ then } Proc \text{ else } Proc \qquad (boolean\ conditional) \\
 & | & Proc \triangleright Proc \qquad (timeout) \\
 & | & Proc \,\triangle\, Proc \qquad (interrupt)
\end{array}
$$

- **Semantics**

Denotational semantics (initially known as mathematical semantics or Scott–Strachey semantics) is an approach of formalizing the meanings of programming languages by constructing mathematical objects (called denotations) which describe the meanings of expressions from the languages .Algebraic semantics is a form of axiomatic semantics based on algebraic laws for describing and reasoning about program semantics in a formal manner. Operational semantics is a way to give meaning to computer programs in a mathematically rigorous way. Operational semantics have two types: structural operational semantics (or small-step semantics) formally describe how the individual steps of a computation take place in a computer-based system. By opposition natural semantics (orbig-step semantics) describe how the overall results of the executions are obtained.

4. STRENGTHS AND WEAKNESSES

After going through the details of each methodology, some strengths and weaknesses are identified, which are given as follows:

4.1 Z-method

(a)Strength(s)

The major strengths of Z method are as follows:

- Z provides a strong base for system designing and testing due to its high level support on abstraction.
- Free tools support is available in the market.
- By using Z method, faults can be found early in the development.
- Z is used to specify the functional aspects of systems.
- Behavioural specification, i.e. describing how things change is the strong side of Z.
- The Z notation is used for the formal definition of all main control-flow testing criteria. These definitions help to eliminate the possibility of inaccuracy in understanding, which is generally happens with definitions in natural language.
- It represents both static and dynamic aspects of a system.

(b)Weaknesses

The major weaknesses of Z method are as follows:

- Specification written using Z notation cannot be used to generate computer source code directly.
- The weak side of Z notation is a lack of graphical notation.
- This method does not have the facility of exception handling.
- It does not provide support for concurrency control.
- Z does not support all aspects of design.
- Z formal specification is non- executable i.e. it does not contain information on how specified functionality should be achieved.

4.2 VDM-method

After going through the details of this technique, some strengths and weaknesses are identified, which are given as follows:

(a)Strength(s)

- The major strengths of VDM method are as follows:
- Specification written using VDM can be used to generate computer source code directly.
- Specification is comprehensive and precise thus, easy to understand.
- VDM emphasizes on the feature of concurrency control.
- Free tools support with reference to VDM is available in the market.
- VDM provides the equivalence of programming language concepts as part of compiler correctness arguments.
- VDM has the facility of explicit exception handling.

- VDM is used to specify the functional aspects of systems.
- VDM++ is based on VDM-SL and used for specification of object oriented models

(b)Weaknesses

The major weaknesses of VDM method are as follows:

- VDM method does not support all aspects of designing of the system.
- It cannot specify usability, reliability, safety and performance requirements.
- VDM Tools has lack of usability. There is no internal editor for the models. In this case the user always has to use the editor, for example Microsoft Word to change the specification.
- In VDM Tools, the error list cannot be emptied and so it is hard to see which errors are new and which have already been fixed.

4.3 B-method

After going through B-method, some strengths and weaknesses are listed, which are given as follows:

(a)Strength(s)

The major strengths of B method are as follows:

- Specification written using B can be used to generate computer source code directly.
- Free tools support is available in the market.
- B models the system in an abstract machine notation that can helps to design system generate its code.
- In this method system design is refined and tested efficiently.
- B method is a tool-supported formal methods. It is based on AMN (Abstract Machine Notation) which is used in the development of correct software.

(b)Weaknesses

The major weaknesses of B method are as follows:

- It does not provide support for concurrency control.
- B is slightly more low-level.
- It is more focused on refinement to code rather than just formal specification.
- No support for object oriented concept is provided by B-method.

4.4 OBJ-method

After going through the details of OBJ method, some strengths and weaknesses are listed, which are given as follows:

(a)Strength(s)

The major strengths of OBJ method are as follows:

- Formal specification languages become more attractive and easy to use by using object oriented concepts.
- Object orientation fills the gap between structural (static) and behavioral (dynamic) specifications.
- Inheritance helps in reusability of specification.
- Class specialization establishes refinement relations.
- Object orientation not only helps in the formal reasoning stages but also simplifies the actual writing of specifications.
- An object-oriented formal method can be integrated smoothly in the whole development cycle.

(b)Weaknesses

The major weaknesses of OBJ method are as follows:

- The adoption of the object paradigm brings powerful abstraction and structuring mechanisms but at the same time creates some additional problems for example, if we want to use the constructs such as classes, inheritance and objects in a truly formal methodology they need to be formally defined and integrated within the semantic model of the language.
- The number of interpretations and contradictory terminology makes it difficult to integrate or even compare different approaches. Attempts to introduce formalism into the paradigm have often been hampered by these terms and concepts.

4.5 LARCH

After going through this approach, some strengths and weaknesses are identified, which are given as follows:

(a)Strength(s)

The major strengths of LARCH are as follows:

- It is used to specify interfaces between programs in different languages.
- Larch family is able to cope with the features of different implementation languages.
- It supports the feature of abstraction.
- Larch interface languages encourage a style of programming that emphasizes the use of abstractions, and each provides a mechanism for specifying abstract types.
- To make it possible to validate specifications before implementing or executing them, Larch permits specifiers to make assertions about specifications.

(b)Weaknesses

- Some larch specifications can be executed; most of them cannot.
- Larch style of specification emphasizes simplicity and clarity rather than executability.
- It is not focused to the major development phase i.e. requirement phase.

4.6 Communicating Sequential Process

After going through the details of this approach, some strengths and weaknesses are identified, which are given as follows:

(a)Strength(s)

The major strengths of CSP are as follows:

- CSP is focused to verify safety and liveness properties of the system.
- CSP allows the description of systems in terms of component processes that operate independently, and interact with each other through message-passing communication.
- Main focus of CSP is in software design and it is usually used to design safety-critical systems.
- CSP is well-suited to modeling and analyzing systems that incorporate complex message exchanges, it has also been applied to the verification of communications and security protocols.
- There exist a number of tools for analyzing and understanding systems described using CSP.

(b)Weaknesses

The major weaknesses of CSP are as follows:

- CSP is well-suited to design software; it is not focus at requirements specification phase.
- Working of CSP is very difficult to understand.
- It is only used in concurrent system.
- It needs at least two processes i.e. sending and receiving process for communication.
- A subordinate process needs to know the name of its using process; this complicates construction of libraries of subordinate processes.

5. COMPARATIVE STUDY

For accomplishing a comparative study of the aforementioned methodologies, some attributes have been identified based on the well known practices with similar cases. These are described as follows:

- **Concurrency Control:** Concurrency is a property used in distributed system that enables software systems to be served in large-scale distributed systems. This property allows several computations to execute simultaneously, and potentially interact with each other.

- **Supporting Tools**: It helps in automation of any process. Supporting tools makes the steps easier; therefore, tools support is highly recommended.

- **Support for Abstraction:** Abstraction is the process by which data and programs are defined with a representation similar in form to its meaning (semantics), while hiding away the implementation details. Abstraction captures only essential details about an object that are relevant to the current perspective.

- **Object Oriented Concept:** The object oriented concepts such as inheritance, polymorphism, and encapsulation are supported by some formal specification languages. Object oriented programming is an approach for developing software system based on the concepts of classes and objects.

- **Structuring**: It is a mechanism for combining specifications, for example, to handle error handling or status information.

- **Requirements Phase Perspective**: Requirements phase is the backbone of any software to be developed [18]. As it is well accepted by the research community, it is necessary for any methodology to consider this perspective.

On the basis of the review results i.e. strengths and weaknesses of each methodology, a comparative study of afore mentioned formal methods are done. A table is made for the comparative study of all the formal methods. If the formal method fully satisfies an attribute, a mark √ is drawn against the column, otherwise × is marked.

Table 2. Comparison of Formal Methods

Attribute	Z-Method	VDM	B-Method	OBJ	Larch	CSP
Concurrency Control	×	√	×	√	×	√
Supporting Tools	√	√	√	√	√	√
Support for abstraction	√	√	√	√	√	√
Object Oriented Concept	√	√	×	√	√	√
Structuring	√	×	√	√	√	√
Requirements Phase Perspective	√	√	×	×	√	×

6. FUTURE RESEARCH DIRECTIONS

Based on this critical review, strengths, and weaknesses of existing formal methods, we have drawn some future research directions, which are given as follows:

- In Z-Method, further research may be done to incorporate refinement techniques at requirement phase to ensure the correctness, completeness and consistency of specification.
- In VDM, further research may be undertaken for inclusion of a structure, making more useful for conceptual phases of SDLC e.g. requirements phase.
- In B-method, future research may be done on developing an AMN (Abstract Machine Notation) that should be more specific for requirements phase.
- In OBJ, future research may be done on inclusion of other attributes like authenticity, non-repudiation, incorporating threat related values in the formula, making the methodology more specific for requirements phase, along with a validation report.
- In LARCH, future research may be done to include semantic type checking in specification. Tools for checking interface specifications should be integrated with other programming language tools for specify arbitrary programs.
- In Communicating Sequential Process (CSP), future research may be done to integrate different approaches for specification, design, implementation, verification and validation of complex computer systems.

7. CONCLUSION

The paper presented strengths, weaknesses preceded by a comparative study of the existing formal methods along with the future research directions. Decision making for selecting a formal method can be easily done by the senior IT personnel by going through the results derived in the paper. Researchers have made significant progress in the area of formal methods but still improvement is needed. We presented a number of research areas, based on the existing published work in which further work is required such as composition, decomposition, abstraction, combinations of mathematical theories, reusable models and theories. Among all, major need is to make the methods to be more specific for the requirements phase because requirements are considered as foundation stone on which the entire software can be built. This work may help to

provide effective and efficient ways to incorporate formal methods in requirements phase so that appropriate level of security can be achieved. Possible future extension of the work has already been discussed exhaustively in the above section.

8. REFERENCES

[1] M. Clarke Edmund, M. Wing Jeannette: Formal Methods: State of the Art and Future Directions, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. Retrieved on April, 8, 2013. http://www.site.uottawa.ca/~bochmann/ELG7187C/CourseNotes/Literature/Clarke%20%20FM%20State%20of%20the%20art%20-%2096.pdf

[2] Fuxman Ariel Damian: Formal Analysis of Early Requirements Specifications, Graduate Department of Computer Science, University of Toronto, 2001. Retrieved on April, 8, 2013 http://wenku.baidu.com/view/806eb2b8fd0a79563c1e72b5.html

[3] Woodcock Jim, Larsen Peter Gorm, Bicarregui Juan and Fitzgerald John: Formal Methods: Practice and Experience, ACM Computing Surveys(CSUR), Volume 41 Issue 4, October 2009 Article No. 19. Retrieved on: March 17, 2013. http://deployeprints.ecs.soton.ac.uk/161/2/fmsurvey%5B1%5D.pdf

[4] Pressman Roger S.: "Software Engineering"- A Practitioner's Approach", McGraw Hill, 5th edition. 2000.

[5] Ogilvie Paul: Formal Methods in Requirements Engineering. Retrieved on: March, 17,2013 http://www.google.co.in/url?sa=t&rct=j&q=paul%20ogilvie%20formal%20methods%20in%20requirement%20engineering&source=web&cd=1&cad=rja&ved=0CC4QFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.93.8000%26rep%3Drep1%26type%3Dpdf&ei=ispFUeXGBNCzrAfP6oHgDQ&usg=AFQjCNHuCyMppgBZ5cxA0QAOhfOIYZL8Lw&bvm=bv.43828540,d.bmk

[6] Kaur Arvinder, Gulati Samridhi and Singh Sarita: Analysis of Three Formal Methods-Z, B and VDM, International Journal of Engineering Research & Technology (IJERT) Vol. 1 Issue 4, June – 2012. Retrieved on April, 8,2013. https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CDMQFjAA&url=http%3A%2F%2Fwww.ijert.org%2Fbrowse%2Fjune-2012edition%3Fdownload%3D297%253Aanalysis-of-three-formal-methods-z-b-and-vdm%26start%3D120&ei=4nNlUZueJdDOrQf8qoDwCw&usg=AFQjCNEXxBMpcbJpIDSCfIUOPa0_iIrAUQ&sig2=wDPHf2gNVeatHhKwh1xGbQ

[7] McGibbon Thomas: An Analysis of Two Formal Methods: VDM and Z, ITT Industries - Systems Division.

[8] Muller Andreas: VDM: The Vienna Development Method, April 20, 2009. Retrieved on April, 7, 2013. https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CDMQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.153.3064%26rep%3Drep1%26type%3Dpdf&ei=FXNlUen9DobqrAfgroCwAg&usg=AFQjCNGYrKJJCI1xnt3tvME2JQFyDe1jaQ&sig2=LpGghBpScvnLXpXuxMxXyQ&bvm=bv.44990110,d.bmk

[9] Spivey J. M.: The Z Notation: A Reference Manual. 2nd Edition, New York,Prentice-Hall, (1998). Retrieved on March, 19, 2013 http://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&sqi=2&ved=0CDgQFjAB&url=http%3A%2F%2Ftrac.assembla.com%2Fci4712%2Fexport%2F54%2FZReferenceManual.pdf&ei=kDNIUfPJNsW8rAfl94CgCw&usg=AFQjCNEsUr2UIEtvntbzhFR9LoVWzGC6Rw&sig2=CrXzdXFsi5PvCc5FQgHR4w&bvm=bv.43828540,d.bmk

[10] B method - An overview through example Ewa Romanowicz, McMaster University, Hamilton, Ontario, Canada April 17, 2008. Retrieved on April, 6, 2013. http://www.ewaromanowicz.com/academics/B_method_An_overview_through_example.pdf

[11] Delgado Antonio Ruiz, Pitt David and Smythe Colin: A Review of Object Oriented Approaches in Formal Methods, The Computer Journal, Vol. 38, NO 10, 1995.

[12] Pedersen Jan Storbank and Klein Mark H.: Using the Vienna Development Method (VDM) To Formalize a Communication Protocol, Technical Report, November 1988. Retrieved on April, 6, 2013. ftp://ftp.sei.cmu.edu/pub/documents/88.reports/pdf/tr26.88.pdf

[13] Schneider S.: The B-Method: An Introduction. Cornerstones of Computing, Palgrave, 2001. Retrieved on April, 5,2013. http://www.amazon.com/B-Method-Cornerstones-Computing-Steve-Schneider/dp/033379284X

[14] Bjorner Dines and C. Henson Martin: Logics of Specification Languages. Springer Berlin Heidelberg, 2007.

[15] Wikipedia. OBJ. Retrieved on March 18, 2013. From http://en.wikipedia.org/wiki/OBJ_(programming_language)

[16] Guttag J.V., Horning J.J., Garland S.J., Jones K.D., Modet A., and Wing J.M.: Larch: Languages and Tools for Formal Specification, January 1993. Retrieved on April 10, 2013. http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/LarchBook.pdf

[17] Dunstan Martin, Kelsey Tom, Linton Steve and MartinL Ursula: Lightweight Formal Methods For Computer Algebra Systems, University of St Andrews, UK Retrieved on April 10, 2013 http://www.eecs.qmul.ac.uk/~uhmm/papers/AdGotMar.pdf

[18] Cheng, Betty H C, Atlee, Joanne M: Research Directions in Requirements Engineering, IEEE, May 2007, pp 285 – 303.