Systematic Review of Verification & Validation in Dynamic Languages

Farrakh Saeed
Muhammad Saeed
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 32 weeks of full time studies.

Contact Information:
Author(s):
Farrakh Saeed
Address: Kamnarsvagen 13B, B102, 22646 Lund, Sweden
E-mail: fasa06@student.bth.se

Muhammad Saeed
Address: Kamnarsvagen 13B, B102, 22646 Lund, Sweden
E-mail: mr.msaeed@gmail.com

University advisor(s):
Dr. Robert Feldt
School of Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 58 87
Fax :
ABSTRACT

The Verification and Validation provides support to improve the quality of the software. Verification and Validation ensures that the product is stable and developed according to the requirements of the end user. This thesis presents a systematic review of dynamic programming languages and verification & validation practices used for dynamic languages. This thesis presents results found in dynamic programming languages and verification & validation over the period of 1985 – 2008. The study is aimed to start from identification of dynamic aspects along with the differences between static and dynamic languages. Furthermore, this thesis is also intends to give overview of the verification and validation practices for dynamic languages.

Moreover to validate the verification and validation results, a survey consisting of (i) interviews and (ii) online survey is conducted. After the analysis of systematic review, it has been found that dynamic languages are making progress in some of the areas like integration of common development framework, language enhancement, dynamic aspects etc. The Dynamic languages are lacking in providing a better performance than static languages. There are also some factors found in this study that can raise the popularity of dynamic languages in the industry.

Based on the analysis of systematic review, interviews and online survey, it is concluded that there is no difference between the methodologies available for Verification and Validation. It is also revealed that dynamic languages provide support to maintain software quality with their characteristics and dynamic features. Moreover, they also support to test softwares developed with static language.

It is concluded that test driven development should be adopted while working with the dynamic languages. Test driven development is supposed to be a mandatory part of dynamic languages.

Keywords: Dynamic programming languages, Verification and Validation.
ACKNOWLEDGEMENTS

We would like to extend our sincere gratitude to our advisor Dr. Robert Feldt. His guidance, criticism, support and appreciation helped us to achieve our goals. Dr. Robert Feldt has provided us detail feedbacks to improve our quality of work and we also thank him for his support to get contacts from the industry. We really admire his ability of always being there, despite his busy schedule and even when he is out for conferences as well.

We also appreciate the efforts and positive feedback of our advisor and Muhammad Naeem (PhD student) for evaluating our review protocol. We would like to express our gratitude to Ola Bini, Jimmy Nilsson and Nicolas for their cooperation for conducting industrial interviews. We also appreciate the respondents who have spared some time for our thesis and give their experiences and thoughts related to our topic.

Colleagues and friends at Blekinge Tekniska Högskola also deserve thanks. We want to appreciate all the encouragement and nice memories during our stay in BTH. We appreciate the efforts of Muhammad Naeem (PhD Student) and Nasir Majeed Awan for proof reading of our thesis. We would also like to thank our friends Uzair Akbar Raja, Sheraz Ahmad, Waqas Siddique and Nasir Majeed Awan for encouraging us throughout the thesis tenure.

Our special thanks are for our parents, brothers and sisters for scarifying their wonderful time by sending us abroad in the quest of knowledge. They are the best who provide us an opportunity to make our future bright and prosperous.
CONTENTS
SYSTEMATIC REVIEW OF VERIFICATION & VALIDATION IN DYNAMIC LANGUAGES........................................... I
ABSTRACT ..................................................................................................................................................... I
ACKNOWLEDGEMENTS .............................................................................................................................. II
CONTENTS .................................................................................................................................................. III
TABLE OF FIGURES ............................................................................................................................... VI
TABLE OF TABLES ................................................................................................................................... VII

1. INTRODUCTION .................................................................................................................................. 1
1.1 PURPOSE ......................................................................................................................................... 1
1.2 AIMS AND OBJECTIVES .................................................................................................................. 2
1.3 RESEARCH QUESTIONS .................................................................................................................... 2
1.4 RELATIONSHIP AMONG DIFFERENT STEPS OF RESEARCH METHODOLOGY AND OBJECTIVES ..... 2
1.5 RESEARCH METHODOLOGY .......................................................................................................... 3

2. DYNAMIC PROGRAMMING LANGUAGES ....................................................................................... 5
2.1 DYNAMISM CRITERIA ........................................................................................................................ 5
2.1.1 Dynamic Aspects .......................................................................................................................... 5
2.2 DIFFERENCES BASED ON USE, FEATURES AND DYNAMIC ASPECTS ....................................... 6
2.2.1 Differences with Use ..................................................................................................................... 6
2.2.2 Differences based on Features ...................................................................................................... 8
2.2.3 Differences based on Dynamic Aspects .......................................................................................... 9

3. SYSTEMATIC REVIEW ....................................................................................................................... 10
3.1 SYSTEMATIC REVIEW ..................................................................................................................... 10
3.1.1 Planning the Systematic Review ............................................................................................... 10
3.1.2 Systematic Review Protocol ....................................................................................................... 10
3.1.2.1 Search Process/Strategy .......................................................................................................... 10
3.1.2.2 Search Terms .......................................................................................................................... 11
3.1.2.3 Study Selection Criteria ........................................................................................................ 13
3.1.2.3.1 Inclusion Criteria ............................................................................................................... 13
3.1.2.3.2 Exclusion Criteria ............................................................................................................. 14
3.1.2.4 Selection Process ..................................................................................................................... 14
3.1.2.5 Study Quality Assessment .................................................................................................... 14
3.1.2.6 Data Extraction Strategy ....................................................................................................... 15
3.1.2.7 Synthesis of Extracted Data ................................................................................................. 15
3.1.3 Conducting the Systematic Review .............................................................................................. 15
3.1.3.1 Screening Process .................................................................................................................. 15
3.1.3.2 Examples of Selection Procedure .......................................................................................... 16
3.1.3.2.1 Title Rejection ............................................................................................................... 16
3.1.3.2.2 Abstract Rejection ........................................................................................................... 16
3.1.3.2.3 Conclusion Rejection ..................................................................................................... 16
3.1.3.2.4 Whole Article ................................................................................................................ 16
3.1.3.3 Systematic Review Statistics .............................................................................................. 17
3.1.3.4 Classification Scheme ........................................................................................................... 17
3.1.4 Reporting the Systematic Review ............................................................................................... 17

4. SYSTEMATIC REVIEW RESULTS ...................................................................................................... 18
4.1 DEFINITIONS/ GENERAL IDEA ABOUT DYNAMIC LANGUAGES ................................................ 18
4.1.1 Analysis ........................................................................................................................................ 18
4.2 ISSUES CONCERNING DYNAMIC LANGUAGES .......................................................................... 18
4.2.1 Analysis ........................................................................................................................................ 19
4.3 LISP DEVELOPMENTS ............................................................................. 19
4.3.1 Comparison .................................................................................. 19
4.3.2 Dynamic Aspects ......................................................................... 20
4.3.3 Framework .................................................................................. 20
4.3.4 Language Development/Enhancement ........................................ 20
4.3.5 Performance Issues ..................................................................... 21
4.3.6 Usage .......................................................................................... 21
4.3.7 Analysis ...................................................................................... 22
4.4 ERLANG DEVELOPMENTS................................................................. 23
4.4.1 Comparison ................................................................................ 23
4.4.2 Dynamic Aspects ......................................................................... 23
4.4.3 Integration .................................................................................. 24
4.4.4 Language Development/Enhancement ........................................ 24
4.4.5 Performance Issues ..................................................................... 24
4.4.6 Usage .......................................................................................... 25
4.4.7 Analysis ...................................................................................... 25
4.5 RUBY DEVELOPMENTS ..................................................................... 26
4.5.1 Integration .................................................................................. 27
4.5.2 Comparison ................................................................................ 27
4.5.3 Enhancement/Development ....................................................... 27
4.5.4 Usage .......................................................................................... 27
4.5.5 Self Development (Performance) ................................................. 28
4.6 PYTHON DEVELOPMENTS ................................................................. 28
4.6.1 Comparison ................................................................................ 28
4.6.2 Dynamic Aspects ......................................................................... 28
4.6.3 Framework ................................................................................ 29
4.6.4 Language Development/Enhancement ........................................ 29
4.6.5 Integration Support .................................................................... 29
4.6.6 Performance ............................................................................... 30
4.6.7 Usage .......................................................................................... 30
4.6.8 Analysis ...................................................................................... 31
4.7 SMALL TALK DEVELOPMENTS ......................................................... 31
4.7.1 Comparison ................................................................................ 31
4.7.2 Dynamic Aspects ......................................................................... 32
4.7.3 Framework ................................................................................ 32
4.7.4 Language Development/Enhancement ........................................ 32
4.7.5 Integration Support .................................................................... 32
4.7.6 Performance Related ................................................................. 33
4.7.7 Usage .......................................................................................... 33
4.7.8 Analysis of Smalltalk .................................................................. 34
4.8 CONCLUSION DRAWN FROM SYSTEMATIC REVIEW ..................... 34
4.9 ANSWERING THE RESEARCH QUESTIONS STATED IN REVIEW PROTOCOL .. 35
5. RESULTS REGARDING VERIFICATION AND VALIDATION IN DYNAMIC LANGUAGES ................................................................. 36
5.1 ERLANG ............................................................................................ 36
5.2 PYTHON ........................................................................................... 38
5.3 SMALLTALK .................................................................................... 38
5.4 ANALYSIS OF VERIFICATION AND VALIDATION ......................... 38
6. SURVEY .............................................................................................. 40
6.1 QUESTIONNAIRE BASED ONLINE SURVEY .................................. 40
6.1.1 Methodology ............................................................................. 40
6.1.1.1 Objectives ............................................................................. 40
6.1.1.2 Description .......................................................................... 40
6.1.1.3 Method ............................................................................... 40
6.1.1.4 Population & Response Rate ................................................ 40
6.2 QUESTIONNAIRE OUTCOMES ........................................................ 41
6.2.1 Warm-up/Introduction ............................................................... 41
6.2.2 Verification & Validation and Dynamic Languages .................... 42
6.2.2.1 Dynamic Languages Support/Hinder to Maintain Quality ....... 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2.2</td>
<td>Dynamic Languages Aspects Important for V&amp;V</td>
<td>42</td>
</tr>
<tr>
<td>6.2.2.3</td>
<td>Difference in V&amp;V Methodologies between DLs and Static Languages</td>
<td>43</td>
</tr>
<tr>
<td>6.2.2.4</td>
<td>Aspects influencing the Popularity of Dynamic Languages</td>
<td>43</td>
</tr>
<tr>
<td>6.2.2.5</td>
<td>Problems/Issues</td>
<td>44</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Techniques/Tools</td>
<td>44</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Suggestions/Opinions</td>
<td>45</td>
</tr>
<tr>
<td>6.3</td>
<td>ANALYSIS</td>
<td>45</td>
</tr>
<tr>
<td>7.</td>
<td>INTERVIEWS FOR VERIFICATION AND VALIDATION IN DYNAMIC LANGUAGES</td>
<td>47</td>
</tr>
<tr>
<td>7.1</td>
<td>INTERVIEWEE’S INTRODUCTION</td>
<td>47</td>
</tr>
<tr>
<td>7.2</td>
<td>VERIFICATION AND VALIDATION</td>
<td>47</td>
</tr>
<tr>
<td>7.3</td>
<td>PROBLEMS/ISSUES</td>
<td>47</td>
</tr>
<tr>
<td>7.4</td>
<td>TECHNIQUES/TOOLS</td>
<td>47</td>
</tr>
<tr>
<td>7.5</td>
<td>ANALYSIS OF RESULTS</td>
<td>48</td>
</tr>
<tr>
<td>7.6</td>
<td>RESEARCH QUESTIONS ADDRESSED BY ONLINE SURVEY AND INTERVIEWS</td>
<td>48</td>
</tr>
<tr>
<td>8.</td>
<td>DISCUSSION</td>
<td>50</td>
</tr>
<tr>
<td>8.1</td>
<td>THREATS TO VALIDITY</td>
<td>51</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Conclusion Validity (Reliability)</td>
<td>51</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Construct Validity</td>
<td>51</td>
</tr>
<tr>
<td>8.1.3</td>
<td>Internal Validity (Causality)</td>
<td>51</td>
</tr>
<tr>
<td>8.1.4</td>
<td>External Validity (Generalizability)</td>
<td>52</td>
</tr>
<tr>
<td>9.</td>
<td>EPILOGUE</td>
<td>53</td>
</tr>
<tr>
<td>9.1</td>
<td>CONCLUSIONS</td>
<td>53</td>
</tr>
<tr>
<td>9.2</td>
<td>RESEARCH QUESTIONS REVISITED</td>
<td>54</td>
</tr>
<tr>
<td>9.3</td>
<td>FUTURE WORK</td>
<td>55</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>A QUESTIONNAIRE FOR ONLINE SURVEY</td>
<td>65</td>
</tr>
</tbody>
</table>
TABLE OF FIGURES

Figure 1: Relationship among different steps of research methodology and objectives 3
Figure 2: Graphical Representation of Results ...................................................... 23
Figure 3: Representation of publications according to development Scheme ............. 26
Figure 5: Representation of respondents’ education level .................................. 41
Figure 6: Representation of respondents’ involvement with DLs ....................... 41
Figure 7: Representation of respondents’ experience in industry ....................... 42
Figure 8: Representation of respondents’ involvement in V&V ......................... 42
Figure 9: Representation of survey responses to popularity influencers for DLs .... 44
# TABLE OF TABLES

Table 1: Differences based on Use............................................................................................................. 7
Table 2: Differences based on Features ........................................................................................................ 9
Table 3: Search Terms ...................................................................................................................................... 11
Table 4: Analysis results according to Issue and Level of Criticism ................................................................. 19
Table 5: Usage publications .......................................................................................................................... 21
Table 6: Major Research Classification Scheme’s .......................................................................................... 22
Table 7: Usage Classification according to Application area .......................................................................... 22
Table 8: Usage Publications .......................................................................................................................... 25
Table 9: Major Research Classification Scheme’s .......................................................................................... 26
Table 10: Usage Classification according to Application area ......................................................................... 26
Table 11: Usage .................................................................................................................................................. 27
Table 12: Self Development (Performance) ....................................................................................................... 28
Table 13: Comparison ....................................................................................................................................... 28
Table 14: Dynamic Aspects ............................................................................................................................. 28
Table 15: Framework ......................................................................................................................................... 29
Table 16: Language Development/Enhancement .............................................................................................. 29
Table 17: Integration Support .......................................................................................................................... 29
Table 18: Performance ....................................................................................................................................... 30
Table 19: Usage .................................................................................................................................................. 30
Table 20: Python Analysis ................................................................................................................................ 31
Table 21: Dynamic Aspects ............................................................................................................................. 32
Table 22: Smalltalk Framework ......................................................................................................................... 32
Table 23: Language Development/Enhancement .............................................................................................. 32
Table 24: Integration Support .......................................................................................................................... 32
Table 25: Conclusion from Systematic Review .................................................................................................. 34
Table 26: Verification and Validation in Erlang ................................................................................................. 36
Table 27: Verification and Validation in Python ................................................................................................ 38
Table 28: Verification and Validation in Smalltalk ............................................................................................ 38
Table 29: Test Tools .......................................................................................................................................... 39
Table 30: Important Aspects for Verification & Validation ............................................................................... 43
Table 31: Tools/Frameworks ............................................................................................................................ 44
1. INTRODUCTION

The static languages are normally compiled and type checked before program execution and these are the most commonly used programming languages nowadays [2]. The static programming languages bound programmers to write code before execution and do not give support to change the code while execution. According to [2] [3] [7], dynamic programming languages are weakly typed languages and facilitate to define variables but eliminate the need to explicitly declare them before their use. The dynamic languages aid to extend the programs by adding new code, extending objects and definitions or modifying the type system, all during program execution. The use of dynamic programming languages increases the flexibility for the programmer to test different programming logics and check the behavior of the system during execution. This flexibility leads to an advantage of getting quick results through compact code [2], an attraction for programmers.

Dynamic programming languages are observed as fashion rather than properly following the known procedures of measurement, validation, root-cause analysis, and defect prevention [2]. According to [5], the dynamic languages cause more run time checks and hence induce runtime costs. However, they help in reducing the development time, a strong reason of their popularity for rapid application development. The software developers desire rapid development of applications as well as a strong focus on quality [6]. Dynamic languages have high expressive power and have no type declarations. In case of static languages the situation is opposite, as the controls applied earlier in development phase cause an increase in development time. But it helps in minimizing time required during maintenance. Based on different studies we may provide some characteristics that are likely to exist in all or most of the DLs. These features include but not limited to technical purity, optimizing person time, open source, platform neutrality etc [8].

Verification and Validation is the process of software engineering which improves the quality of final software product. Verification is the process in which we investigate that requirements are mapped accordingly as mentioned in start of the phase. Verification helps to find solution of question “Are we building the product right?”. Whereas validation is the process in which we investigate during or at the end of the development process whether it fulfills the specified requirements. Validation helps to solve the question “Did we build the right product?” [1].

Verification starts from the requirements phase including elicitation, analysis, and specification. The inspection process is mainly followed by organizations who intend to do verification. This process involves, inspecting requirements documents, writing test cases from requirements, writing a user manual and then defining acceptance criteria [1].

There has been an increasing trend in the use of dynamic languages, so it will be of interest to the professionals to discern quality essentials related to software development in dynamic programming languages. It will be helpful for researchers and programmers to find a research that includes the state of the art about verification and validation and its ease or difficulty to maintain the quality of software product in dynamic programming languages. There is a need to investigate the gap, if there exists any, dealing with the popularity differences between static and dynamic programming languages and their impact on verification and validation processes. There is a need to identify the rationales behinds this gap so as to provide a focused line of action for future researchers and practitioners.

1.1 Purpose

The purpose of this thesis is to offer visibility into the dynamic programming languages, their development in recent years to investigate the verification and validation processes in dynamic programming languages.
1.2 Aims and Objectives

The aim of this thesis is to investigate the verification and validation processes in dynamic programming languages. To meet this goal, following objectives will be achieved.

- Identify the factors which make the languages static or dynamic.
- Investigate the difference between static and dynamic programming languages with respect to software development life cycle.
- Analyze the current state of the art in research on dynamic programming languages.
- To analyze the verification and validation processes in dynamic programming languages.
- Investigate the difference between popularity graphs of dynamic versus static programming languages and study the rationale behind. What role verification and validation processes play here?

1.3 Research Questions

Following research questions will be answered during systematic review and survey:

- **RQ.1** What is the current state of the art in research on dynamic programming languages and their comparison with static languages?
- **RQ.2** What are the aspects of dynamism for a dynamic programming language?
- **RQ.3** Does dynamic languages support or create hindrance to maintain the software quality?
- **RQ.4** What are the popularity factors necessary to raise the graph of dynamic languages?
- **RQ.5** What are the differences between the verification and validation techniques and tools for static and dynamic languages?

1.4 Relationship among Different steps of Research Methodology and Objectives

The following Fig 1 shows the relationship among different steps of our method of research and the objectives.
1.5 Research Methodology

The thesis will be progressed in three steps, starting from over viewing the literature about dynamic languages to find out different dynamism aspects according to a certain predefined criteria. Afterwards systematic review [4] will be next step in which current state of the art in research on dynamic programming languages will be explored. After getting material, the research will focus on verification and validation in dynamic programming languages, in detail.

The last step is a survey which will be carried out in two ways i.e. through interviews and online survey. A set of three to four professionals will be selected to conduct interviews related to verification and validation with respect to static and dynamic languages. The questionnaire will encompass different issues of verification and validation including techniques, methods, ease or difficulty of techniques etc. In parallel, the respondents will be selected for online survey. Then through the comparison of current state of art in research and industry professional’s view, it will be analyzed, whether the use of dynamic languages supports or hinders the verification and validation in maintaining the quality of software.
product.
2. **DYNAMIC PROGRAMMING LANGUAGES**

This chapter explains the dynamic aspects in dynamic programming languages and their differences to the static languages with respect to features, uses and dynamic aspects.

2.1 **Dynamism Criteria**

After careful analysis and understanding of related articles, the authors have tried to explain the dynamism concept. According to our understanding of word ‘dynamic’ in terms of programming languages is that when things are taken care at runtime then every aspect that belongs to those implications could be considered as dynamic aspect. Hence, keeping in mind the general understanding of ‘dynamic’, we highlighted some of the runtime aspects which can be considered as dynamic aspects.

2.1.1 Dynamic Aspects

Traditional software development is carried out by different programming languages, static as well as dynamic programming languages. Dynamic Programming Languages have been around since 1960’s but could not impress industry due to various issues including but not limited to extensive memory usage and runtime overhead issues. Typing is generally considered the most promising factor in distinguishing a language either to be statically or dynamically typed. In statically typed languages, the type checking is done at compile time e.g. variable type declarations which do not change during the whole execution. Static typing refers to define both the variable and its type at the time of declaration. In case of inappropriate type, it leads towards a type error which is checked at compile time and also requires a great deal of code to handle inappropria
te types [9]. Type checking aids to ensure the compatibility between the operands and operator.

In dynamically typed languages, the case is opposite; the type is checked at runtime. It means while declaring a variable there is no need to specify its type. Dynamic typing enables the programmer to solve the complex problems without imposing any variable type limitations. It creates flexibility for the programmers to play with variables by assigning values of different data types and it helps in faster developments.

The concept of dynamic typing can be better understood with the example of Eval feature in various programming languages which takes a string as an argument, process it with an expression and gives the result of that expression. Distributed programs are also typeless until runtime which exchange the information between different processes [10]. Dynamic programming languages support to alter the object and types on runtime. Class hierarchies, structures and other behavioral aspects of the program can be altered at runtime. For instance: new type addition, modification in existing function or addition of new function and variables in classes and objects [11].

Due to the syntactical issues, dynamically typed languages are quite small and flexible in use. The development of domain specific languages (DSL) is quite easy as compared to statically typed languages. The DSL developed from dynamically typed language is maintainable and evolutionary, resulting in better management of user’s ever changing needs [12]. The advantage with dynamic typing is simplification in design and also gives support in rapid development as well as testing and prototyping. [13].

Additionally, the programmers are more productive while writing code due to the aid of dynamic typing. Programmers do not have to focus on issues like memory management, and garbage collection etc. It gives programmer the flexibility to solve the problem without imposing constraints like variable type [9]. Dynamic typing also gives languages some more dynamic aspects like automatic memory management. Most of the dynamic languages are weakly typed. When the notion weak typing comes into mind then it means that it uses dynamic typing. Weakly typed sometimes also called type-less which support dynamism in
dynamic programming languages. Another aspect of dynamicity is dynamic memory allocation in dynamic languages. Rationale behind this thought is the runtime type detection of methods, variables and objects and hence memory allocation does not happen until runtime.

The Dynamic type binding is the dynamic dialect of dynamic programming language, which in fact raises their popularity. Using the concept of dynamic binding the programmer can write a generic code to handle different data types under one roof. One of the advantages is that the input of any data type is acceptable; the variable will be bound to any specific data type after having assignment of data. Dynamic binding provides a high degree of flexibility which results in advantageous ability for clients to request an operation without considering the alternatives. The choice of selection is only possible at runtime. This technique is particularly attractive for large systems where it is possible to have different alternative [14].

Code optimization could also be considered as a dynamic aspect of DPL, as the dynamic languages don’t need any declaration of the variables or objects to make them type specific. This helps to reduce the overhead of code written for type declarations which also reduces the size of code.

2.2 Differences based on Use, Features and Dynamic Aspects

2.2.1 Differences with Use

The debate for typing is ongoing to prove it as strength or weakness. The typing feature has its strengths and it compels a programmer to use the system programming language. When the application requires more reliability and involves criticality. But the absence of type specificity, balances this advantage by providing ease in programming and programmer’s productivity [15] [16]. The static programming languages are less flexible towards modification as compared to the dynamic programming languages. It is difficult to apply changes through strong typing due to different constrains [17].

The syntax of dynamic programming languages shows elegance and simplicity. But people think that programs with no type system will be difficult to debug. Hence there are greater chances for semantic errors in dynamic languages. In dynamic programming languages, weak typing ability or no type existence helps in writing a clean code [16].

However the dynamic languages have their own powerful features that help to maintain their safety from some specific errors. For example the null pointers errors, stack overflows, and type mismatch errors may crash programs in static languages. But these things are properly taken care in dynamic programming languages [17]. For example, the stack over flow error does not occur due to the feature of dynamic memory allocation in dynamic programming languages. As these languages allocate memory size according to the assigned value which does not allow variable to exceed from its limit.

Earlier, the only means of getting a time efficient application was the use of static programming language [15]. It was believed that the static programming languages are only source of producing a reliable piece of code. But recent studies show that this trend is changing [18]. The research shows that the dynamic programming languages are perfectly capable of producing very reliable software as compared to static languages. One of the reasons is the advent of modern and cheaper hardware that can perform faster enough to compute efficiently.

Structural Conformance exists among objects of compatible methods and these methods can be interchangeably used. It is a built-in feature in dynamic programming languages e-g Smalltalk-80. Structural conformance helps in reducing coupling between interfaces and their uses. Static languages may make use of this facility with special strategy implementation, for example [19].

The graphical user interface developed in C or C++, for example, are hard to learn, modify or maintain. On the contrary almost all the modern applications make use of dynamic
languages for GUI support. Because GUI mostly involves gluing of application components together and dynamic languages are best suitable for gluing. Moreover they are easy to modify and to add behaviors to controls, is much easier as compared to GUIs in static programming languages [15].

The dynamic languages provide ease to programmer in terms of producing quick code and hence support the rapid application development [15]. That’s why dynamic programming languages are first choice in Rapid Application Development environments. An interesting benefit of dynamic languages is their support towards software reuse. Together with gluing concept discussed earlier, the software reusable components, e.g. scripts, help to increase the programmer productivity [15]. Although the software reusability is also available in modern static programming languages but the flexibility of dynamic programming languages have another advantage. They relieve the programmer from spending worthwhile time in struggling with types and other syntax rigidities of static language code [20].

The myth that the time efficient programs can only be built in static programming languages, is becoming suppressed with both, advancement in hardware and maturity of dynamic languages. It is a fact that the program execution speed is faster in static (system) programming languages because most of the error detection process is done during compile time [19]. While in dynamic languages, the execution time is comparatively more than the static programming languages. The rationale is their fundamental feature of runtime code handling. Though most of the large scale applications are currently built using static programming languages but with advanced hardware support, it seems that the next era will be of dynamic languages.

It will be something interesting to discuss applicability of static languages and which kind of applications are developed using dynamic programming languages. The tasks where complexity lies in connections, then the dynamic languages also referred as scripting languages are more suitable for gluing applications together. On the other hand, when the program components involve more complex data structures and extensive algorithms, then the system programming languages are better suitable [15].

The dynamic languages are mostly open source [15]. Although there exist very comprehensive online communities for static programming languages as well, but those for dynamic languages exist from the very beginning and they provide an extensive support. The online availability of open source material and reusable code has helped a great deal in advancement of dynamic programming languages. This encourages beginners who mostly like to learn and make use of the language in a short period. The dynamic languages suit the most, their needs and this style of learning. Both kind of communities provide easy to use tutorials, help materials, and discussion forums. But additionally, one may find open source code for glue-able components to dynamic programming languages to enhance their functionality according to programmer’s desires [21].

### Table 1: Differences based on Use

<table>
<thead>
<tr>
<th>Feature Aspects</th>
<th>Dynamic Languages</th>
<th>Static Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer Productivity</td>
<td>More productive</td>
<td>Less productive</td>
</tr>
<tr>
<td>Portability</td>
<td>Usually interpreted so produce byte code and easily portable</td>
<td>Some of these languages are very good for portability but normally they are less portable</td>
</tr>
<tr>
<td>Use in applications</td>
<td>Normally used for gluing applications, plugging and extending components and GUI support.</td>
<td>Normally used for applications from scratch, component development and solution for complex data structures and algo’s.</td>
</tr>
<tr>
<td>Flexibility</td>
<td>More flexible</td>
<td>Less flexible</td>
</tr>
<tr>
<td>Code readability/simplicity/complexity</td>
<td>Being weakly typed they have more readability and simplicity</td>
<td>Strongly typed and have complex code</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Less efficient</td>
<td>More efficient</td>
</tr>
<tr>
<td>Reliability/safety</td>
<td>have less reliability and safety</td>
<td>More reliability and safety</td>
</tr>
</tbody>
</table>
Reusability | Highly reusability support | Comparatively less
---|---|---
Learning tool | Mostly they are easy to learn and some of them are used for education purpose e.g. Java | Comparatively difficult and have longer learning curve

2.2.2 Differences based on Features

According to [22] any definition falls under the category of dynamically scoped if its binding is looked up in the current call while the program is being executed. Dynamic scoping provides flexibility while the static scoping offers simplicity. The descendent of LISP i.e. Scheme makes use of static scoping while the other descendent of LISP i.e. Common LISP has support for both dynamic scoping as well as static scoping [16].

Encapsulation is the process that helps in hiding the details of internal implementation and restricts access rights of different modules towards other modules. On object level the ability of encapsulation is well understood with the concepts of public, private and protected access levels. These access levels determine the access rules for an instance to access member functions and variables. Static languages support encapsulation from their beginning. But Dynamic programming languages do not support this feature or have very limited support for encapsulation [23]. The actual implementation of modules in an object-oriented environment remains hidden from the outside environment. This ability is referred to as information hiding where control of the actual implementation remains limited to the particular class. This access control and information hiding feature are collectively known as encapsulation and it is necessary for safety and privacy purposes.

Several suggestions have been raised to enable encapsulation facility into dynamic programming languages. But study shows that there are some drawbacks and even after introducing some features, they were removed again [23]. Different rationales are discussed for the support of study rejection in [23]. Another discussion was made by [20]. Policies are suggested by [24] to enable the encapsulation facility in a better way in dynamic programming languages.

EVAL is introduced in LISP which basically means to evaluate expressions. Eval usually used to evaluate more than one argument or commands and then concatenate them together into a single command. Usage of EVAL is to pass code as run time arguments. But its not a good approach because of the threats involved in code of scripts from un-trusted sources. The better approach for the same is the use of higher order functions [25]. Higher order functions accept functions as their arguments instead of variables and in the same way they can return functions as a result [26].

The ability of a dynamic programming language is to inspect its own program, typically data, and types of objects and structure of the program at runtime. It is called introspection and it is also referred to as reflection [27]. This feature is mostly available in all dynamic programming languages. For example Smalltalk provides this feature to analyze the S-expressions [28] [17] [19].

Multiple inheritance is supported by most of the static languages for example C, C++. But among types of inheritance discussed in [30], the subtype inheritance is not supported by most of the dynamic languages. Hence, the dynamic programming languages mostly make use of multiple implementation inheritance rather than multiple subtype inheritance [29] [30]. Smalltalk and Python support both interface and implementation single inheritance. Python is a little more flexible as it also supports multiple inheritances. A special type of inheritance called Mixin is very popular in Dynamic programming languages e.g. in Ruby and Python. Mixin classes are special classes which have the sole purpose of adding properties to other classes. Mixin classes do not have instances. Additionally they don't have a super class and they can be placed anywhere in hierarchy. The only drawback is that the code becomes a bit complex with the use of mixins [30].
The closure and continuation are some other concepts which are explained in context of functional programming. But they are also sometimes referred to as a feature of dynamic programming languages, for example [17] [28].

Macros combine EVAL and introspection together and hence provides a powerful feature in some of the dynamic programming languages. In these dynamic languages macros can be used to change the grammar of language, to allow access to all features of interpreter, and virtual machine. It gives freedom to the programmer to make applications using macros to manipulate the interpreter, compiler and run-time states. Some of the static languages for example C and C++ also have 'macro' but its functionality is limited to string substitutions on the text of the program [28] [20].

<table>
<thead>
<tr>
<th>Feature Aspects</th>
<th>Dynamic Languages</th>
<th>Static Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation/Interpretation</td>
<td>Usually interpreted</td>
<td>Usually compiled</td>
</tr>
<tr>
<td>Powerful language</td>
<td>Built-in Powerful operations</td>
<td>Less built-in support</td>
</tr>
<tr>
<td>Automatic memory management</td>
<td>Mostly memory is managed automatic</td>
<td>Some languages support automatic memory management like java</td>
</tr>
<tr>
<td>Typing</td>
<td>Weakly typed</td>
<td>Strongly typed</td>
</tr>
<tr>
<td>Type checking</td>
<td>Run time</td>
<td>Compile time</td>
</tr>
<tr>
<td>Polymorphism/Dynamic Binding</td>
<td>Specialized and mature processes exist</td>
<td>Special strategies need to be implemented</td>
</tr>
<tr>
<td>Macros</td>
<td>Specialized powerful feature</td>
<td>Only used for string manipulation</td>
</tr>
<tr>
<td>Structural Conformance</td>
<td>Built in support available</td>
<td>Strategy needs to be implemented</td>
</tr>
</tbody>
</table>

2.2.3 Differences based on Dynamic Aspects

Dynamic aspects are also distinguishing differentiation between static and dynamic languages. The dynamic aspects described in Section 2.1 plays a vital role to draw a clear line between static and dynamic languages. Because all the issues described in above mentioned section are handled at runtime as static languages do not provide such facility to handle different issues at runtime. We think these dynamic aspects are clear and bold to make a separate entity in programming paradigm.
3. **SYSTEMATIC REVIEW**

3.1 **Systematic Review**

A systematic review or systematic literature review has become a well known method or technique to summarize the results within a period of study. A systematic review is a mean to identify, evaluate and interpret the results gathered for the particular research question, or topic. A systematic review is carried out in three steps namely planning, conducting and reporting the review [4].

3.1.1 **Planning the Systematic Review**

The reason for conducting a systematic review is to truly identify the work done in dynamic programming languages during the years 1985-2008. Furthermore, the review should also reveal the verification and validation aspects in dynamic languages. The knowledge gained from the systematic review about different developments and verification and validation will support to analyze the results of survey. The systematic review and survey for verification and validation (V&V) from industry will give support to find a gap and propose a model for verification and validation.

In order to be able to conduct the systematic review, a structured approach called ‘Review Protocol’ is required. A review protocol describes techniques and strategies regarding: searching, papers inclusion/exclusion, quality of paper, data extraction and synthesis. A review protocol is based on the guideline given by Kitchenham [4].

3.1.2 **Systematic Review Protocol**

The purpose of a systematic review is to investigate and compare existing work in order to derive creditable information useful for a trustworthy study result [4]. Kitchenham identifies three reasons for performing a systematic review [4]:

1. Provide a summary of existing evidence for a certain treatment.
2. Identify a gap in current research within the area.
3. Provide a framework/model for positioning new research activities in future.

The objective with this systematic review relates to the above mentioned reasons in the following way:

1. The review shall result in a summary of several classification schemes sorted out after gathering material regarding dynamic programming languages.
2. After collecting all existing developments in dynamic languages, it will be possible to determine how much research is done.
3. How much research is done in verification and validation?
4. The results gathered from systematic review, interviews and online survey will help to analyze the V&V practices in dynamic languages. The collected results will help us to propose a model for V&V in dynamic languages.

3.1.2.1 **Search Process/Strategy**

In this systematic review, the literature search process will be both ways, i.e. electronic and manual. The main focus shall be on dynamic programming languages. The articles from journals, conferences, proceedings and transactions will be starting from 1985 to date. The primary sources will contain the following:
IEEE and ACM are renowned databases for scientific and technical peer reviewed research papers. The Google scholar is getting wide with research options in many different disciplines, peer reviewed papers, books etc. Compendex, a comprehensive bibliographic database pertaining to scientific and technical engineering research in all engineering disciplines including millions of bibliographic citations and abstracts from thousands of engineering journals and conference proceedings. Inspec has bibliographic citations and indexed abstracts from publications in the engineering and scientific fields [1]. These two dialects give very fast and better results which have been covered from different resources. Compendex and Inspec works like a centralized database through which every researcher does not have to look individual databases. But as far as our concern for including more databases is to get more and relevant articles. It is possible to get less and irrelevant results against any search term, in such case same search term will be used for another database.

### 3.1.2.2 Search Terms

<table>
<thead>
<tr>
<th>ID</th>
<th>Term</th>
<th>Rationale</th>
<th>Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>IEEE</td>
</tr>
<tr>
<td>1</td>
<td>All: Dynamic AND All: Programming Language</td>
<td>First of all to get knowledge about the basic building blocks of dynamic languages.</td>
<td>X X X X X</td>
</tr>
<tr>
<td>2</td>
<td>All: Dynamic Aspects AND All: Scripting Languages</td>
<td>Due to the factor that dynamic languages are also known as scripting languages.</td>
<td>X X X X X</td>
</tr>
<tr>
<td>3</td>
<td>All: Programming Languages AND All: Strong Typing</td>
<td>Strong typing although found in static languages but it supports to cope the difference between static and dynamic.</td>
<td>X X X X X</td>
</tr>
<tr>
<td></td>
<td><strong>All: Programming Languages AND All: Weak Typing</strong></td>
<td>Weak typing is one of the dynamic aspects of dynamic languages, which supports to get the relevant material.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td><strong>All: Dynamically Typed AND All: Languages</strong></td>
<td>Another dynamic aspect of the dynamic languages which gives differences between static and dynamic languages.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><strong>All: Runtime Dynamics AND All: Programming Languages</strong></td>
<td>Dynamic languages are based on runtime modification; it will help to indirectly get the material.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><strong>All: Strong vs. Weak Typing</strong></td>
<td>Complements terms 3 and 4 to get the combined ideologies about both typing types.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td><strong>All: Dynamic Languages AND All: Verification</strong></td>
<td>To get knowledge about the role of dynamic languages in verification.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td><strong>All: Python OR All: Smalltalk OR All: Ruby AND All: Language Aspects</strong></td>
<td>Helps get details about different language aspects related to dynamic languages.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td><strong>All: Dynamic Behavior AND All: Programming Languages</strong></td>
<td>Compliments term 9.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td><strong>All: Python OR All: Smalltalk OR All: Ruby OR LISP AND All: Software Development</strong></td>
<td>Mainstream languages and their role in software development phases.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td><strong>((Smalltalk and (dynamic language)) OR (Smalltalk and (scripting language)) OR ((Smalltalk) and (testing) and (language)) OR ((Smalltalk) and (verification) and (language)) or ((Smalltalk) and (validation) and (language)) OR ((Smalltalk) and (Software Development Life Cycle)))</strong></td>
<td>Collective term to support the previously applied terms along with the focus on specially selected dynamic language</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td><strong>((Python and (dynamic language)) OR (Python and (scripting language)) OR ((Python) and (testing) and (language)) OR ((Python) and (verification) and (language)) or ((Python) and (validation) and (language)) OR ((Python) and (Software Development Life Cycle)))</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>-Do-</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Study Selection Criteria

#### 3.1.2.3.1 Inclusion Criteria

The articles published from 1985 to date on dynamic programming languages will be included in the study. The included articles will have the following topics:

- The papers which generally discuss about dynamic programming languages and different aspects like type binding, dynamic binding, code efficiency etc.
- Static or Dynamic typing/ Strong or Weak typing are well known differentiating factors between static and dynamic languages. The papers discussing these aspects would be included in the study.
- Another terminology which is mostly used for dynamic languages is “Scripting Language”, which also helps to cover the material regarding dynamic languages.
- The papers which are describing the Verification and/or Validation in general about dynamic languages or focusing on any of the selected dynamic language.
3.1.2.3.2 Exclusion Criteria

- The articles which do not meet the requirements defined in the inclusion criteria will be neglected. Otherwise if the topic is same but information is not relevant to answer the defined questions then articles will also be not included in the selection process.
- Studies where a particular dynamic language is discussed in relation to some technology not directly relevant to our research questions.

3.1.2.4 Selection Process

The articles found according to the inclusion criteria will be assessed by two researchers. In case of any ambiguity or conflict, the researchers will solve the conflict through discussion and inclusion criteria. The requirements or tollgates according to which articles will be selected or rejected are described as follow. These tollgates or requirements will be helpful to pilot the inclusion and exclusion criteria’s.

- **T1-Title**
  - The title contains the term dynamic programming language or has some idea regarding the dynamic aspects like dynamic binding etc.
  - The title reveals the topics described in the inclusion criteria e.g. scripting languages, dynamic typing etc.
  - The title reveals about the verification and validation in the dynamic programming languages.

- **T2-Abstract**
  - The abstract findings will be helpful to gain better understanding and to have some clearer picture regarding the questions and topics.

- **T3-Conclusion**
  - The concluded results have a chance to use it for further suggestions regarding the dynamic languages.

- **T4-Whole Article**
  - The contents of the paper as a whole if found related to our study, then it will be selected.

3.1.2.5 Study Quality Assessment

There exist different quality assessment guidelines given in different systematic review frameworks. Kitchenham *et al.*[4] provided a comparative study on such frameworks devised by Systematic Reviews Group and some others. Based on the area of research and study structure, CRD provides better guidelines. Mainly the CRD suggests criteria to evaluate the quality of systematic reviews. We shall use the guidelines based on CRD [4] after modifying them according to our study requirement.

The study quality will be accessed according to the following guidelines [4].

- Does the research article follow the inclusion and exclusion criteria?
- Does the research article address the biasness issues by including views of other relevant approaches?
- Were the issues of internal and external validity threats adequately described?
- Do the results help to provide a better understanding for dynamic languages and verification and validation in dynamic languages?
The study is a systematic review; therefore the biasness issue will be implicitly minimized, as all the state of the art will be researched.

3.1.2.6 Data Extraction Strategy

The data extracted from each paper will consist of following:

- The source (the conference or journal)
- The year of publication
- Type of study (research, development, usage, other)
- Main study area (like Dynamic Languages: i.e. Python, Lisp, Smalltalk, Erlang, Self and Ruby, Verification and Validation)
- The author(s)
- Main Theme of the article (used for the classification in developments)

The data will be extracted by each researcher and sample from extracted data by one researcher will be assessed for inter-researcher consistency, by the other researcher and vice versa.

3.1.2.7 Synthesis of Extracted Data

The Systematic review is a qualitative study; therefore the results obtained are expected to be of heterogeneous nature. Contradicting claims are likely to emerge in different studies. There can be two possibilities: either the claims will be given with valid experimental evidence, or they will merely be claims based on different qualitative studies. In first case, the quality assessment criteria will be followed and after discussion between researchers, both contradicting claims will be included. In the second case, the inclusion/exclusion criteria will be followed again and the screening process will be applied more rigorously. In case of any remaining discrepancy, both the researchers will decide after discussion.

3.1.3 Conducting the Systematic Review

The authors used a search log to maintain the documentation of search results. The log is beneficial to keep record of searched articles and to avoid duplication. It also minimizes time to maintain and locate searched material. The search log is maintained using Microsoft Excel and Microsoft Word files and it contains the information regarding article title, main category and a sub category according to field of research. Brief information about the theme of the selected article and the reason of the excluded article is also mentioned in the search log. Additionally it includes the search terms used, articles found against each term, duplications and the remaining articles after initial screening are reported.

3.1.3.1 Screening Process

This phase emphasizes on excluding the irrelevant research material. In other words, those articles which go beyond our scope are screened out.

The papers will be selected based on the inclusion criteria defined earlier in section 3.2.1. For example:

- The abstract explicitly mentions the verification or validation in the context of dynamic programming languages.
The articles will be excluded according to the exclusion criteria defined earlier in sections 3.2.2. For example:

- The abstract does not clearly describe any of the areas included in our scope, as defined in section 3.2.1

This phase of our study investigates the validity of the statistics given in section 3.1.4, that were obtained after applying the inclusion-exclusion criteria for papers selection.

### 3.1.3.2 Examples of Selection Procedure

This section presents some examples of how papers were excluded with help of the inclusion-exclusion criteria.

#### 3.1.3.2.1 Title Rejection

Some papers were rejected due to their title. Examples of rejected papers are:

- “*OWL-Full Reasoning from an Object Oriented Perspective*”, Koide. S, Takeda. H, 2006 – was rejected because article title is out of scope.

#### 3.1.3.2.2 Abstract Rejection

The title of the article was not clearly focusing our study scope; therefore the abstract was studied to get clearer picture. The article passed title criterion but failed abstract criterion. Following is an example:

  - **Title Accepted** – the title gives usage of the python in implementation and hence it passed the title criterion.
  - **Abstract Rejected** – after reading the abstract it becomes clear that the *monte python* is mathematical term and hence it was out of scope which leads to exclusion.

#### 3.1.3.2.3 Conclusion Rejection

Articles which could not be excluded on the basis of title and abstract are looked further for their conclusion. In such cases the article passed the title and abstract but could not pass conclusion criteria. Following is an example:

  - **Conclusion Rejected** – the title and abstract somewhat seemed relevant but confusing. To remove the confusion, the conclusion was read, which resulted in describing the simulation used on some prolog dialects to measure the performance. The rationale for exclusion is that Prolog is not included in our languages list.

#### 3.1.3.2.4 Whole Article

If it did not clearly fall into accepted or rejected articles from title, abstract and conclusion, then the whole article body had to be studied. After reading the whole paper an overall view could be obtained which, in some cases, was not suitable with the view perceived from the title, abstract and conclusion. For example:

  - **Whole Rejected** – the title and abstract were somewhat pointing towards
the dynamic issues. But after reading the whole article it was clear that article focus was on ‘nu-calculus’ which lead it to exclusion.

3.1.3.3 Systematic Review Statistics

Both authors collectively searched articles on dynamic programming languages and dynamic aspects in the initial literature overview. In the next phase six dynamic languages were selected and both authors searched the research material for three languages each. The language set is combination of old and new dynamic languages. The rationale of having old and new languages is to have understanding of research progressed in their eras. The articles included are screened out according to the inclusion and exclusion criteria in 3.2.1 and 3.2.2 respectively. In case of any confusion during screening process, the issue is resolved after discussion. Initially 1850 articles were selected for research and after screening process we had 325 articles to carry out further research. These articles were selected after the removal of duplication and irrelevant articles. The tollgate is applied again on these 325 articles to further purify the research. After applying tollgates the no of articles reduced to 174.

3.1.3.4 Classification Scheme

The classification scheme helps us to differentiate between research materials. The research has been classified according to articles obtained after screening. There are two scheme’s found i.e. development and usage in various fields. Therefore we have made two main categories of our researched material. After studying the current trends in dynamic languages and the sample of six selected languages, the main classes are further divided into smaller subclasses.

- **Development**
  There are six subclasses for development category and four subclasses for usage category. The subclasses of development are: language (i) constructs/dynamic aspects, (ii) comparison, (iii) enhancement, (iv) integration, (v) testing, (vi) verification and validation, (vii) performance, definitions of dynamic language, issues/challenges in dynamic language.

- **Usage**
  The four subclasses that lie in usage category are: support, framework, architecture, and general. These are the identified categories for usage but all the sub-classes are reported under usage.

Once the classification is complete, we shall elaborate the development subclasses further in detail along with a timeline of relevant researched work. Next step will be to explore in detail the subclass i.e. verification and validation/testing. Brief description of the subclasses of usage category shall be included and purpose of research work will be included into this category.

3.1.4 Reporting the Systematic Review

The classification scheme defined in section 3.1.3.4 will be explored in detail in next chapter, which will focus on the two main classifications i.e. development and usage.
4. SYSTEMATIC REVIEW RESULTS

The systematic review results reported in the next sections are according to the classification of research i.e. development and usage in section 3.1.3.4. The section 4.1 and 4.2 explain definitions and issues revealed from the differences and dynamic aspects. The sections from 4.3 to onward explains the development and usage in languages selected for systematic review.

4.1 Definitions/ General idea about Dynamic Languages

According to [13], the author mentioned dynamic languages as: “languages those basically enables programmers to change their code, logical structures, adding variable types, module names, classes, and functions at runtime. These languages frequently are interpreted and generally check typing at runtime”. Some of other authors have also supported this definition in some other way for example [12] [9] [31] [32] [33] [17]. These thoughts clearly and precisely explained the general idea about dynamic languages.

Some other arguments are found to entitle the dynamic languages as dynamically typed and also scripting languages. The rationale for being scripting language is that some of the dynamic languages are also lie in the category of scripting languages like perl or ruby [13] [12] [9].

4.1.1 Analysis

The systematic review results have shown that this concept is widely used and hence establish state of the art idea about dynamic languages. The idea depicts the general understanding of the dynamic languages, which gives a solid mind set to grasp the phenomena behind dynamic languages.

4.2 Issues concerning Dynamic Languages

Systematic review has outlined some issues concerning dynamic languages with both positive and negative criticism. The author has mentioned that dynamic languages are more compact in writing programs as compared with static languages. The reason for being compact is absence of type declaration overhead which in return reduces the code size and let the developers to write exact code as he/she requires [13][17]. Trats and Wuyts [12] also supported this claim about dynamic languages.

According to Carlson [9], dynamic languages are well equipped to glue the different components in software application. Dynamic languages provide support for faster and rapid application (software) development. It also facilitates to reuse and glue the components easily [33][9][13]. Another claim supported by author in [13] and Trats et.al. [12] that dynamic languages are being used by industry giants like Microsoft and Google. Both are using dynamic languages for some reasons which are explained in [13][12][32]:

- “First, current static type systems in mainstream object-oriented languages have little expressive power”. For example, although they restrict the users to add or assign the desired value like adding a string to a bool. In fact, they argue that static type systems can’t facilitate to detect most common programming errors.
- “Static type systems tend to ossify programs, making them less flexible, more difficult to work with, and less amenable to change. When a user wants to modify one aspect of a system, that system’s types could force numerous rewrites in unrelated parts of the system”.
There is also a disagreement that dynamic languages are slow in nature, unreliable and can be used for small applications but they have been used by IT giants which in fact does not support this claim \[9\] \[12\] \[13\] \[32\]. Some other aspects like low performance and high memory usage are also analyzed. Different authors are supporting this claim by declaring dynamic languages with low performance and high memory usage due to dynamic typing, for example \[12\] \[13\] \[32\].

4.2.1 Analysis

Different claims have been analyzed during systematic review both consisting of positive and negative criticism. To perform the analysis related to issues of dynamic languages, we categorized it according to concerning issues and level of criticism. The level of criticism is either positive or negative. Most of the issues are in favour of dynamic languages which can raise the popularity graph among the followers of static languages. Some of them think that dynamic languages have low performance and high memory usage. But with the advancement in technology and high processing machines, these claims could not make their way to lower down the graph of dynamic languages.

Table 4: Analysis results according to Issue and Level of Criticism

<table>
<thead>
<tr>
<th>SR No</th>
<th>Concerning Issue</th>
<th>Criticism Level</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flexibility</td>
<td>Positive</td>
<td>[13][17][12]</td>
</tr>
<tr>
<td>2</td>
<td>Gluing components and Rapid application</td>
<td>Positive</td>
<td>[9][33][13]</td>
</tr>
<tr>
<td></td>
<td>development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Programmers productivity</td>
<td>Positive</td>
<td>[13]</td>
</tr>
<tr>
<td>4</td>
<td>Usage</td>
<td>Positive</td>
<td>[13][12][32]</td>
</tr>
<tr>
<td>5</td>
<td>Performance and reliability</td>
<td>Negative</td>
<td>[9][12][13][32]</td>
</tr>
<tr>
<td>6</td>
<td>Small Applications development</td>
<td>It’s not a criticism as positive or negative but it is just an argument.</td>
<td>[9][12][13][32]</td>
</tr>
</tbody>
</table>

4.3 Lisp Developments

The development scheme is further drilled down to sub-schemes which have insight according to the relevant developments in Lisp.

4.3.1 Comparison

The Author describes the comparison between ‘Lisp’ and ‘C’ which consist of different aspects like source code size, object size and some qualitative aspects like type checking, primitives (built in functions), memory management, testing and debugging etc. Experiment is used to know these differences which ends in concluding that type checking is versatile feature of dynamic languages i.e. Lisp and there seems no difference in development of large programs[34]. Hodgkinson [35], also focuses on comparison between the same enemies to each other. The author give a balance percept of both languages by focusing on aspects like memory management, garbage collection etc.

An experiment was performed by Erran [36] to compare the performance, development time and memory consumption etc of Lisp with its counterparts i.e. Java and C++. The author justify the arguments by using the raw data of previous study from Prechelt by performing an experiment. It is concluded that Lisp has everything which makes it a competitor of Java in terms of performance, development time, automatic memory management etc. It also
superior to Java and comparable to C++ in runtime issues and also has better quality in programming effort and results variability.

4.3.2 Dynamic Aspects

Dynamic aspects are very important traits or features of dynamic languages. Moreau [14] describes the syntactic theory of dynamic binding which is useful for performing equational reasoning on programs. This theory will aid to derive the sequential evaluation function in the perspective of context-rewriting system. The evaluation function is refined by two implementation strategies i.e. deep and shallow binding. It is also claimed that deep binding is suitable for parallel evaluation and.

Costanza [37] focused on the dynamic scoping which is used in dynamic languages. The author proposed some changes in dynamic scoping which will be used for aspect oriented methodology. The change in dynamic scoping will give some value to previous attempts and add some language constructs in functions at runtime in aspect oriented programming.

Neubauer and Sperber [38], presented the dynamic scoping analysis which is actually on the Lisp subset called Emacs Lisp. The acceptability relation for formulation and implementation is used to track the flow of bindings along with flow of values. Primarily this study focuses on the lisp programs translation into modern languages with support of lexical scoping. The previous study presented by Moreau [14] is prerequisite of this study.

Dynamic typing is another main trait of dynamic languages, Steenkiste and Hennessy [39] implemented dynamic typing strategy in Lisp through ‘Tags’. Tags are used for type checking at runtime in Lisp and normally tags implementation consist of four operations i.e. tag insertion, tag removal, tag extraction and tag checking. The authors optimized run-time checking for performance comparison with Lisp.

4.3.3 Framework

Brinkley and Prother [40], has presented a framework which supplement their research. The framework is a hybrid Lisp-C toolkit i.e. ‘SLisp’ for the distributed and scripting development. It is actually based on Xlisp which used in the development of visualization program. ‘SLisp’ generalized the hybrid approach which is used for Xlisp applications which supplements programmers to extend Xlisp with C-coded functionality according to Common Lisp Standard. The ‘SLisp’ toolkit allows creating C modules by using primal Lisp functions. The C-coded modules will later help in composition with Lisp for the development of applications ranging from small to large and Lisp also used as gluing for large C-modules.

4.3.4 Language Development/Enhancement

There are some developments found in Lisp and its dialects. The Common Lisp, a dialect which have same features as Lisp. This paper [41] focuses on the advancement presented to support logic programming in Common Lisp framework. These constructs has support for locally defined and first class relations and logical variables as first class Lisp objects. Two new types have been presented i.e. logical variable and relations. Sevinc [42] also presented an extension of common lisp to support discrete event modeling and simulation. The author’s simulation environment is model oriented which support behavioral and structured simulation. The structural changes are handled through first order statements and also involve in creation, integration and destruction of models. Tung [43] designed a module system and a programming environment to support interactive programming in Lisp dialect i.e. ‘Scheme’. The module system focuses on the extension of lexical scoping and programming environment has support for dynamic linking, separate compilation, production code compilation and a user interface with multiple read eval print contexts.
Wright and Cartwright [44] have presented a Soft type system for Lisp dialect i.e. ‘Scheme’. The authors described that “soft type system infers types for the procedures and data structures of dynamically typed programs. Like conventional type system, soft types express program invariants and thereby provide valuable information for debugging and optimization.” The authors has presented practical soft system and its based on the Hindley-Milner type system. This system is capable of being expressive, solid, efficient for computation and supportive for new type definitions.

Another study is carried out by Henglein [45] related to type system to present the optimization of tagging operations in Lisp by using type inference. This paper presents dynamic type inference which provides a well-acceptable, independent (from implementation) and well organized way to eliminate tag handling operations in dynamic languages. It will help to create new horizon’s for the construction of compilers and reduces the gap between statically typed and dynamically (run time) typed languages.

### 4.3.5 Performance Issues

Being interpretative many thoughts have made a blur sketch about the performance issues among the pool of different professionals. Many thoughts have raised the performance issues like Pumplin [46] declared it being slow due to its interpretative nature and it also takes time to interpret the whole program. It influences the slow processing and speed of program and the author also focuses on the specification of the system which makes it slow. But with the aid of advancement in technology, this issue can be neglected and will not affect more.

Larus and Hilfinger [47] presented an algorithm to improve the performance of concurrent programs in Lisp. The author’s focused is to reduce the conflict among pointers, recursive functions instead of loops, to maintain sequence and serialization among conflicting statements and library transformations to improve overall performance of Lisp programs. These techniques are helpful because Lisp like languages have a different structure and have to identify more things at runtime as compared to traditional languages like C, Fortran etc. These techniques will be helpful for those languages which belong to the same family as Lisp.

Vogt [48] has explored the performance issues of floating points in Lisp dialect i.e. Common Lisp and favoured dynamic languages to be good in processing of floating point operations as compared to their counterparts static languages. The author described that “When a programming task entails a significant amount of floating point (FP) operations, the FP processing speed becomes the primary consideration in selecting a language to be used for implementation”[48]. So with a little more analysis the dynamic languages provide the same facility proficiently.

### 4.3.6 Usage

There is also a set of papers which describes the usage of Lisp in different systems development and architectures. The more focused papers are described with their purpose and their technique in table 5.

**Table 5: Usage publications**

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Purpose</th>
<th>Technique/Idea Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This paper focuses on architecture advancements with respect to adding new tag representations as well as separation of register execution and ALU operation timing constraint.</td>
<td>Branch control mechanism is used to handle the control transfer.</td>
<td>[49]</td>
</tr>
<tr>
<td>2</td>
<td>A simulation based system is developed to support the SDL (Specification Description Language) processes to detect logical and temporal problems.</td>
<td>Lisp dialect i.e. Common Lisp is used to implement the system. CLOS and multiprocessing techniques are used.</td>
<td>[50]</td>
</tr>
<tr>
<td>3</td>
<td>A hyper media implementation is described with the aid of dynamic object methodology.</td>
<td>Dynamic object methodology with Common Lisp is used.</td>
<td>[51]</td>
</tr>
</tbody>
</table>
4 For the communication of agents in real world and for enhancing coordination and cooperation, a multi agent system RESTINA is developed. It is also made possible that agents can also communicate through Agent Communication Language. For this purpose the system is developed using Lisp and some other languages like Java, C etc. [52]

5 Object oriented systems are very common in software engineering. A new object oriented system is developed to give a non-centralized knowledge base system to users. “The terminology ‘KNO’ is used which refers to knowledge acquisition, dissemination and for manipulation of objects”. Lisp is used for the implementation of system. [53]

6 Another knowledge based system almost similar to [53] is introduced to give insight of dynamic behaviors of complex systems and also give support for management of dynamic processes. The system uses the quantitative simulation with symbolic techniques and the Lisp is used as a part in implementation of system. [54]

4.3.7 Analysis

With the passage of time there are advancements being observed in different schemes made according to the trend of papers. Among these advancements there are some areas where enough research has been done like language enhancement, dynamic aspects and performance. The usage scheme can be further classified according to the different areas of application which are

- Artificial Intelligence
- Knowledge Base
- Simulation
- Hyper Media
- Architecture

Table 6 shows the major research done according to classification scheme and table 7 lists the areas which are taken care in research.

**Table 6: Major Research Classification Scheme’s**

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Classification</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dynamic Aspects</td>
<td>Dynamic binding[14], scoping [37][38] and typing [39]</td>
</tr>
<tr>
<td>2</td>
<td>Enhancement</td>
<td>Simulation [42], Logic programming [41], Environment for Interactive programming [43], Soft typing and Tagging in Type system [44][45]</td>
</tr>
<tr>
<td>3</td>
<td>Performance Issues</td>
<td>Floating point [48], Concurrent processing [47] and Processing speed [46]</td>
</tr>
</tbody>
</table>

**Table 7: Usage Classification according to Application area**

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Area</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Artificial Intelligence</td>
<td>[52][54]</td>
</tr>
<tr>
<td>2</td>
<td>Knowledge Base</td>
<td>[53][54]</td>
</tr>
<tr>
<td>3</td>
<td>Simulation</td>
<td>[50]</td>
</tr>
<tr>
<td>4</td>
<td>Hyper Media</td>
<td>[51]</td>
</tr>
<tr>
<td>5</td>
<td>Architecture</td>
<td>[49]</td>
</tr>
</tbody>
</table>
The figure 2 shows the results of publications in different schemes. These results are based on the selected articles for Lisp i.e. 25.

![Graphical Representation of Results](image)

### Figure 2: Graphical Representation of Results

#### 4.4 Erlang Developments

##### 4.4.1 Comparison

Phil [55] writes comparatively about two C++/CORBA telecom applications reengineered in Erlang. The analysis of two components DCC and DM shows empirical studies concluding Erlang to be more robust and resilient language. It provides distributed products development with higher availability, dynamic configurability, better productivity and maintainability. The size of Erlang code for the development of these components is three times less than code of C++. Erlang resulted in better performance in terms of time, space, productivity and automatic memory management.

##### 4.4.2 Dynamic Aspects

An interesting discussion about implicit type inference in dynamic languages is carried out in [56]. The authors propose the notion of success typing and apply algorithms on Erlang to show that a significant part of type information can be reconstructed along with automatically annotating function interfaces and definite type clashes detection. There however, exist some issues for example the recursive types other than lists were not handled automatically. Moreover type dependencies between domain and range of functions still needed to be worked out.

Claes in [57] gives an overview of Erlang’s dynamic interfaces that helped in development of fault tolerant software. Availability of the high speed, cheap off-the-shelf CPUs is one of the influencing factors for the development of fault tolerant software instead of fault tolerant hardware. Joe Armstrong [58] discussed different issues related to Erlang and pointed out weak areas where Erlang could be improved for instance garbage collection of atoms [58].

Another work related to runtime garbage collection in Erlang is proposed and implicated in industry [59]. Jay in [60] elaborates Erlang as a COPL (Concurrency Oriented Programming Language) and also highlights features of COPLs that enable them the preferable choice over normal structured programming. For instance, strong encapsulation, dynamic interfaces, fault isolation and automated testing are some of the aspects discussed in [60]. The author recommends the fundamental design element as process to obtain these characteristics along with concurrency.
4.4.3 Integration

Francesco et al [61] gave the idea of a web framework that could eliminate the problems in combining multiple programming paradigms. The Web Platform is developed in Erlang with having the ability to run components from other languages, databases and web applications within same platform and same memory space. This generic framework is intended to help in development for gluing logic and back-end web based applications. The Web Platform will basically reduces time and effort on developing similar design patterns.

4.4.4 Language Development/Enhancement

The development of Erlang started in 1980 at Ericsson [61]. Initially Erlang's code was based on Prolog. But with the passage of time, the code matured and efficiency increased. By 1990 Erlang had its own code. Claes Wikstrom enhanced Erlang language by adding distribution to Erlang. For instance, the Erlang OTP (Open Telecom Platform) libraries were organized which lead to stable software base for Erlang programming in telecommunication [62].

Later on the language enhancement continued and by 2005, Erlang became a standard language for concurrent development [58]. In 2006 Erlang for multi-core CPUs became hot issue as in August 2006 Erlang was released for SMP (Symmetric Multiprocessing) by OTP Group [58]. Hans and Lars [63] investigated pitfalls of programs using semantics for distributed Erlang and a model checker named as McErlang. They identified two problems: (1) reuse of Pids (process identifier) and (2) silently dropped messages during node disconnect/connect. Generally, the reuse of Pids does not have problematic impact on applications, but in case of highly critical systems where unique Pids need to be used; this issue must have to be considered beforehand. They suggest enhancement in API of Erlang to handle these issues. Another suggestion is the use of guided node supervisors to tackle the second issue. In [64] Hans and Lars provide a considerable improvement of their earlier work [63] done related to Fredlund’s single-node semantics. The extensions included restructuring of parts of distributed layer and corrected many errors related to node disconnection.

Vincenzo [65] identifies a gap for the development of hard-real-time applications, where imperative languages are still preferred. He proposed a working prototype of a hard real-time scheduler developed in Erlang i.e. HARTE. He also provided performance analysis for HARTE with promising results. Miguel et al [66] has proposed an extension to Erlang language providing the facility to specify contracts with type information at individual function level. A generic server for the priority based message reception is introduced in [67].The generic FSM and prioritized FSM were used in combination and the result of the study were demonstrated by using the Erlang language. Based on the new research an Erlang extension Proposal was also suggested.

4.4.5 Performance Issues

Erik and Sven-Olof [68] have worked on inter-process optimization of Erlang by suggesting a profile driven compiler optimization technique. The authors used prototype to validate their proposed method and found promising results. Erik et al [69] also developed a compiler for the programs written in Erlang. The existing Erlang compilers on SPARC machines are known as BEAM. HiPE (High Performance Erlang) as compiler is another effort to increase the performance of Erlang. HiPE has become an integral part of Open Source Erlang distribution[58].

Erik and Konstantinos [70] have exposed the reasons of extra time taken by HiPE (High Performance native code compiler for Erlang) during compilation of Erlang programs. Their experiments on register allocation mainly focused on linear scan and performance analysis is done on IA-32 using different options. The study results provides empirical evidences along with theoretical results advocating the linear scan to be the best choice for register rich environments i.e. SPARC.

According to [71] Scheme programs are faster than their counterparts in Erlang. To enhance Erlang's performance, they developed Erlang to Scheme compiler (EtoS). EtoS
compile Erlang code into Scheme code. The performance of EtoS was found to be better when compared to existing Erlang compilers like HiPE native code compiler, BEAM/C and JAM. The results produced by using EtoS coupled with Gambit-C were significantly faster than others. Only the list processing and process management related programs have less performance. It is claimed in [71] that Gambit-C part of EtoS compiler could be worked out more to remove this drawback.

4.4.6 Usage

The following table represents the different publications available in terms of usage.

**Table 8: Usage Publications**

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Purpose</th>
<th>Technique/Idea description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To address the problem of process network verification where interconnection structure keeps changing.</td>
<td>The specification logic and proof system used here, is making use of Erlang specification primitives with first order μ-calculus.</td>
<td>[72]</td>
</tr>
<tr>
<td>2</td>
<td>Use in manual and automatic reasoning (model checking).</td>
<td>The Erlang language has features that support in developing automated reasoning and model checking tools. Erlang being a representative of concurrent languages is a good tool to experiment and manipulate it using CCS for Erlang Programming language</td>
<td>[73]</td>
</tr>
<tr>
<td>3</td>
<td>McErlang, a tool for model checking.</td>
<td>This tool is written in Erlang and is specialized for model checking of distributed functional programming language i.e. Erlang</td>
<td>[74]</td>
</tr>
<tr>
<td>4</td>
<td>Erlang was used to implement a new network communication design because the support for processes in Erlang allows runtime configurability and parallelization for efficient resource usage.</td>
<td>Idea of concurrent caching is manipulated by Jay [6] and a design was devised for spreading the ached data across a cluster of computers.</td>
<td>[75]</td>
</tr>
<tr>
<td>5</td>
<td>Erlang-based framework intended to be used for development of autonomous mobile robots softwares.</td>
<td>The authors analyzed the problem in existing implementations for robots and derived an infrastructure of their own, based on layers and modules.</td>
<td>[76]</td>
</tr>
<tr>
<td>6</td>
<td>To provide support in telecommunication development through built in language features.</td>
<td>Having special set of libraries, tools, debugging support, foreign language interfaces, and design principles for Open Telecom Platform (OTP), Erlang has become a de facto standard for softwares in telecommunication.</td>
<td>[77]</td>
</tr>
<tr>
<td>7</td>
<td>A framework for education purposes</td>
<td>Kara with Erlang</td>
<td>[78]</td>
</tr>
</tbody>
</table>

4.4.7 Analysis

We have distributed research results into different categories and the visibility of our classification scheme has become clearer. Our analysis shows that the research in Erlang is mainly focused on following areas:

- Dynamic Aspects
- Language Enhancement
- Usage
  - Communication
    - Telecommunication
Network Communication
  o Robotics
  o Hardware Support
  o Verification and Model Checking tools
  o Education

Following tables list the main focus of past research in classification scheme and different areas.

Table 9: Major Research Classification Scheme’s

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Classification</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dynamic Aspects</td>
<td>Type system [56], Dynamic interfaces [57], Garbage collection [58] [59], Encapsulation, Dynamic Interfaces [60]</td>
</tr>
<tr>
<td>2</td>
<td>Language Enhancement</td>
<td>Development History [61], Open Telecom [62], concurrent development [58], Node Communication (API Enhancement) [63] [64], HARTE (Scheduler) [65], Contracts [66]</td>
</tr>
</tbody>
</table>

Table 10: Usage Classification according to Application area

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Area</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Telecommunication</td>
<td>[72] [77]</td>
</tr>
<tr>
<td>2</td>
<td>Network Communication</td>
<td>[75]</td>
</tr>
<tr>
<td>3</td>
<td>Robotics</td>
<td>[76]</td>
</tr>
<tr>
<td>4</td>
<td>Hardware Support</td>
<td>[58]</td>
</tr>
<tr>
<td>5</td>
<td>Verification and Model Checking Tools</td>
<td>[73] [74]</td>
</tr>
<tr>
<td>6</td>
<td>Education</td>
<td>[78]</td>
</tr>
</tbody>
</table>

The figure 3 shows the results of publications selected for reporting i.e 45.

4.5 Ruby Developments

Ruby being the new member in dynamic languages is getting a handsome feedback from the professionals. There have some research found in ruby which is explained in coming sections.
4.5.1 Integration

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To check the applicability of dynamic languages, Ruby is extended to support middleware projects integration. Developers can create applications developed in dynamic languages which use preexisting services. These services are originally developed in middleware languages. A dynamic language uses separate address for applications which can be accessible through avenues [93].</td>
</tr>
<tr>
<td>2</td>
<td>Vinoski [94] has supported the Schimldt[95] work to integrate enterprise applications with Ruby. The idea described for the integration seems to be a practical thought. He argued that systems with multiple techniques and technologies like databases, XML, messaging could be glued with the use of Ruby [94].</td>
</tr>
</tbody>
</table>

4.5.2 Comparison

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ruby on Rails (ROR) is a framework which is based on Ruby. Ebert discusses a comparative study of ROR with already developed frameworks such as apache struts,tapestry, cocoon and .net framework for web applications [96].</td>
</tr>
<tr>
<td>2</td>
<td>The author has given a brief insight about code readability and standards with respect to static and dynamic languages. The well structured code itself provides a good sense of documentation otherwise purpose of code does not fulfill [97].</td>
</tr>
</tbody>
</table>

4.5.3 Enhancement/Development

<table>
<thead>
<tr>
<th>Ruby</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To improve parallel programming, a merger of MPI and Ruby is introduced called MPI Ruby. It also include Remote Memory Access (RMA) to facilitate one side communication [98]. Furthermore MPI ruby also supports for parallel environments [99].</td>
</tr>
</tbody>
</table>

4.5.4 Usage

**Table 11: Usage**

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This study provides a bridge for professionals to understand the design issues of aspect-oriented software systems. These aspect oriented softwares are developed in Java and Ruby. It is also figure out design problems related to aspect oriented and extensions to object oriented principles to support aspect oriented designing [100].</td>
</tr>
<tr>
<td>2</td>
<td>Dynamic languages flexibility and rapid developments have gripped their way to develop model driven applications. The author describes their experiences with development of model driven tool based on domain specific languages and tool is developed in Ruby [101].</td>
</tr>
<tr>
<td>3</td>
<td>T-Ruby is a designing system to support and simulate designing of VLSI circuits [102]. Furthermore, another study is conducted to support VLSI through Ruby [103].</td>
</tr>
<tr>
<td>4</td>
<td>A tool to support visualization of data (NIFScope) is developed with Java and Ruby. Ruby is used to process the text file which are obtained after analysing the data [104]. Ruby-Helix is introduced to support image processing and analysis of helical objects in [105].</td>
</tr>
<tr>
<td>5</td>
<td>For hardware design, a study is conducted which is implemented with the help of Ruby. The study is based to support transformation of calculations from software to hardware [106].</td>
</tr>
<tr>
<td>6</td>
<td>Some of the studies are also found for web services [107], education [108], bioinformatics [109] and ruby tutorial [110].</td>
</tr>
</tbody>
</table>
4.5.5 Self Development (Performance)

Table 12: Self Development (Performance)

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>There have some advancement for self like Ungar et al [111] have improved the performance by introducing maps, dynamic customized compilation, type prediction and message in-lining. Another study carried out by Ungar et al [111] [112] to improve performance by message splitting and iterative type analysis in Self. Both of these concepts have reduced the penalty level in terms of performance in dynamically typed languages. Furthermore Unger et al [113] also involved in another study to improve the performance of Self by introducing optimization techniques to recapture the efficiency of object oriented languages by message splitting.</td>
</tr>
</tbody>
</table>

4.6 Python Developments

The following subsections elaborate the development scheme regarding Python.

4.6.1 Comparison

Table 13: Comparison

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Python's similarities and contrasts with C++ are discussed in [114]. Python shell implementation for ORBIT (code for simulation) outcasted old SuperCode shell written in C++ like language. Python helped to develop PyRobot and proved to be more flexible and easier to extend.</td>
</tr>
<tr>
<td>2</td>
<td>The relative effectiveness of C, C++, Java, Perl, Python, Rexx, and Tcl is determined in a study and dynamic programming languages resulted in a comparably equivalent or better results.</td>
</tr>
<tr>
<td>3</td>
<td>Perl, Python and Java are analyzed for the ease of use and performance of socket communication in distributed systems. The results indicate that productivity is enhanced when developing distributed applications with Python. On the other hand, its performance measure is not the best, although it is acceptable.</td>
</tr>
<tr>
<td>4</td>
<td>At the department of CS at The George Washington University, a course of comparative programming is introduced. A number of different features of C++, Java, Perl and Python are included in the study including typing and memory management.</td>
</tr>
</tbody>
</table>

4.6.2 Dynamic Aspects

Table 14: Dynamic Aspects

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Python is selected for the automation of laboratory system environment. The reasons of selection are the dynamic typing and type less declaration types in Python.</td>
</tr>
<tr>
<td>2</td>
<td>Dynamic typing and methods to infer the types are discussed for Python in specific and dynamic languages in general.</td>
</tr>
<tr>
<td>3</td>
<td>Python is discussed in a broad spectrum of dynamic languages where dynamic typing is claimed to be the best feature of these languages.</td>
</tr>
</tbody>
</table>
Dynamic binding, mix-ins and Python’s ability to change its class structure dynamically are discussed in [120].

### 4.6.3 Framework

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MPI for Python is a tool to support bindings of Message Passing Interface (MPI) standard. It also enables interaction of python programs with multiple processors. [121]</td>
</tr>
<tr>
<td>2</td>
<td>SPyRO stands for Simple Python Remote Objects and it manages the transparent access of simple Python remote objects. [122]</td>
</tr>
<tr>
<td>3</td>
<td>Python Robotics (Pyro) is a Python based robotics development environment which provides portability to a number of robotics platforms. [123]</td>
</tr>
</tbody>
</table>

### 4.6.4 Language Development/Enhancement

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Python has 'Generators’ which help in developing automated systems. (a good discussion on different developmental aspects and tools) [118]</td>
</tr>
<tr>
<td>3</td>
<td>It is claimed that the statically typed programming languages are less suitable in the analysis and design phase of a prototyping-oriented software life cycle. Python being interpreted is claimed to be better suited for such developments. An enhancement in Python to enable the support for DBC is presented and a DBC model for Python is proposed. It is claimed that the similar approach can be used for all Dynamic languages. [119]</td>
</tr>
<tr>
<td>4</td>
<td>An extension to Python, a library has been created using C for the management of XML files. [124]</td>
</tr>
</tbody>
</table>

### 4.6.5 Integration Support

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An integration tool that enables automatic documentation and integration of C/C++ code with scripting languages code. Mainly it supports Python. [126]</td>
</tr>
<tr>
<td>2</td>
<td>Python is used to provide a flexible abstraction layers between Virtual Reality (VR) system components and external software libraries and applications. Python is used as a glue language. VR framework provides a base for integrated work of C++ and Python in the field of virtual reality. [127]</td>
</tr>
<tr>
<td>3</td>
<td>Scientific modeling can be benefitted by integrating Python and C++. A discussion of Python's usage to steer C++ code and benefits of this integration are found in [128]. [128]</td>
</tr>
<tr>
<td>4</td>
<td>VTK is a visualization toolkit with its library written in C++ and providing interfaces to Tcl, Python and Java. It is claimed that [129]</td>
</tr>
</tbody>
</table>

A brief analytic discussion is provided in [nest] about python tools, frameworks and extensions and variants including Jython (for Java), IronPython (for .Net), PyPy (Python in Python), Stackless Python and Python decorators. [125]
efficient systems can be developed in VTK by integrating C++ and Python for speed and flexibility respectively.

5 A comparison of Python/C++ wrapper generation systems is included in [115]. Among SWIG, SIP, CXX, SCXX and Boost, Python, the last one i.e. Boost. Python is claimed to be the most suitable system to generate Python/C++ wrappers. [115]

6 A successful wrapping system to integrate C++ and Python is presented in [130]. An efficient code with the support of these two powerful languages is a lot easier to build with Boost. Python library. [130]

### 4.6.6 Performance

**Table 18: Performance**

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A strategy is devised in [131] to give direct access of system resources to Python developers. They can develop and manage synthetic programs by modifying critical performance areas.</td>
<td>[131]</td>
</tr>
<tr>
<td>2</td>
<td>Wrapped Application Debugger (WAD) is introduced to provide an error reporting and debugging mechanism for scripting languages particularly Python. It is believed to enhance the performance efficiency by helping in debugging during integrated software development.</td>
<td>[132]</td>
</tr>
<tr>
<td>3</td>
<td>Restricted Python (RPython) is a statically typed object oriented language and is a subset of Python. Two developments in JVM and CLI back-ends have been devised in RPython. The basic purpose is to avail the productivity benefits of Python while not compromising on performance issues.</td>
<td>[133]</td>
</tr>
<tr>
<td>4</td>
<td>Along with providing an implementation of genetic algorithm in Python, a number of performance enhancement tools for Python are also discussed in [134]. These include PyChecker, Numerical Python, Numarray, SWIG, SIP, Pyrex and Psyco.</td>
<td>[134]</td>
</tr>
<tr>
<td>5</td>
<td>Synthetic Programming Environment (SPE) is a library for Python, which generates and executes machine instructions at run-time. It provides mechanism to optimize serial and parallel applications without relying on intermediate languages.</td>
<td>[135]</td>
</tr>
<tr>
<td>6</td>
<td>No single language is single handed solution to develop large scale linear algebra software [136]. However, high performance can be achieved by mixing Python, C/C++ and Fortran code according to Pythonic framework presented in [136].</td>
<td>[136]</td>
</tr>
</tbody>
</table>

### 4.6.7 Usage

**Table 19: Usage**

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Technique/Idea Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A python toolbox is used for canonical problem solving in the field of electromagnetic.</td>
<td>[137]</td>
</tr>
<tr>
<td>2</td>
<td>Python is used to develop a tool for the Cryo electron microscopy.</td>
<td>[138]</td>
</tr>
<tr>
<td>3</td>
<td>Python is used to develop a compiler construction tool called TRAP. This tool can be used for rapid development of compilers and translators for medium complexity special purpose languages.</td>
<td>[139]</td>
</tr>
<tr>
<td>4</td>
<td>It is very easy to access online databases via Python. For learning purpose, [140] is a good description about querying MySQL over the internet.</td>
<td>[140]</td>
</tr>
</tbody>
</table>
5 PyGPS is a GPS data processing system developed in Python. [141]

6 Python can be very helpful for the laboratory environment automation. A special note on handling the Race Conditions is included in [118]. [118]

7 GPC, a generic job scheduler used in NASA's Shuttle Radar Topography Mission (SRTM), is implemented in Python. Python is found to be the most suitable choice as a GUI prototyping tool. [142]

8 PsychoPy is developed using Python to help in the field of Psychophysics. [143]

9 Python has clean syntax, flexible and available as open source. Its expressive power and VPython’s excellent graphics make it ideal for computational physics. [144]

10 Python supports bulk synchronous parallel (BSP) computing by providing classes supporting distributed data. Python/BSP avoids deadlocks and makes it better than message passing technique for parallel programs. [145]

11 Python is appreciated as a first programming language for computer science students. [146]

4.6.8 Analysis
Python has been recognized by industry giants like Google and Microsoft as they are providing its support in their development infrastructures i.e. IronPython in .Net framework [2]. It has been believed that Python can only be used for less critical software or as a glue language. But the results of systematic review shows that Python has been successfully used for both small and large scale products [12].

Evidences from industry shows that Python have a very vast range of applications in different fields ranging from electromagnetic [137], cryo-electron microscopy [138], compiler construction tools [139], accessing online databases [140], GPS [141], laboratory automation [118], NASA’s Shuttle Radar Topography Mission (SRTM) [142], Psychophysics [143], computational physics [144], bulk synchronous parallel processing [145] and as well as for education purpose [146].

There have been some efforts made to overcome the performance issue in Python. Performance is geared by giving focus to libraries improvement, better debugging support and providing performance tools. Some important findings are

<table>
<thead>
<tr>
<th>SR No</th>
<th>Area</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Performance</td>
<td>[131][132][133][134][135][136]</td>
</tr>
<tr>
<td>2</td>
<td>Integration of C++ and Python</td>
<td>[126][127][128][115][129][130]</td>
</tr>
<tr>
<td>3</td>
<td>Usage Domains</td>
<td>[137][138][139][141][143][144][145]</td>
</tr>
<tr>
<td></td>
<td>• Networks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Physics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Telecommunication</td>
<td></td>
</tr>
</tbody>
</table>

4.7 Smalltalk Developments
The following subsections elaborate the development scheme regarding Smalltalk.

4.7.1 Comparison

<table>
<thead>
<tr>
<th>SR No</th>
<th>Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ada is stated as more type safe as compared to OOPLs e.g. Smalltalk, C++. Hence It is claimed that Smalltalk is less suitable than Ada especially for critical systems.</td>
<td>[93]</td>
</tr>
</tbody>
</table>
Smalltalk, C++ and CLOS are examined in context of Object Oriented Programming. Smalltalk is claimed to be the strongest representative of OOP languages.

### 4.7.2 Dynamic Aspects

**Table 21: Dynamic Aspects**

<table>
<thead>
<tr>
<th>SR No</th>
<th>Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The benefit of 'type-less declaration' in Smalltalk is discussed in [148]. The author proposed a solution to provide similar feature in Java without JVM code modifications.</td>
<td>[148]</td>
</tr>
<tr>
<td>2</td>
<td>The Automatic Garbage collection in Smalltalk is discussed in [147]. Moreover a discussion of Dynamic Binding is included in context of object oriented languages.</td>
<td>[147]</td>
</tr>
</tbody>
</table>

### 4.7.3 Framework

**Table 22: Smalltalk Framework**

<table>
<thead>
<tr>
<th>SR No</th>
<th>Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A declarative framework developed in logic meta-language SOUL to give reasoning about the structure of Smalltalk.</td>
<td>[149]</td>
</tr>
<tr>
<td>2</td>
<td>Omni Browser framework is developed using Squeak Smalltalk (based on Smalltalk-80). Browsers based on explicit meta-model can be defined in this browser framework.</td>
<td>[150]</td>
</tr>
</tbody>
</table>

### 4.7.4 Language Development/Enhancement

**Table 23: Language Development/Enhancement**

<table>
<thead>
<tr>
<th>SR No</th>
<th>Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To separate the specification from implementation of reusable Smalltalk modules, Larch/Smalltalk is developed. The tools integrated into the Smalltalk programming environment are used to develop Larch/Smalltalk.</td>
<td>[151]</td>
</tr>
<tr>
<td>2</td>
<td>Chuck is a semantic navigation code browser for Smalltalk that supports very large code bases. It is based on DDP algorithm and Chuck is shipped with standard version of Squeak Smalltalk.</td>
<td>[152]</td>
</tr>
<tr>
<td>3</td>
<td>A library ByteSurgeon is introduced for Squeak Smalltalk in [153]. Dynamic on-the-fly modifications can be made in applications via ByteSurgeon ability to transform byte code at runtime.</td>
<td>[153]</td>
</tr>
<tr>
<td>4</td>
<td>A full exception handling system for object oriented language is presented in [154]. The implementation of internal exception handlers and their algorithms is based on Objectworks Smalltalk.</td>
<td>[154]</td>
</tr>
<tr>
<td>5</td>
<td>An attempt is made to introduce safe type system for Smalltalk language in [155]. It is claimed that this new type system will improve optimization.</td>
<td>[155]</td>
</tr>
</tbody>
</table>

### 4.7.5 Integration Support

**Table 24: Integration Support**
<table>
<thead>
<tr>
<th>SR No</th>
<th>Technique/Idea Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENVY/Developer is a collaborative team programming environment including optional capability of managing the organization and integration of C and C++ code within the Smalltalk.</td>
<td>[156]</td>
</tr>
<tr>
<td>2</td>
<td>An experience report [157] discussed a multiple-platform and multi-language distributed object oriented messaging system integrating C, C++ and Smalltalk. This system supported Common Object Request Broker Architecture.</td>
<td>[157]</td>
</tr>
</tbody>
</table>

### 4.7.6 Performance Related

<table>
<thead>
<tr>
<th>SR No</th>
<th>Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The MUSHROOM architecture is designed to support the implementation of Dynamic Object Oriented languages especially Smalltalk. Conventional architectures provide 20 to 30 percent slower performance of Smalltalk programs as compared to C. But Mushroom had special design to handle dynamic binding, memory allocation, and garbage collection. Therefore, MUSHROOM is claimed to be simpler and more efficient for dynamic Object oriented languages.</td>
<td>[158]</td>
</tr>
<tr>
<td>2</td>
<td>A tool is devised inside Smalltalk environment to convert Smalltalk programs into petri-nets. The benefit is to use petri-nets to check the code for concurrency and invariants.</td>
<td>[159]</td>
</tr>
</tbody>
</table>

### 4.7.7 Usage

**Table 25: Usage publications**

<table>
<thead>
<tr>
<th>SR No</th>
<th>Technique/Idea Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENVY/Embedded is a toolkit including a large set of facilities for Rapid Application Development. It includes tools to support interactive development of embedded systems. It also has the facility for profiling and analysis. This environment also includes cross development tools for packaging and debugging.</td>
<td>[156]</td>
</tr>
<tr>
<td>2</td>
<td>CAOSTALK M&amp;S is a modeling and simulation environment which is developed using Smalltalk. Its main purpose is to implement HFSS i.e. Heterogeneous Flow System Specification which is a modeling formalism for hierarchical dynamic structure Hybrid systems.</td>
<td>[160]</td>
</tr>
<tr>
<td>3</td>
<td>Suitability of Smalltalk for VLSI CAD systems is discussed. The author pointed the object oriented and dynamic nature of Smalltalk to be the reason of its suitability for CAD systems.</td>
<td>[161]</td>
</tr>
<tr>
<td>4</td>
<td>Smalltalk/V is a Smalltalk dialect used mainly for VLSI circuit designing. The authors discussed a prototype for VLSI circuit editor in this language.</td>
<td>[162]</td>
</tr>
<tr>
<td>5</td>
<td>Smalltalk is used to verify the model representing the notion of Prototype-Based Languages. The idea is to provide a new taxonomy to enable constructive proposals and their validation.</td>
<td>[163]</td>
</tr>
<tr>
<td>6</td>
<td>A new process is introduced based on Extreme Programming and Smalltalk is used as underlying language. Smalltalk has direct support for refactoring that’s why the authors used Smalltalk for prototyping.</td>
<td>[164]</td>
</tr>
<tr>
<td>7</td>
<td>Squeak Smalltalk is used to develop Chuck. The main benefit of Chuck is to help programmers for Semantic navigation for large code bases, edit and analyze its semantics without any delay overhead.</td>
<td>[152]</td>
</tr>
<tr>
<td>8</td>
<td>Smalltalk-80 was used to develop a prototypic environment for</td>
<td>[165]</td>
</tr>
</tbody>
</table>
automatic robot assembly. The focus of the paper was on acquisition, analysis and validation of information for automated mechanical assembly in CAD systems.

9 The increased use of Smalltalk based information systems in Insurance industry is discussed in [166]. The object oriented nature, improved quality, extension flexibility and reduced maintenance costs are some of the benefits of Smalltalk.

### 4.7.8 Analysis of Smalltalk

Smalltalk results have verified a common trend in the research of dynamic languages. It has been observed that there are some parts in which researchers have given more focus. According to the study results most work is done in language enhancement and usage to support other areas e.g. insurance industry, simulation, rapid application development etc. From the results of studied languages, there have been less involvement found for improvement of performance in dynamic languages.

Smalltalk results shows that there have been more focus found towards language enhancement and usage. Language enhancement focuses on built-in libraries, type system and provide better code browsers for smalltalk. These enhancements are somewhat improvements in performance for smalltalk. But still more work is needed to improve performance.

Furthermore, our study reveals that the researchers are trying to make bridge between static and dynamic languages development environment. The benefit will be a single development environment where programmers can write code for both languages. For example, there have been some studies found for integration between C,C++ and Smalltalk. They have tried to give a single environment for these languages. Smalltalk is also used for giving support in insurance industry, simulation, artificial intelligence etc.

### 4.8 Conclusion Drawn from Systematic Review

On the basis of extracted information we concluded that dynamic languages are lacking in some areas. These areas include performance, frameworks and tools for language support. Performance is missing from the popularity group factors for dynamic languages. There have been different school of thoughts found. Some of them claims that its due to hardware. Whereas some of them claims that it needs better libraries.

It is also revealed from systematic review that researchers wants a common development environment for both static and dynamic languages. The work is ongoing to get amalgamation of common development environment for both type of languages. It is also observed that most of the integration is done for C/C++ with other dynamic languages like Python etc.

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Following are the popularity factors found for dynamic languages:</td>
</tr>
<tr>
<td>- Flexibility</td>
</tr>
<tr>
<td>- Type less Declaration</td>
</tr>
<tr>
<td>- Programmer Productivity</td>
</tr>
<tr>
<td>- Dynamic Typing</td>
</tr>
</tbody>
</table>

**Table 25: Conclusion from Systematic Review**
4.9 Answering the Research Questions Stated in Review Protocol

In this section research questions addressed in review protocol are answered.

**RQ.1.** What is the current state of the art in research on dynamic programming languages and their comparison with static languages?

The researches in dynamic programming languages have been ongoing and the current available studies found are described from section 4.1 to 4.7. A detail study related to comparison is described in Section 2.2

**RQ.2** What are the aspects of dynamism for a programming language?

The dynamic aspects play an important role in the popularity of dynamic languages. A detail description of dynamic aspects is given in section 2.1.
5. RESULTS REGARDING VERIFICATION AND VALIDATION IN DYNAMIC LANGUAGES

The results gathered from systematic review regarding verification and validation in dynamic languages are reported in this chapter. These results are based on the set of dynamic languages selected for state of the art research.

5.1 Erlang

There have been some developments found in Erlang regarding verification and validation. These developments are according to models, techniques and framework etc.

<table>
<thead>
<tr>
<th>Framework/Model</th>
<th>Purpose</th>
<th>Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A discrete time model to extend the existing work and to support the translation of timed Erlang functions into μCRL.</td>
<td>Use of explicit \textit{tick} action in the μCRL specification supports to enhance the existing work on translation of Erlang components to μCRL.</td>
<td>[79]</td>
</tr>
<tr>
<td></td>
<td>Framework is developed for abstract interpretation and model checking</td>
<td>The framework takes into consideration of those properties of a system that must hold in all paths e.g. the properties in LTL. These properties are then checked for abstract operational semantics and if they are proved, then it is deducible that they hold for Erlang programs.</td>
<td>[80]</td>
</tr>
<tr>
<td></td>
<td>An approach towards development of verified Erlang programs is discussed.</td>
<td>Suggested a tool including translation component, state space generation component and a model checking component for Erlang programs.</td>
<td>[81]</td>
</tr>
<tr>
<td></td>
<td>An approach towards Automated tool development for verification of Erlang programs</td>
<td>Manfred used the concept of flow graphs of functions for coverage criteria to help in analyzing the Erlang programs [5]. Once control flows of functions is traced for single order as well as higher order functions. It can be recorded in the form of graphs and hence may lead to the development of an automated tool for testing Erlang programs.</td>
<td>[82]</td>
</tr>
<tr>
<td></td>
<td>An approach to formally model and verify the Erlang programs</td>
<td>It formally models and verifies Erlang programs by mapping programs written in Erlang to (\pi)-calculus. It used automated model checking techniques instead of simulation or testing and introduced a process-algebraic model. The model is applied on a subset of Erlang called PiErlang and future work may bring a model applicable to complete Erlang language features</td>
<td>[83]</td>
</tr>
<tr>
<td>Tool</td>
<td>EUnit framework to help in testing Erlang code.</td>
<td>The framework is based on lightweight unit testing concepts and influenced from JUnit written by Kent Beck.</td>
<td>[84]</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>Tool (Dialyzer) for analyzing discrepancy of Erlang programs.</td>
<td>It is a defect detection tool that makes use of static analysis and automatically detects and removes software code errors. Nowadays a part of Erlang/OTP</td>
<td>[85] [86]</td>
</tr>
<tr>
<td></td>
<td>A Tableau-based proof system to make use of Verification techniques applied on $\mu$CRL</td>
<td>The tool provides facility to backward generate a (tableau) tree that helps users in model checking and verification.</td>
<td>[87]</td>
</tr>
<tr>
<td></td>
<td>McErlang helps to verify Erlang programs in a better way as compared to Etomcrl tool and to increase modifiability.</td>
<td>The tool is developed in Erlang hence correctness properties are available in Erlang which increase the ease in identifying bugs by model checkers</td>
<td>[88]</td>
</tr>
<tr>
<td></td>
<td>To test the Erlang programs effectively Quick-Check is introduced.</td>
<td>This is a property based testing tool that is based on the same tool for Haskell. Ericsson is also using this tool.</td>
<td>[89]</td>
</tr>
<tr>
<td>Techniques</td>
<td>Erlang processes are concurrent in nature and the current technique i.e. Dependency Pair technique is found to be well suited for Erlang processes' verification.</td>
<td>Dependency pair technique is considered more suitable for automation as compared to other related approaches for proof checking [21]. Narrowing, rewriting and instantiating dependencies are undertook to manipulate dependency pair technique and applied to termination proofs of CTRSs Conditional Term-Rewriting Systems</td>
<td>[92]</td>
</tr>
<tr>
<td></td>
<td>Thomas et al [20] used their semi formal analysis method for fault tolerant distributed algorithms on leader election algorithm.</td>
<td>Through analysis of algorithm, stimuli creation for traces generation and by applying formal methods, they found bugs in existing implementation of leader election algorithm.</td>
<td>[91]</td>
</tr>
<tr>
<td>Test Suite</td>
<td>This test suit project helped in detecting faults and hence resulted in a world-class middleware certified to telecom-class quality</td>
<td>Erlang’s debugging options: declarative syntax and pattern matching are helpful in developing the high quality test suit environment. It run on nightly builds with thousands of test cases developed by individual programmers, designers and testers.</td>
<td>[90]</td>
</tr>
<tr>
<td>Research Work</td>
<td>Work on automating verification process.</td>
<td>Formal reasoning, linear algebra and model checking techniques using $\mu$-calculus for automating verification process.</td>
<td>[73]</td>
</tr>
</tbody>
</table>
5.2 Python

There have been some developments found in Python regarding verification and validation. These developments are according to models, techniques and framework etc.

Table 27: Verification and Validation in Python

<table>
<thead>
<tr>
<th>Technique/Idea Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python’s own debugger is found helpful to find the faults while using Python/BSP API for</td>
<td>[145]</td>
</tr>
<tr>
<td>bulk synchronous parallel computing.</td>
<td></td>
</tr>
<tr>
<td>JPython, a Python implementation in Java, is found to be very useful and easy to develop</td>
<td>[167]</td>
</tr>
<tr>
<td>tests for any Java class.</td>
<td></td>
</tr>
</tbody>
</table>

5.3 Smalltalk

Focus of this section is on the research material related to the area of verification and validation. Following is a brief description of the articles for V&V in Smalltalk.

Table 28: Verification and Validation in Smalltalk

<table>
<thead>
<tr>
<th>Technique/Idea Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td>An automated test driven development approach AOTDD is presented in [168]. The approach</td>
<td>[168]</td>
</tr>
<tr>
<td>is implemented using SUnit (a Smalltalk testing framework) devised in this paper for the</td>
<td></td>
</tr>
<tr>
<td>first time. The SUnit framework and AOTDD approach is claimed to be better for iterative</td>
<td></td>
</tr>
<tr>
<td>and incremental software development.</td>
<td></td>
</tr>
<tr>
<td>A workshop is held to investigate testing in Smalltalk and discussion on various issues</td>
<td>[169]</td>
</tr>
<tr>
<td>is enlisted in [169]. It is identified that the major difference in testing Smalltalk</td>
<td></td>
</tr>
<tr>
<td>applications with other languages is the development environment. However it is observed</td>
<td></td>
</tr>
<tr>
<td>that Smalltalk community lacks standard discipline/tools for testing and measuring quality.</td>
<td></td>
</tr>
</tbody>
</table>

5.4 Analysis of Verification and Validation

It is observed from the systematic review that verification and validation is also not focused by researchers. Systematic Review results reveals that there is a limited work available in verification and validation for dynamic languages. Some of the languages (lisp, smalltalk and self) have been around from so many years but there is no focus given towards these languages. Among these studied languages, Erlang has given some more importance in verification and validation as compared with other dynamic languages like
Lisp, Self, Smalltalk, Ruby and Python. For smalltalk, a unit testing framework is developed called Sunit.

There is also a test suite developed for handling verification issues and some research is ongoing for automating verification in Erlang. There are some other frameworks and tools found from web references. These tools and frameworks are being used for verification and validation in different dynamic languages.

### Table 29: Test Tools

<table>
<thead>
<tr>
<th>Framework/Tool Description</th>
<th>Acknowledged by</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ruby</strong></td>
<td></td>
</tr>
<tr>
<td>RSpec</td>
<td>[173]</td>
</tr>
<tr>
<td>TestUnit</td>
<td>[174]</td>
</tr>
<tr>
<td><strong>Python</strong></td>
<td></td>
</tr>
<tr>
<td>JPython</td>
<td>[167]</td>
</tr>
<tr>
<td>Pytest</td>
<td>[175]</td>
</tr>
<tr>
<td>PyUnit</td>
<td>[176]</td>
</tr>
</tbody>
</table>

![Figure 4 – Verification and Validation Results](image-url)
6. SURVEY

The Survey for verification and validation in dynamic programming languages has been carried out in two ways i.e. Questionnaire based online Survey and Interviews. This chapter elaborates the questionnaire based online survey and description of Interviews is described in chapter 7.

6.1 Questionnaire Based Online Survey

Following subsections describe the survey objective, methodology and participants details. The next section deals with survey outcomes and analysis based on the achieved results.

6.1.1 Methodology

6.1.1.1 Objectives

The main purpose of the questionnaire is to obtain the tacit and practical knowledge from industry practitioners regarding V&V in Dynamic Programming Languages. The objectives of this study are:

1. Examine the state-of-practice V&V methodologies in industry with the perspective of dynamic languages.
2. Examine the differences between state-of-art and state-of-practice in V&V methodologies in dynamic languages.

The questionnaire is devised to validate the results of systematic review in comparison with industry practices in the area of V&V in dynamic languages.

6.1.1.2 Description

There are five main sections in the questionnaire i.e. Warm-up/Introduction, Verification & Validation, Problems/Issues, Techniques/Tools and Summing-up/Suggestions. In general, all the sections in this questionnaire contribute to cover both of the aforementioned objectives. However, specifically the first, second and fourth sections cover the first objective, while third and fifth sections are more specific to second objective.

For the sake of simplicity, minor rearrangements of some questions have been performed while reporting the results. The questions are regrouped in the following subsections with a little modification in their original sequence, as depicted in Appendix A.

6.1.1.3 Method

We have selected web-based medium to target a wide range of industry practitioners. Initially the questionnaire consisted of 21 questions. After a pilot study, the questions are modified and reduced in number. Final questionnaire consists of 15 questions (Appendix B). Majority of the questions are open-ended, allowing the respondents to reflect their views flexibly and to cover the areas beyond the scope of close-ended. Final questionnaire is made available on internet and respondents are able to access, process and save results online.

6.1.1.4 Population & Response Rate

It is already mentioned that the focus of the survey are industry practitioners who are specialized in the field of V&V in Dynamic languages. Therefore, the survey participants are selected after a very careful investigation of their credentials. Google special purpose groups are used to search for respondents who are active participants of discussions related to the said area. Only those participants are selected who either had a good star-rating, a popular blog, or found imparting their implicit knowledge in response to queries in group discussions. Out of all potential respondents, 18% have participated in the survey.
6.2 Questionnaire Outcomes

6.2.1 Warm-up/Introduction

This section is related to participants’ background and their professional experience related to our area of research. We have collected brief and specific information from their professional profile that might help in validating the survey results. Our online survey is not intended to target a particular company or a region and that’s why respondents are geographically dispersed. Hence there is no possibility of any biasness. Additionally, almost all the respondents are well educated (Bachelors 38.89%, Masters 33.33%, PhD 16.67% and 5.56% belonged to Post Doctorate, some College, and High School). See Figure 5.

![Figure 5: Representation of respondents’ education level.](image)

The most popular languages among all the respondents are Lisp (55.56%), Python (50%), Ruby and Smalltalk (38.89% each) and Erlang (11.11%).

![Figure 6: Representation of respondents’ involvement with DLs.](image)

We have only selected industry practitioners to participate and expect them to have a good experience as well. The average experience of respondents is 10 years in industry. A detailed graph is given in Figure 7 below. Out of all the respondents 61.11% have more than 8 years of experience, while 22.22% and 16.67% are having 2-4 and 4-8 years of experience respectively.
6.2.2 Verification & Validation and Dynamic Languages

This section enlightens the V&V practices in industry regarding dynamic languages with the perspective of respondents’ professional experience in dynamic languages. Some questions from previous portion are included in current section and its subsections due to their relevancy to its perspective.

The respondents have been involved in Unit Testing (88.24%), Integration Testing (76.47%), System Testing (58.82%) and Acceptance Testing (47.06%). Other kinds of testing activities with 5.88% rating are Code Quality / LINT Check, Load/Performance/Scaling Testing, Performance Testing, Smoke Testing, Story Framework and Trial & Error. Figure 8 shows the feedback level from the respondents.

85% of respondents are supporting the view that dynamic languages do not pose any hindrance to maintain the software quality. However it has been consistently mentioned that better IDE support is required for dynamic languages. The static languages are in general rich in this regard.

6.2.2.2 Dynamic Languages Aspects Important for V&V

To examine the importance of dynamic aspects for V&V, the respondents have been asked to select among some given options and to specify other aspects if they feel important. We have analyzed that dynamic typing (55.56%) and absence of type declaration have been considered the most important dynamic aspects with respect to V&V. Respondents have also specified their opinions about several other aspects and features important for V&V, enlisted in table 31.
Table 30: Important Aspects for Verification & Validation

<table>
<thead>
<tr>
<th>Aspects/Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language Features</strong></td>
</tr>
<tr>
<td>ability to write DSLs</td>
</tr>
<tr>
<td>high quality container implementations</td>
</tr>
<tr>
<td>powerful libraries</td>
</tr>
<tr>
<td>class introspection</td>
</tr>
<tr>
<td>meta-classes</td>
</tr>
<tr>
<td>test driven and prototype rich methods</td>
</tr>
<tr>
<td><strong>Closures</strong></td>
</tr>
<tr>
<td>dynamic code generation/import</td>
</tr>
<tr>
<td>dynamic code modification</td>
</tr>
<tr>
<td>ease to fix and continue</td>
</tr>
<tr>
<td>dynamic deployment</td>
</tr>
<tr>
<td>highest programmer productivity</td>
</tr>
<tr>
<td>code readability</td>
</tr>
<tr>
<td><strong>Dynamic Aspects</strong></td>
</tr>
<tr>
<td>focus on BDD</td>
</tr>
<tr>
<td>tight feedback loops in V&amp;V</td>
</tr>
<tr>
<td>robust error handling</td>
</tr>
<tr>
<td>the ability to reflect for testing</td>
</tr>
<tr>
<td>tools for acceptance</td>
</tr>
</tbody>
</table>

6.2.2.3 Difference in V&V Methodologies between DLs and Static Languages

There are two schools of thoughts found for verification and validation methodologies; one thinks there is a difference in methodologies and others think don’t. Majority of respondents (60%) think that there is no difference in V&V methodologies between dynamic and static languages. Other respondents (40%) think that dynamic languages focus more on testing and debugging. In static languages sometimes the testing activities may be skipped in case of known or easily reviewable code but in dynamic languages testing is very rarely skipped. To conclude, we may say that as such there is no difference in V&V methodologies between both kinds of languages. The difference lies only in relatively more (in case of dynamic languages) and less (in case of static languages) focus on applied testing activities.

6.2.2.4 Aspects influencing the Popularity of Dynamic Languages

To examine the popularity of dynamic languages, we have investigated six carefully identified factors from Systematic Review. Survey respondents think that flexibility (89.9%) is the most influencing factor for the popularity of dynamic languages. Immediately following are programmer productivity (83.3%) and RAD (83.3%). Modern IDE tools (77.8%) greatly influence the popularity of a dynamic language but they are ranked lower than flexibility, productivity and RAD. Code optimization and performance are regarded considerably less important with the perspective of fame of dynamic languages. The detailed results are enlisted in Figure 9.
6.2.2.5 Problems/Issues

The results show that most of the respondents (80%) do not face any specific issues in dynamic languages as compared to static languages. Whereas, some of the respondents (5%) think that the handling of *internal exception code* can be problematic in dynamic languages. Others (10%) had the opinion that inexperienced programmers may write sub-standard code which in return creates difficulty to update due to high flexibility offered by the dynamic languages.

6.2.3 Techniques/Tools

There are a lot of different tools and frameworks being used in industry. Table 32 includes the important ones specified by survey respondents. We have observed that most of these tools and frameworks are open source.

<table>
<thead>
<tr>
<th>Tools and Frameworks used in industry</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Python</strong></td>
</tr>
<tr>
<td>Pytest</td>
</tr>
<tr>
<td>Python Unittest</td>
</tr>
<tr>
<td><strong>Ruby</strong></td>
</tr>
<tr>
<td>RSpec (Behavior driven development in Ruby)</td>
</tr>
<tr>
<td>TestUnit (framework for unit testing in Ruby)</td>
</tr>
<tr>
<td>Ruby zentest/unittest testspec</td>
</tr>
<tr>
<td>Rake</td>
</tr>
<tr>
<td>Rails</td>
</tr>
<tr>
<td>Tools and Frameworks</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>EUnit test framework</td>
</tr>
<tr>
<td>SUnitToo</td>
</tr>
<tr>
<td>TextTest</td>
</tr>
<tr>
<td>TestMentor UI test</td>
</tr>
<tr>
<td>Nose</td>
</tr>
<tr>
<td>Mock Objects</td>
</tr>
<tr>
<td>XUnit</td>
</tr>
<tr>
<td>Selenium</td>
</tr>
<tr>
<td>JUnit</td>
</tr>
<tr>
<td>Assert</td>
</tr>
</tbody>
</table>

### 6.2.4 Suggestions/Opinions

The survey results revealed the positive comments and thoughts about ongoing advancement in dynamic languages. Some of them suggest more work in the area of better IDE support. Others have mentioned the need to practice test driven development (TDD) and behavior driven development (BDD). Other than these, some contradicting statements have also been raised. For example one of the respondents has said that dynamic languages are needless.

### 6.3 Analysis

In general, the results of the online survey affirm the outcomes of systematic review and also able to give results for V&V. However, there are a few exceptions which have been discussed in later part of current section.

All of the respondents have several years of industry experience involving dynamic languages and verification & validation. They have worked with several languages including but not limited to Lisp, Python, Ruby and Smalltalk. Moreover, Unit Testing is the most frequently performed activity that our respondents have been involved in.

The survey results depict that flexibility is the main reason of preferring dynamic languages over static languages. Mostly, the programmers adopt dynamic languages where rapid application development is required. It is also observed that dynamic languages increase the programmer productivity with its rich libraries and flexibility.

It is observed that dynamic languages still lack behind in some areas like performance and code optimization. The results regarding these areas are slightly different from the systematic review results. Both of these aspects have been found to be less important and considered not relevant to affect the popularity of dynamic languages significantly.
Survey results show that run-time code modification along with dynamic typing and type-less declaration are considered the most important aspects of dynamic languages for V&V.

There is no difference found in V&V methodologies for dynamic and static languages. The only difference is that dynamic languages emphasize more on testing and debugging as compared to static languages. In fact dynamic languages demand rigorous automated testing procedures to ensure the software quality. It is also observed that no testing is required to check for type safety issues in dynamic languages. The tests written to validate the behavior of classes/objects are considered enough to remove any mishandling of data. This gives an extra edge to dynamic languages by eliminating the need of compiler. However, the test code needs to be written in a careful and disciplined manner.

There are several tools and frameworks have been identified in online survey and are listed in table 31. Most of them are open source software. There is no need for another framework for testing as already existing frameworks are well equipped. Almost all the existing test frameworks are based on Smalltalk testing framework (SUnit). Even the famous testing framework for Java (JUnit) has been developed on the same grounds as SUnit. This fact proves the suitability of using dynamic languages for development and testing purpose.

Dynamic languages support to maintain software quality. These libraries allow the programmer to write a code relatively in fewer lines as compared to static language. This enhances the flexibility and productivity for the programmer. Furthermore Test Driven Development (TDD) and Behavior Driven Development (BDD) should be adopted while working with dynamic languages. According to our knowledge, TDD and BDD should play an important role in improving and maintaining the quality of software.

Dynamic languages do not specifically pose a hindrance in maintaining the quality of software. However there exists a contradicting opinion that the absence of type system refers to absence of a ‘safety net’ in dynamic languages. But our analysis of survey results and systematic review motivates our point of view that it is only a problem in case of Junior/Inexperience programmers developing a substandard code.

In light of the results of literature review and industry survey we are confident to suggest that there is a need to adopt TDD in dynamic languages. It is also observed from survey results that writing tests in a dynamic language is easier than static languages. Quite often a test suit built in dynamic languages is used to test the software developed in a static language.
7. INTERVIEWS FOR VERIFICATION AND VALIDATION IN DYNAMIC LANGUAGES

In order to get concrete results regarding verification and validation and to have a comparison with literature, authors conducted some interviews to get thoughts from experienced professionals. This chapter will not include every detail of the interview but it would be a collective analysis of their thoughts, commonalities and contradictions. Furthermore, the things are described in the form of categories which are being used in the questionnaire (Refer to Appendix B).

7.1 Interviewee’s Introduction

In order to get a picture of Verification and Validation in Dynamic Languages, authors have selected three experienced professionals (X,Y,Z) having a wide experience and knowledge in both static and dynamic programming languages. All the interviewees have an average of 8 years experience in working with dynamic languages. Furthermore, they have earned high school degree and two of them are owning their companies. The interviewees have worked with more than one languages like Ruby, Python, Lisp and Java etc. The rationale for selection is their wide experience and work with more than one language.

7.2 Verification and Validation

Verification and Validation is performed with the help of available test frameworks for particular language. All of them are involved in the unit testing but interviewee X,Y are also involved in integration testing as well. According to all the interviewees, they focused on the test driven development approach while working with dynamic languages. They have mentioned TDD(Test Driven Development) an important step to get a quality software. X and Y have also pushed dynamic languages to support quality attributes like maintainability of the software.

It has also been revealed that programmer productivity, flexibility and tool support also increases the popularity graph. The interviewee ‘Y’ has also focused that if rapid software development and performance become more stable. Then these factors will raise the popularity even to a larger scale.

7.3 Problems/Issues

Problems and uncertain issues can sometimes affect the quality of work. According to interviewees, no specific problem or issue raised which affected negatively on quality of work.

7.4 Techniques/Tools

Automated tools and frameworks have made the things simpler for professionals. These tools and framework have tremendously decreased the time constraint in software development. The interviewees have common skills and a kind of same experience of working with same languages. In their experiences they have used some common tools like Rspec, Nunit, Xunit, Jpython, Eunit and Pyunit etc. There are different tools which are used with different languages like Rspec for Ruby, Pyunit for Python etc. The common trend for using these tools is using through Coding ➔ Testing Cycle.
7.5 Analysis of Results

From interviews we come up with commonalities and disagreements with thoughts and experiences. We concluded that dynamism in dynamic languages do not create hindrance in the way of software quality. It has also revealed from industry experiences that dynamic languages are also used to test the code of static languages. It is also revealed that dynamic languages support to maintain software quality in both ways i.e. when it is used to test software developed in dynamic language and also for software developed in static languages.

Furthermore, an issue raised by all interviewees is to focus towards TDD (Test Driven Development). They have emphasized TDD as mandatory for dynamic languages. From their perspectives authors have also verified some aspects/characteristics found in Systematic Review like performance, flexibility, less code etc.

There are also some contradictions found regarding performance issue, interviewee 'X' considers performance less important as compared with other aspects of dynamic languages like flexibility, maintainability etc. He emphasized that sometimes it can be achieved by re-writing the code to solve a particular problem. Hence due to dynamic nature of these languages we can apply different solutions and select the suitable one with higher performance. Whereas the other interviewees, 'Y' and 'Z' consider performance an important issue but they also consider current hardware and technology to reduce performance issue.

Furthermore interviewee 'Z' also thinks that dynamic languages are lacking behind in some aspects such as less understandability in code, need more testing tools etc. We think the reason could be type less declaration which causes confusions or make the things unstructured to read.

We concluded from interviews that dynamic languages have made their way in industry. The professionals are well aware of the strengths of dynamic languages and they don’t feel any problem to use them. We have also concluded from the interviews and online survey that dynamic languages need test driven development for further growth. Because there are stable and good frameworks available in industry and these frameworks are based on SmallTalk unit testing framework i.e. Sunit. We don not feel to propose a model for verification and validation for dynamic languages. The only need is to reduce the performance issue and test driven development should be adopted to improve the software quality.

7.6 Research Questions Addressed by Online Survey and Interviews

This section address the answers found from online survey and interviews.

RQ.3 Does dynamic languages support or create hindrance to maintain the software quality?

The results obtained from online survey and interviews reveals that dynamic languages don’t create any hindrance in maintaining the software quality. Most of the industry people think that dynamic languages help to improve the quality with their features and dynamic characteristics. More detail is given in section 6.2.2.1 and 7.2.

RQ.4 What are the popularity factors necessary to raise the graph of dynamic languages?

Dynamic Languages are commonly used in industry along with static languages. The study
results reveal that dynamic languages are still popular in industry. But due to some certain factors, people are ignoring them. Among these factors, Performance is main issue which decreases the graph for dynamic languages. Different factors are discussed in section 6.2.2.4 which increases or decreases the popularity of dynamic languages.

**RQ.5 What are the differences between verification and validation techniques and tools for static and dynamic languages?**

The differences between verification and validation techniques are discussed in sections 6.2.3, 7.2, 6.2.2.3 and 6.2.2. These sections also describes the verification and validation tools and practices found in industry. The results reveals that there is no difference in techniques for static and dynamic languages. The same verification techniques such as inspection, reviews etc are used for dynamic languages. It is also extracted that unit testing framework for Java is also developed from Smalltalk unit testing framework i.e. SUnit. Another interesting point is extracted that unit test frameworks for dynamic languages are also child of SUnit framework. The same framework is mold in their respective languages like EUnit, RUnit etc.
8. **Discussion**

The systematic review results reveal some interesting results. There have been found a common trend in research, most of the research is done in language enhancements, performance and usage. The performance work is related to enhance the libraries, giving better debug support etc and also every language has contributed to support other fields like networks, artificial intelligence etc. Further description about the no of articles in each development sub scheme and usage can be found in the given figure.

There is a group of people who thinks that performance is a key issue in dynamic languages. Although having bad performance, these are used in the industry. The survey and interviews results show that dynamic languages are suffering with performance issue. But industry professionals don not care about this issue.

Features and characteristics found for popularity of dynamic languages in systematic review are also considered important by industry professionals. These popularity factors (such as code optimization, dynamic typing and type less declaration) along with rapid application development, better IDE support and performance are considered an important combination for popularity.

Another point is highlighted in academia that dynamic languages cannot be used for the development of complex and large systems. But the survey and interview results negate this statement. The industry professionals think that dynamic languages can be used for complex and large projects. But they also think that dynamic languages are not fit for every domain.

---

**Overall Research detail in each sub-scheme**

Verification and Validation is also not focused in the articles found for systematic review. There have been tools and frameworks described which are used in Erlang, Smalltalk and Python. The figures for verification and validation articles can be seen from the above given figure. The industry professionals feel that there is no difference between the techniques available for verification and validation. For example, reviews, inspection and walkthroughs are used for dynamic languages. But interesting thing is that not even single article is available which support this claim. Furthermore, another important aspect rise that dynamic languages have stable and reliable test frameworks and built-in classes. The only
improvement needed is to adopt Test Driven Development (TDD) with dynamic languages. It is believed that Test Driven Development is mandatory for dynamic languages.

8.1 Threats to Validity

The accuracy of research design can be improved by identifying those potential factors which can affect the results. In this section four different kinds of validity threats and their implications are discussed. These four types of threats as outlined by [170] are conclusion validity, construct validity, internal validity and external validity.

8.1.1 Conclusion Validity (Reliability)

Conclusion validity is used to ensure the reliability of results taken from a particular study. These threats are concerned with those issues that affect the reliability of results [170] [171] [172]. Conclusion validity is also known as reliability [171]. The review protocol (Section 3.1.2) used for systematic review is reviewed by two researchers one from Blekinge Institute of Technology, Sweden and other from outside Sweden.

The questionnaires used for industrial interviews and online survey are also validated to remove the unnecessary and unclear questions. To make it more accurate, a pilot study is also conducted to make further improvements in questionnaire. The heterogeneity of subjects is also another threat to conclusion validity [170]. Heterogeneity of subjects means that the subjects belong to different group with respect to background, education and experience. In our study subjects are from the same group discipline i.e. dynamic programming languages and there is no subject having different discipline. In spite of this, some of them have some extra skills. From the study results the subjects have an average of 8 years experience (section 6.2.1). Therefore the subjects of our industrial interviews and online surveys are homogeneous.

8.1.2 Construct Validity

Construct validity identifies the relationship between theory and observation [171]. For instance if we are using survey as a research methodology for a research study, then according to construct validity the results of survey should be generalized to the main idea behind the survey [170]. It should focus on the rationale behind conducting the survey. Evaluation apprehension is the main threat to construct validity. According to [170], evaluation apprehension means that humans have the tendency to look better when they are evaluated. Some people do not want to share their internal information and their personal information. In order to eliminate this threat the interviewees and respondents of online survey are ensured about their confidentiality.

Mono-operation bias is another threat to construct validity. Mono-operation bias occurs when you are treating with a single case or subject in a research study [170]. We have eliminated this threat by conducting interviews and online survey with respondents from different companies and geographical location.

8.1.3 Internal Validity (Causality)

Internal validity ensures that research design should allow researchers to draw results and conclusions according to the different effects found from study [170] [171] [172]. Internal validity is also known as causality [171].

The interviews are recorded to eliminate this threat. It is assured to interviewees that recordings are only for thesis purposes and there information will not be leak out. To remove the publication bias, authentic resources are used for study. Selection of subjects also has impact on the results of study [170]. In our study we have selected the respondents from the same area i.e. dynamic programming languages. However the only constraint is of their educational background, experience and practices in the studied area.
8.1.4 External Validity (Generalizability)

External validity deals with generalizability of the results means it ensures that the results can be generalized or not [170] [171] [172]. One of the threats to external validity is the interaction of selection and treatment [170]. But in our study the subjects are working in dynamic programming languages. But one concern might affect that whether these results can be generalized for practices of dynamic languages or not. Our results show a common trend that people thoughts and practices have commonalities with each other. But we can generalize it because our respondents are geographically dispersed and they have common skills and experience.
9. EPILOGUE

In this chapter the conclusions from systematic review and survey are presented.

9.1 Conclusions

This thesis presents the study of state of the art research in dynamic languages and practices used for verification and validation. The two major contributions of thesis are systematic review of dynamic languages along with verification and validation and survey. The survey is intended to validate and improve results regarding verification and validation from industry.

The thesis is intended to fulfill the requirements of systematic review and it is devised according to standards for systematic reviews. The thesis dealt with all the research questions defined with respect to aims and objectives. It presents a study of current research in dynamic languages over the last two decades. As dynamic languages are introduced in 60’s and major research is done between 90’s and 2000’s.

The results reveal that there has been some advancements found in dynamic languages. The most researched areas for dynamic languages are language enhancement, integration with static languages for common development environment, dynamic aspects and less focus towards performance. It is also seen that dynamic languages are also used to support other areas e.g. artificial intelligence, networks, robotics etc.

Based on systematic review results it is intended that dynamic languages don’t have support for large projects. But this issue has been resolved from interviews and online survey that it is suitable for small and large projects. But another interesting point raised that it can not be used for all domains.

The issues found in systematic review has no importance for industry. It has been claimed that dynamic languages are slow in performance. But most of interviewees and respondents dont care about performance issue. It is claimed that modern technologies have reduced this gap. It is also concluded from systematic review results that performance is still a critical issue for the growth of dynamic languages. There is a need of more research for performance issues. Furthermore, some researchers tried to improve the performance by improving libraries and debug support.

The dynamic aspects and strong features have created a better space for dynamic languages in industry. From systematic review, it has been observed that dynamic languages are better than static languages in many ways. For example they give flexibility for programmer to change their code at runtime and also gives code optimization facility. From systematic review, some popularity factors are identified which raise the popularity graph as compared with static languages. These popularity factors not limited to flexibility, programmer productivity, dynamic typing, type-less declaration and automatic memory management.

From interviews and online survey results, it has been verified that these factors can raise popularity of dynamic languages. Some more listed factors can also raise the popularity graph such as modern IDE (Integrated Development Environment), rapid application development and code optimization. It is also concluded that performance can influence on popularity graph very much.
Verification and Validation plays an important part for maintaining the quality of developed software. From systematic review, there is a very limited evidence found for verification and validation in dynamic languages. Erlang being the most successful language in getting attention towards verification and validation issues. There is less attention found towards smalltalk, ruby, python, self and lisp. There are some tools and frameworks found for verification and validation (section 5.4). But from the results of online survey and interviewees, some of these tools and frameworks are being used by industry. It is concluded that these tools and frameworks are considered less important for researchers. There are different tools, framework and techniques found for verification and validation in industry like Sunit, Eunit, Rspec, PyUnit etc.

The results from interviews and online survey depicts that there is no difference between verification and validation techniques for static and dynamic languages. Industry is using the same verification techniques for dynamic languages like inspection, reviews etc. It is also observed that industry is not facing any problems and issues regarding verification and validation in dynamic languages. The only concern found is test driven development which is believed mandatory for dynamic languages.

The results also reveal that dynamic languages don not create hindrance to support quality in developed softwares. It has been believed that dynamic aspects like type-less declaration and dynamic typing plays an important role for maintaining quality. The type-less declaration saves testing time because we dont need to check type errors. It is also concluded from results that we dont need to specify test cases for checking type errors in dynamic languages. Dynamic languages are also helpful to test software developed in static languages.

From analysis of interviews and online survey, it is concluded that test driven development is important part for dynamic languages. Another interesting point is observed that most of the test frameworks for dynamic languages are developed from Smalltalk unit test framework i.e Sunit.

Furthermore, Junit for Java is also developed from same framework. It is concluded from the interviews and online survey results that there is no need to develop model for verification and validation. The rationale is test driven development, because it is believed that current testing frameworks are sufficient for dynamic languages. These frameworks have been successfully used in industry but the only need to use these frameworks with test driven development.

It is concluded that dynamic languages are being used in industry but they need to resolve performance issue and industry have to adopt test driven development while working with dynamic languages. Their popularity can be seen from advancement of making a common development environment for both static and dynamic languages. For example Microsoft is working on dynamic language runtime (DLR) to give support to some of dynamic languages like python in their .net framework. Another example can be observed from Sun Micro System, they are also working on common development environment for dynamic languages e.g IronPython.

9.2 Research Questions Revisited

RQ.1. What is the current state of the art in research on dynamic programming languages and their comparison with static languages?

The research in dynamic programming languages is ongoing and a brief analysis is given for
current studies in section 4.8. A detail description of these studies is given in sections 4.1 to 4.7. The dynamic languages are compared with static languages based on use, features and dynamic aspects. These differences are described in section 2.2.

**RQ.2 What are the aspects of dynamism for a dynamic programming language?**

The dynamic aspects play an important role in the popularity of dynamic languages. The most important dynamic aspects are dynamic typing, type-less declaration etc and a detail description of dynamic aspects is given in section 2.1.

**RQ.3 Does dynamic languages support or create hindrance to maintain the software quality?**

The dynamic languages don’t create any hindrance in maintaining the software quality. Most of the industry people think that dynamic languages help to improve the quality with their features and dynamic characteristics. More detail is given in section 6.2.2.1 and 7.2.

**RQ.4 What are the popularity factors necessary to raise the graph of dynamic languages?**

Dynamic Languages are commonly used in industry along with static languages. There are several factors found which increases/decreases the popularity of dynamic languages. Among these factors, performance is main issue which decreases the graph for dynamic languages. These factors are further described in section 6.2.2.4.

**RQ.5 What are the differences between verification and validation techniques and tools for static and dynamic languages?**

The verification and validation techniques are same for static and dynamic languages. The results reveals that there is no difference in techniques for static and dynamic languages. The same verification techniques such as inspection, reviews etc are used for dynamic languages. The details are described in sections 6.2.3, 7.2, 6.2.2.3 and 6.2.2. These sections also describes the verification and validation tools and practices found in industry. It is also extracted that unit testing framework for Java is also developed from Smalltalk unit testing framework i.e. SUnit.

**9.3 Future Work**

From the results of systematic review, there is a need to do more research to overcome the performance issue. These results will give a base line to researchers to identify the area in dynamic languages to improve performance issue. The researchers can give more focus towards libraries and debug support to further enhance the popularity of dynamic languages.

There is also need to have a comparison and differences of current available tools for dynamic languages. Due to the time constraint this thesis is not intended to give focus on this issue.
REFERENCES


[56] L. Tobias, S. Konstantinos, “Practical Type Inference Based on Success Typings”, PPDP’06, ACM, 2006


[142] H. Hubert, “Python’s role in the SRTM mission”,


Appendix - A Questionnaire for Online Survey

1. Personal Information
   - First Name
   - Last Name
   - Company Name (If applicable)
   - Email Address

2. What is your educational background? Post Doctorate
   - PhD
   - Post Graduate/ Master
   - Bachelor
   - Others (If applicable)

3. Describe your work experience (Area, Years)?

4. Which dynamic languages do you have experience with? Erlang
   - Lisp
   - Python
   - Ruby
   - Self
   - Smalltalk
   - Others

5. What is your level of experience with dynamic languages? Less than 1 year
   - 1-2 years
   - 2-4 years
   - 4-8 years
   - More than 8 years

6. Do you work professionally with dynamic programming languages?

7. What is the scope of V&V that you perform? Unit Testing
   - Integration Testing
   - System Testing
   - Acceptance Testing
   - Specify if other

8. Which characteristics do you think can raise popularity graph of Dynamic Languages?
   Agree Disagree Don't know
   - Programmer productivity
• Flexibility
• Code optimization
• Rapid software development
• Performance
• Modern Tools (IDEs, Testing tools etc)

9. Which aspects of dynamic languages are important for V&V?

• Dynamic Typing
• No Type Declaration
• Specify if other

14. As types are checked at runtime in dynamic languages, how do you test software for type safety and how much extra time/effort will it take?

15. Which tools and frameworks you have used for V&V of softwares developed in dynamic languages?

16. Is there any difference between the methodologies used for V&V (reviews, testing etc) in dynamic languages (Python, Ruby, Erlang, Lisp etc) and static languages (Java, VB.Net, C#)?

• Yes
• No
• If Yes, what are the differences?

17. Does dynamic languages support or create hindrance to maintain quality of developed software programs.

• If support, then how it improves quality?
• If create hindrance to maintain quality then how and what are the factors?

18. Do you face any specific problems/issues in testing of software developed in dynamic languages? If so, how do you handle them?

15. What do you think about current V&V methodologies and tools in support of testing dynamic languages?
Appendix-B Questionnaire for interviews

Base / Warm-up Questions:
1. What is your name?
2. What educational background do you have?
3. What is your current job description?
4. Briefly describe your work experience especially related to dynamic languages (e.g. years, area).
5. Which kind of Verification and Validation (V&V) activities have you been involved in?
6. Do you work professionally with dynamic programming languages? If so, in what way?

Verification and Validation

7. How is V&V implemented in your organization or personal experience?
8. What is the scope of V&V that you perform?
   a. Unit Testing
   b. Integration Testing
   c. System Testing
   d. Acceptance Testing
   e. Specify if other
9. Does dynamic languages support or create hindrance to maintain quality of developed software programs.
   a) If support, then how it improves the quality.
   b) If create hindrance to maintain the quality then how and what are the factors.
10. Which aspects of dynamic languages play an important role in V&V?
    a. Dynamic Typing
    b. No Type Declaration
    c. Specify if other
11. Which characteristics do you think can raise popularity graph of Dynamic Languages?
    a. Programmer productivity
    b. Flexibility
    c. Code optimization
    d. Rapid software development
    e. Performance
    f. Modern Tools (IDEs, Testing tools etc)
12. As types are checked at runtime in dynamic languages, how do you test software for type safety and how much extra time/effort will it take?

13. Which aspects of dynamic languages are challenges for you while testing?
   
   a. How do you tackle these challenges?

**Problems/Issues:**
14. Do you have any specific problems/issues in testing of software developed in dynamic languages, which affect the quality of product?

15. How do you handle these problems and issues?

**Techniques/Tools:**
16. Is there any difference between the methodologies used for V&V in dynamic languages (python, ruby, erlang, lisp etc) and static languages (java, VB.net, C #)?
   a) If yes, what are the differences?

17. Have your organization developed its own standard or framework for V&V? Do you use any open source technology for V&V?

**Summing up:**
18. What do you think about current V&V methodologies and tools in support of testing dynamic languages?

19. Which improvements will you suggest for V&V in dynamic languages?