Scientific Research Publishing

Journal of Computer and Communications, 2019, 7, 112-127
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

System Implementation Failures in the ERP Development Process

Samantha Mathara Arachchi*, Siong Choy Chong, Alik Kathabi

University of Colombo School of Computing (UCSC), Colombo, Sri Lanka
Email: saman17lk@yahoo.com, *ssp@ucsc.cmb.ac.lk, eddyscchong@yahoo.com, alik@msu.edu.my

How to cite this paper: Arachchi, S.M., Chong, S.C. and Kathabi, A. (2019) System Implementation Failures in the ERP Development Process. *Journal of Computer and Communications*, **7**, 112-127.
https://doi.org/10.4236/jcc.2019.712011

Received: November 28, 2019
Accepted: December 24, 2019
Published: December 27, 2019

Abstract

ERP mainly relates to business management software which consists of several modules that are important for the growth and survival of different types of businesses and organisations. The integration of related business applications in an organisation can bring significant advantages to the business by enabling faster supply chain management. ERP implementation enables firms to adapt and configure information flows and integrate business processes in order to enhance business performance. System implementation has been reviewed with adequate literature to significantly support the importance of arithmetic and logical concepts and basic errors in programming including typographical errors in a programme, identification of syntax errors and programme output process errors. It has further discussed the variables declaration, significance of modularization and necessity of supporting accessories along with the usability of library files and the importance of the use of frameworks. Enhancement of coding formatting and significance of coding syntax were also critically reviewed with sufficient literature to identify the system implementation failures in the ERP development process.

Keywords

System, Implementation, Programming, Failures, ERP, Factors, Framework

1. Introduction of System Implementation

ERP implementation is not an easy task for either the implementation team or the firm. It has to be carried out by a competent group of technical experts with a well-defined plan as it is a complex and expensive process to be carried out. Although it is a difficult task [1], most organisations are compelled to implement ERP systems as it can facilitate both productivity and efficiency of the current and future organisational business processes [2].

Selecting a programming language for implementation is a main task in the ERP context and ERP customisation does not have a lesser value attached to it. Therefore, this procedure has to be followed with much care and there are many facets to be considered, such as selecting a good programming language, designing methodology and using the correct tools [3]. When selecting a good programming language to implement ERP systems, there are many variables that need to be considered [4]. Considering the ERP capabilities and the rapid advancements of technology, high-performance computing, web services support and service-oriented architecture (SOA) are the key areas when evaluating a suitable programming language [5] [6].

2. Considered Factors in Implementing

There are eight subdomains in this component. They are, arithmetic and logical concepts (ALC), basic errors in system implementation (BESI), programme output process errors (POPE), variables declaration (VD), modularisation (MO), supporting accessories (SA), coding formatting (CF) and coding syntax (CS).

2.1. Arithmetic and Logical Concepts (ALC)

Algorithms are constructed using arithmetic and logical concepts. They are step-by-step finite sequences of instructions to solve a well-defined computational problem. Algorithms are used to solve any complex real-life problems visible in the industry when implementing an ERP system. Hence, it is very important to design the algorithm to solve the problem while writing and executing programmes to get the expected output. There are two approaches for algorithm design, namely the top-down and the bottom-up algorithm designs [7].

When automating business scenarios, there are logical concepts that need to be fulfilled to complete the flow of business. All these logics must be constructed accurately. If not, the ERP failure rate could be high, giving wrong or unrealistic outputs [7]. According to Donald's theory [8], some of the causes leading to high ERP failure rate are as follows: branching performed incorrectly, loop terminations not defined well-violated programming language rules violated programming standards and misinterpreted language constructs by the programmers. The recognition of arithmetic algorithms overcomes the shortcomings of some typical current approaches and introduces logical correlation matrices as a solution to the logic diagrams. In addition, a dynamic pruning policy that offers a specific instance is executed and also the complexity analyses which proves the validity and sophistication of the new arithmetic separately [9]. These concepts are important to avoid inadequate decision logic, arithmetic computations, and erroneous arithmetic computations.

2.2. Basic Errors in Programming (BESI)

In programming, different types of errors can occur. They are typographical errors such as syntax errors, data errors and programme output errors [10]. These errors are explained in the following sub-sections as follows: 1) Typographical

Errors in a Programme [11], 2) Syntax Errors [12], 3) Programme Output Process Errors [13] [14].

The following errors should be remembered and addressed, when necessary, to fix the bugs in the application: 1) identified indexing errors; 2) violated parameters or subscripts; 3) identified data errors; 4) identified non-terminating sub-programmes; 5) identified disk handling errors; 6) identified output processing errors; 7) identified iterative; 8) procedural errors; and 9) identified initialisation errors.

Document management is very important to the software industry and needs to be maintained well to identify the errors in programming. However, the techniques of document management are applicable with little modification across a wide variety of disciplines, including software. These documents should be modified along with design changes as quickly as possible. Formatting, checking spelling and grammar, organising content and flow of content and also maintaining templates are advisable to make retrieval easy. It should include memos, recipients, and details of senders as well [15]. At its simplest, document management has usually been considered to cover the techniques of creating and or acquiring, storing, locating and retrieving documents throughout their life cycle. However, as the use of computers has increased, documents have increasingly become available. Electronically, the remit of document management has evolved to a point where a "document" can be virtually any sort of computer file, a spreadsheet, a graphics file, a scanned image, a video clip, a voice-mail message and so on [15].

2.3. Variables Declaration (VD)

Variables play an important role in computer programming because they enable programmers to write flexible programmes. Rather than entering data directly into a programme, a programmer can use variables to represent the data. When the programme is executed, the variables are replaced with real data. This makes it possible for the same programme to process different sets of data [16]. Every variable has a name, designated as the variable name, and a data type. A variable data type indicates what sort of value the variable represents. The opposite of a variable is a constant. Constants are values that never change. Because of their inflexibility, constants are used less often than variables in programming [16].

2.4. Modularisation (MO)

Modularisation programming is a software design technique that emphasizes on separating the functionality of a programme into independent, interchangeable modules [17], especially when automating a complex business process like an ERP application. It maintains an easy and clear way to minimise the error rate of the application. In addition, it gives a clear overview of the application [18].

2.5. Supporting Accessories (SA)

Nowadays, in programming, there are supporting applications, objects, classes

and many more readymade sets of codes that can be incorporated with the existing codes. This is an advantage rather than constructing a programme from scratch. The usability of library files and the importance of use of frameworks are identified as separate types as described in the following sub-sections: 1) Usability of Library Files [19], 2) Use of Frameworks [20] [21] [22].

2.6. Coding Formatting (CF)

Coding formatting enhances the readability of lengthy programming and eases the identification of the errors or bugs when fixing bugs. Especially when control structures are used, it is important to format the code and align them properly by using tabs for indentation, without using both tabs and spaces because not all text editors treat tabs as exactly eight spaces and also leave only one blank line between subroutines to limit blank lines in the programme [23].

2.7. Coding Syntax (CS)

The programming language syntax, which forms part of a larger analysis of different programming languages, has been identified to be able to reduce the gap between programming languages [24]. Code syntaxes are the most powerful, which provides the grammar to write a programme. If the syntax is wrong or is misused, the programme will not execute properly and it will take two to three minutes to even days to recover, even if a colon or a semicolon is missing. It could become a major issue or mislead output [14]. To address this problem, the following rules are recommended to be adopted: 1) use shift operators instead of multiplication for constructing bit patterns; 2) always check for default case in switch statement; 3) use each variable for one purpose only; 4) use each field of structure for one purpose only; 5) avoid using global variables within outlines; 6) avoid using nonlocal variables within routines; 7) declare each variable in the smallest scope possible; 8) include all syntax errors to reduce the failure rate; and 9) correct errors promptly as they occur to keep the code simple [23] [25].

In addition, the data visualisation technique also helps to reduce the dimensions of data through the use of self-organising neutral networks. Manually generated programming codes as well as automated programming codes show similarities between them and are computed to get a generalised meaning of the syntax trees for the non-vectorial self-organising maps model [22].

3. Methodology

3.1. Research Framework

Figure 1 shows the research framework for this study. The proposed framework comprises two constructs: (1) ERP systems implementation (6) programming language.

The independent variables consist of ERP systems implementation. Systems implementation is the construction of the new system and the delivery of that system into output using a computer programme [26].

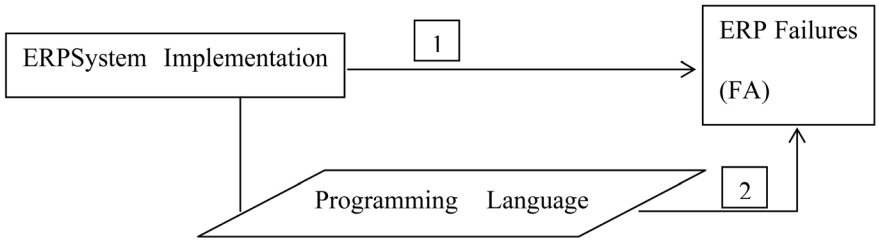

Figure 1. Research framework.

The mediating variable is a programming language that can be selected according to the type of application or convenience of the developer [22]. ERP failure rate represents the dependent variable.

3.2. Hypotheses Development

The proposed research framework broadly depicts the possible relationships connecting the two constructs. To substantiate these relationships, the literature was explored to discover the theoretical evidence upon which the hypothetical relationships connecting the constructs were built. These relationships have been planned as a set of research hypotheses addressing the research questions. The following two hypotheses were developed based on the research objectives and research framework:

H1: There is a significant relationship between system implementation and ERP failure.

H2: Programming languages significantly mediate the relationship between system implementation and ERP failure.

3.3. Research Method

A quantitative approach was adopted in this study. It is a normative survey approach and evaluation which is commonly used to explore the opinions of respondents who represent the whole population. The quantitative approach is more appropriate for this study since the objectives were to identify factors leading to ERP failure [27]. This method describes the nature of a condition as it takes place during the time of the study and explores the system or systems of a particular condition at each and every stage of the SDLC. In addition, the method is appropriate because it enables the researcher to generalise the findings obtained [28]. The constructs in the questionnaire were identified based on relevant literature, as well as the challenges and the concepts cited by respondents during the pre-survey. The questionnaires were self-administered.

3.4. Study Setting

The researcher has examined the ERP development companies under the Companies Registration Act of Sri Lanka and found that such development work is undertaken by such companies registered under the Board of Investment (BOI) and the Public Limited Companies (PLC) Act of Sri Lanka. However, the soft-

ware development companies registered under the PLC Act were not considered in this study due to their lack of investment capacity and involvement of ERP application development [29]. In addition, the PLCs do not have enough capital and resources as well as bank guarantee when signing vendor agreements. Hence, they are not engaged in developing total solutions for ERP applications. There are some companies that are registered under the BOI but have less than twenty employees, making little investments and having less capital. These companies were omitted as well.

3.5. Unit of Analysis

The unit of analysis consists of employees engaged in developing ERP applications at each and every stage of the software implementation in software development companies in Sri Lanka. The respondents were selected using a stratified sampling method by considering staff members with similar educational qualifications and working experience. Under this sampling method, each member of a population has an equal opportunity to become part of the sample based on the levels of developers prescribed. Since all of the employees have an equal chance of becoming research participants, this sampling method is said to be the most efficient sampling procedure [30]. A total of 48 software development companies registered under BOI qualified for this study. In the sampling strategy, the researcher obtained a list of all the employees and selected the sample accordingly software implementation.

3.6. Sample Design

According to the statistics of BOI, there were 66 companies registered for BOI software development projects [31]. Based on the justifications mentioned in the study setting, only 48 companies fulfilled the inclusion criterion. In all of the 48 companies, there were a total of 3640 employees. In the study, the representative sample was selected using a stratified sampling technique to select the employees followed by applying the random sampling approach to distribute the questionnaires. A stratum in this study is a subset of the population that shares at least one of the following common characteristic which is Software Implementers (SI)—everyone performing the same job role must have the same academic qualifications. The sample of 188 out of the 3640 employees has been used in the programming or developing the domain. The survey questionnaire is the research instrument consisting of a series of questions and other prompts for the purpose of gathering information from respondents and records their answers.

4. Data Analysis

4.1. Hypothesis 1: There Is a Significant Relationship between System Implementation and ERP Failure

The quality of system implementation is directly related to ERP failure. Accordingly, if system implementation is not being carried out appropriately, this could cause ERP failure and vice-versa. **Table 1** shows the results of the multiple

Table 1. Result of multiple regression for system implementation.

Independent Variables	Unstandardised Coefficients		Standardised Coefficients	T	Sig.	VIF*
	B	Std. Error	Beta			
(Constant)	0.341	0.144		2.364	0.019	
ALC	0.171	0.027	0.299	6.440	0.000	1.847
BESI	0.144	0.031	0.240	4.674	0.000	2.257
POPE	0.139	0.030	0.199	4.568	0.000	1.637
VD	0.052	0.023	0.086	2.215	0.028	1.282
MO	0.110	0.020	0.196	5.560	0.000	1.062
SA	0.067	0.025	0.121	2.693	0.008	1.733
CF	0.057	0.024	0.104	2.341	0.020	1.700
CS	0.161	0.026	0.219	6.287	0.000	1.042

VIF = Variance Inflation Factor.

regression analysis between system implementation (SI) as represented by its subdomains of ALC, BESI, POPE, VD, MO, SA, CF, CS and ERP failure (ERPF) according to the data collected.

Table 1 shows that the p-value for ALC, BESI, POPE, VD, MO, SA, CF and CS was less than 0.05. Hence, SI depends on ALC, BESI, POPE, VD, MO, SA, CF and CS. The R-square value was 0.799, which means that 79.9% of the variation in SI is explained by ALC, BESI, POPE, VD, MO, SA, CF and CS. The value of the VIF was less than 5 and hence, there is no problem of multicollinearity. In terms of residual diagnostics, the residuals were independent and normally distributed. The Kolmogorov-Smirnov test of normality on the residuals showed a p-value of 0.049, which is close to 0.05. Thus, the assumption of normality of the residual terms is met. The equation has been constructed as follows:

The Equation:

$$SI = 0.538\ (\text{Constant}) + 0.164\ (\text{ALC}) + 0.135\ (\text{BESI}) + 0.114\ (\text{POPE}) + 0.025\ (\text{VD}) + 0.155\ (\text{MO}) + 0.051\ (\text{SA}) + 0.077\ (\text{CF}) + 0.122\ (\text{CS}).$$

In stepwise regression, only income was significant. The R-square value was 0.999, which means 99.9% of the variation in SI is explained by ALC, BESI, POPE, VD, MO, SA, CF and CS.

The Equation:

$$SI = 0.004\ (\text{Constant}) + 0.147(\text{ALC}) + 0.206\ (\text{BESI}) + 0.146\ (\text{POPE}) + 0.083\ (\text{VD}) + 0.085\ (\text{MO}) + 0.061\ (\text{SA}) + 0.084\ (\text{CF}) + 0.186\ (\text{CS})$$

According to the above result, the hypothesis one has explained that there is a significant relationship between system implementation and ERP failure.

4.2. Hypothesis Testing 2: Programming Languages Significantly Mediates the Relationship between System Implementation and ERP Failure

The programming languages are posited to have a significant mediating effect on

the relationship between SI and ERPF. Hence, the mediating effect of PL between SI and ERPF is illustrated in Figure 2.

Table 2 presents the results of the multiple regression analysis on the relationship between SI and ERPF as mediated by PL, while Table 3 presents the results of PL upon SI.

The score of the Sobel test was 2.4683904 with a significant p-value is 0.0135722, which is less than 0.05. The finding implies that PL is a significant mediator in the relationship between SI and ERPF as shown in Figure 3.

$$z = \frac{ab}{\sqrt{\left(b^2 SE_a^2\right) + \left(a^2 SE_b^2\right)}}$$

Based on the above formula, a is the regression coefficient for the relationship between the independent variable and the mediator, b is the regression coefficient for the relationship between the mediator and the dependent variable, SE_a is the standard error of the relationship between the independent variable and

Table 2. Result of multiple regression analysis between system implementation and ERP failure as mediated by programming languages.

Coefficients[a]						
Model		Unstandardised Coefficients		Standardised Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	−0.003	0.007		−0.397	0.692
	SI	0.702	0.003	0.999	359.678	0.000
	PL	0.182	0. 056	0.001	0.299	0.765

[a]Dependent Variable: ERPF.

Table 3. Result of multiple regression of programming language upon system implementation.

Coefficients[a]								
Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Collinearity Statistics	
		B	Std. Error	Beta			Tolerance	VIF*
1	(Constant)	0.688	0.182		3.773	0.000		
	PL	0.702	0.185	0.776	16.125	0.000	1.000	1.000

[a]Dependent Variable: PL.

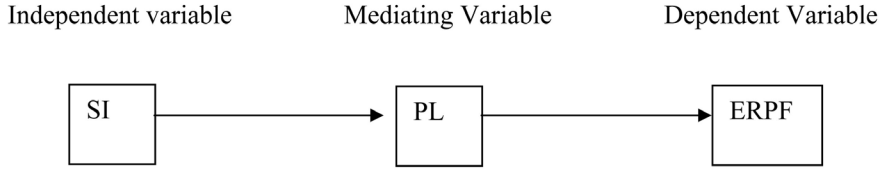

Figure 2. Mediating effect of advanced programming language on system implementation and ERP failure.

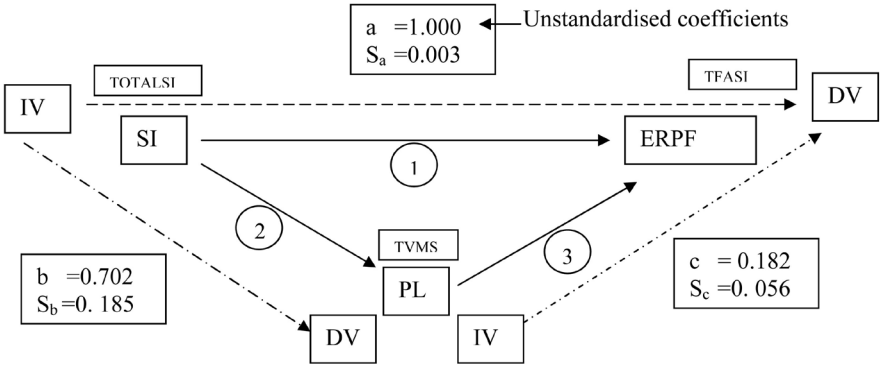

Figure 3. Mediating effect with coefficient values of programming languages between system implementation and ERP failure.

the mediator, while SE_b is the standard error of the relationship between the mediating variable and the dependent variable. The data was applied as follows:

1) (UnStandardised Beta) $a = 1.000$; $S_a = 0.003$.

2) $b = 0.702$; $S_b = 0.185$.

3) $c = 0.182$; $S_c = 0.056$.

$$\text{Indirect Effect (IE)} = b \times c = 0.702 \times 0.182 = 0.12776$$

$$\text{Variance in IE} = (b \times S_b)^2 + (c \times S_c)^2 = (0.702 \times 0.185)^2 + (0.182 \times 0.056)^2 = 0.016866 + 0.000103 = 0.0169698$$

$$SE \text{ in IE } \sqrt{0.0169698} = 0.1302$$

$$Z = 0.12776/0.1302 = 0.98112$$

$$\text{The p-value} = P\,[Z > 0.98112] < 0.001 < 0.05$$

Thus the indirect effect is significant.

Using online Sobel application:

Sobel test statistic: 2.4683904.

One-tailed probability: 0.006786.

Two-tailed probability: 0.0135722, this is less than 0.05 and therefore PL is a significant mediating factor.

Since the p-value is less than 0.05, PL mediates the relationship between SI and ERPF. H6 is accepted.

4.3. Findings and Discussion—Hypothesis One

The first hypothesis was to determine whether there is a relationship between system implementation and ERP failure. It posited that if the system implementation does not do a proper job, this could be a case of failure to the ERP application and vice-versa.

Variables representing system implementation such as arithmetic and logical concepts (ALC), basic errors in system implementation (BESI), programme output process errors (POPE), variables declaration (VD), modularisation (MO), supporting accessories (SA), coding formatting (CF) and coding syntax (CS) are points of variance that are close to each other in their distribution.

The mean score for system implementation of ERP applications was 3.70 with

a standard deviation of 0.3792. The maximum and minimum scores were 4.58 and 2.83, respectively. The median was 3.80, which indicates that at least 50% of the system developers graded more than 3.80. Thus, the most frequent rating amongst the system developers was 4.00. This shows that the factors identified in the system implementation questionnaire for ERP system design are very important. The constructed framework is shown in **Figure 4**.

The eight clusters shown in **Figure 4** include 45 items to represent the relationship between system implementation and ERP Failure, which were confirmed by the tested model of the research. The established testing environment was supposed to fulfill the requirement of all items based on the above eight clusters in the system implementation stage. This demonstrates the high diversity of ERP implementation.

The correlation coefficient for items for each factor, such as ALC, BESI, POPE, VD, MO, SA, CF, and CS, was represented by p-values as low as 0.028, which are less than 0.05. Thus, the items representing system implementation are significant predictors of ERP failures.

For every unit increase in ALC, ERP failure is expected to drop by 0.171. For every unit increase in BESI, ERP failure is expected to drop by 0.144, and for every unit increase in POPE, ERP failure is expected to drop by 0.139. For every unit increase in VD, ERP failure is expected to drop by 0.052. Furthermore, for every unit increase in MO, ERP failure is expected to drop by 0.110, and for every unit increase in SA, ERP failure is expected to drop by 0.067. In addition, for every unit increase in CF, ERP failure is expected to drop by 0.057, and for every unit increase in CS, ERP failure is expected to drop by 0.161.

According to the regression analysis, ALC, BESI, POPE, VD, MO, SA, CF, and CS are the significant predictors that support the objective, *i.e.* there is a significant relationship between system implementation and ERP failure, and the relationship is in an inverse direction. This implies that the more these factors are being practiced, the lower would be the ERP failure rate.

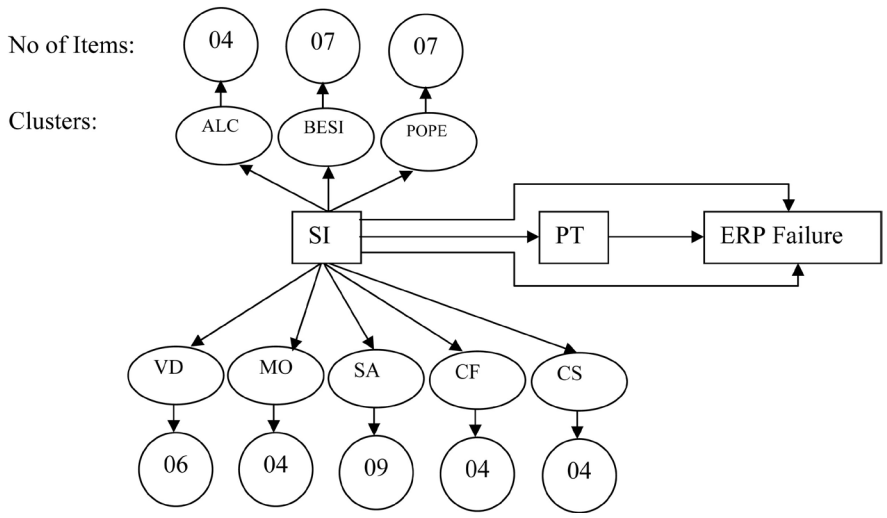

Figure 4. Conceptual framework for H5.

The VIF values for ALC, BESI, POPE, VD, MO, SA, CF and CS are 1.847, 2.257, 1.637, 1.282, 1.062, 1.733, 1.700 and 1.042, respectively. Since they are below 5, there is no problem with multicollinearity among these factors.

The findings show that arithmetic and logical concepts should take into consideration that inadequate decision logic, arithmetic computations, branching performed incorrectly, loop terminations undefined, violated programming language rules and standards, as well as misinterpreted language constructs by the programmers, are the main causes that directly affect ERP failure [7].

In addition, this study has also confirmed that basic errors in system implementation include typographical, syntax, indexing, data, disk handling, output processing, iterative procedural, initialisation and violated parameters or subscripts, as well as non-terminating sub-programmes are factors that increase the ERP failure rate [10]. Donald [14] also found that programme output process errors play the main role in order to reduce the ERP failure rate. These include errors such as input-output format, main storage allocation, identified software interface and identified erroneous "error message" processing, database interface errors, user interface errors [13] and compiler errors.

According to Andrei [11] and [16], the necessity of the declaration of a variable is very important. The variable type and dimensions should not be incorrectly declared while unique names for variables and standard naming methods for library files should be meaningful. In addition to this modularization, precautions such as including codes within the main() routine, limiting the number of lines in a routine to 50 or less, having subroutines or loops without duplication of codes are also supposed to reduce ERP failure.

Then the value of the following items formatting programming code such as tabs and spaces for indentation, a consistent indentation pattern for a programme's control structure, and limiting blank lines in programmes, which were identified by Chris [23] and was also confirmed by this research. In addition, the researcher also found that coding syntax with the use of shift operators instead of multiplication for constructing bit patterns, switch statement, check for default case, using each variable for exactly one purpose, avoiding using global variables within routines and nonlocal variables within routines, declaring each variable in the smallest scope possible, correcting errors promptly as they occur, as well as keeping code simple were important to enhance the success of the ERP system [12].

The first hypothesis of this research has been examined through H5 which includes eight clusters along with the 48 items. The findings showed that there is a relationship between system implementation and ERP failure. If a proper system implementation is not carried out, then the ERP failure rate will be high. Therefore, to reduce the failure rate, system implementation has to be carried out according to the research framework proposed in this study.

There are consistent findings as well for this phase of system implementation. According to [32], ERP implementation is the ability of the firm to adapt and

configure information flows and integrate business processes in order to enhance business performance. Considering the ERP back capabilities and the rapid advancements of technology, high-performance computing, web services support, and service-oriented architecture will be the key areas when evaluating a suitable programming language to implement an ERP [5] [6]. The method that generates the syntax tree and uses the feature tree to match the knowledge in the syntax tree to identify code knowledge automatically is being widely followed [12]. The experimental results show that the system can effectively and accurately gather the statistic of knowledge in the programme code in real-time.

If the input programme covers the whole range of the language syntax constructs, then the parser corresponding to the generated annotated grammar is able to parse and transform into an Abstract syntax tree (AST) any programme of the given language [11]. It is important to use code review correctly. Formatting, spell check and grammar, organising content and flow of content and maintaining templates are advisable. In order to retrieve them, it should include memos and the details of recipients and senders as well [15].

The results also showed that misusing or improper use of arithmetic and logical concepts, including inadequate decision logic, arithmetic computations, branching performed incorrectly, loop terminations undefined, violated programming language rules and, standards, misinterpreted language constructs by the programmer are the main causes that directly affect the ERP failure [7]. Zhang [9] has pointed out similar results. In this research, more factors have been considered together to determine their impact. That is, again, a major significant contribution to this research.

Basic errors in system implementation which include typographical [11], the syntax [12], indexing, data, disk handling, iterative procedural, initialisation and violated parameters or subscripts and also non-terminating sub-programmes are factors that increase the ERP failure rate. Widera [10] has identified the same factors which are needed to enhance the programme. All these scholars have separately explained how to write a programme with the minimum of those errors.

Consistent with Donald [14], this research found that programme output process errors play the main role in order to reduce the failure rate. This includes the following errors such as input-output format, main storage allocation, identified software interface and identified erroneous "error message" processing, database interface errors, user interface errors [13] and compiler errors.

According to [11] and [16], a variable declaration with the variable type and dimensions incorrectly declared, unique names for variables used meaningfully and standard naming methods for library files are also imperative. In addition to this modularisation, routine, limit number of lines in a routine to 50 or less, subroutines or loops without duplicate codes also contribute to reducing the ERP failure.

Chris [23] explains that coding, formatting with the tabs and spaces for indentation, programme's control structure to follow consistent indentation pat-

tern, and limiting blank lines in the programme were also proved again in this research with regard to all the above factors.

The researcher found that coding syntax with the use of shift operators instead of multiplication for constructing bit patterns, switching statement, checking for default case, using each variable for exactly one purpose, avoiding using global variables within routines and nonlocal variables within routines, declaring each variable in the smallest scope possible, correcting errors promptly, as they occur and finally keeping code simple can enhance the success of the system [12].

4.4. Findings and Discussion—Hypothesis Two

The 2nd hypothesis was to determine whether programming languages (PL) mediate the relationship between system implementation and ERP failure. Sobel's test results showed the value of 2.4683904 with a significant p-value is 0.0135722. This implies that PL is a significant mediator between system implementation (SI) and ERP failure (ERPF).

The findings are in line with the literature based on the two key points supporting PL as a significant mediator. First, modularisation [17], especially when automating a complex business process like an ERP application It maintains an easy and clear way to minimise the error rate of the application. In addition, it gives a clear overview of the application [18].

On the other hand, Al-Hossan [33] has also identified that limited access to supporting accessories like library files, frameworks, objects, and modules also increase the ERP failure rate among the software development companies.

In the 2nd hypothsis, modularisation which aligns with the codes within the main() routine, limiting the number of lines in a routine or without duplication codes influencing the relation between system implementation using different programming languages is important to reduce ERP failure [25]. In addition, awareness of supporting accessories such as library files, frameworks, and objectives are also important, while Zimin [17] and [33] explained similar facts with the limiting factors discussed under the literature review.

5. Conclusions and Recommendation

Another interesting recommendation is derived from the perspective of system implementation. The logic programming is a type of programming paradigm which is largely based on formal logic. It expresses facts and rules about some problem domains in the business process. Therefore, the handling of arithmetic and logical concepts is the most important consideration when automating ERP application.

Knowing the whole range of the language syntax helps to construct an error-free programme. It was explained that it is important to use code review correctly. Formatting, spell check and grammar, organising content and flow of content and maintaining templates are advisable.

To minimise that the basic errors it is necessary to reduce typographical errors

for developing abstract syntax trees, syntax errors, indexing, data, disk handling, iterative procedural, initialisation and violated parameters or subscripts and also non-terminating sub-programmes to fix the bugs in the application.

This researcher has also found such errors in input-output format to get the expected results, main storage allocation to run the application smoothly with enough ram and rom and for store data and retrieval, identified software interface and erroneous "error message" processing for enhancing the user-friendliness, database to store enough data physically or in cloud virtually, reduce the compiler and user interface errors and are highly recommend to produce more efficient and productive error-free code.

According to the finding of this research also explained the necessity of variables declaration with the variable type and dimensions to represent data correctly. Unique names for variables should be used meaningfully. Standard naming methods for library files are necessary to structure the programme properly and reduce the conflict with system files. In addition to that, the modularization is also important to separate the functionality of a programme into independent, interchangeable modules to make it faster and improve the performance.

The researcher has explained that coding formatting with the tabs and spaces for indentation enhances the readability of the coding when referring them again for correction or modification. The Programme's control structure recommends following consistent indentation patterns and limiting blank lines in programs. As a whole, these were also recommended in order to reduce the programme implementation errors.

Furthermore, the researcher, found that coding syntax has also explained and recommended the use of shift operators instead of multiplication for constructing bit patterns, switch statement, using each variable for exactly one purpose, avoid using global variables within routines, declaring each variable on the smallest scope possible and correcting errors promptly as they occur, and finally keeping code simple was factor which enhances the success of the system. The syntax is language-specific and differs either a little or a lot depending on a language. They are important only in the scope of a particular language. There are some languages that compile to another base language, and they usually have a different syntax. Changing the syntax of a language can be relatively easy or hard, depending on what the syntax should be. Therefore, it is recommended to know the language accurately in order to save time and budget.

Selecting a proper and programming language is also very important because day by day technology changes, thereby creating new updates as well as plug-ins and supporting objects. It is important, recommended and necessary to keep in touch to decide when to quit the existing application and switch to the new generation.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Samuel, A.A., Reddy, B.S. and Nair, J. (2014) Conceptualizing Dimensions of Enterprise Resource Planning Systems Success: A Socio Technical Perspective. *International Journal of Enterprise Information Systems*, **10**, 53-75.

[2] Shivkumar, S. and Hasmukhrai, T. (2012) Software Testing Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, **7**, 16.

[3] Howden, W. (2006) Reliability of the Path Analysis Testing Strategy. *IEEE Transactions on Software Engineering*, **SE-2**, 208-215. https://doi.org/10.1109/TSE.1976.233816

[4] Chaim, M., Maldonado, J. and Jino, M. (2003) A Debugging Strategy Based on Requirements of Testing. *Seventh European Conference on Software Maintenance and Reengineering*, Benevento, Italy, 28 March 2003, 160-169.

[5] Galin, D. (2004) Software Quality Assurance. Pearson Education Limited, Harlow.

[6] Lewis, W.E. (2005) Software Testing and Continuous Quality Improvement. Auerbeach Publications, New York.

[7] Vinu, V.D. (2006) Principles of Data Structures Using C and C++. New Age International, Telangana, India.

[8] Donald, E.K. (1995) The Art of Computer Programming. Vol. 3, Odd Bookworm, Hackensack, NJ.

[9] Zhang, L., Yuan, S., Tang, J. and Xie, X. (2008) Research on the Composite Arithmetic of Logic Compound Sentences in Decompilation. 2008 *International Symposium on Computer Science and Computational Technology*, Shanghai, 20-22 December 2008, 442-446.

[10] Widera, M. (2011) Why Testing Matters in Functional Programming. Fern University at in Hagen, Hagen.

[11] Andrei, A. and Daniel, I.V. (2012) Automating Abstract Syntax Tree Construction for. 14*th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania, 26-29 September 2012, 152-159.

[12] Gayle, L.M. (2011) Cracking the Coding Interview: 150 Programming Questions and Solutions. 5th Edition, CareerCup.

[13] Carlton, R. (2017) Three Dangers of a Poorly-Designed ERP User Interface. Converted Media.

[14] Knuth, D.E. (2007) The Art of Computer Programming. Volume 1, 3rd, Edition, Dorling Kindersley, Delhi.

[15] Doverton, D. (2001) Techniques of Document Management: A Review of Text Retrieval and Related Technologies. *Journal of Documentation*, **57**, 192-217. https://doi.org/10.1108/EUM0000000007082

[16] Vangie, B. (2015) variable.html. http://www.webopedia.com/TERM/V/variable.html

[17] Jin, Z.M., Fu, Q., Jin, J. and Tao, J.W. (2013) Characteristics and Module Design of Weaving ERP. 3*rd International Conference on Information Management, Innovation Management and Industrial Engineering*, Kunming, 26-28 November 2010, 422-425.

[18] Kenneth, E. (2011) Modularization. Aalborg University, Copenhagen.

[19] Georges, E.K. (2009) Building a Service-Oriented ERP from an Open Source Software. *Fourth International Conference on Software Engineering Advances*, Porto, Portugal, 20-25 September 2009, 33-38.

[20] Leopoulos, V., Kirytopoulos, K. and Voulgaridou, D. (2005) ERP Systems as a Component of the Electronic Supply Chain: Classification of Implementation Risks. *International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on intelligent Agents, Web Technologies and Internet Commerce*, Vienna, Austria, 28-30 November 2005, 676-682.

[21] Song, H., Huiyou, C. and Qing, W. (2009) Component Library-Based ERP Software Development Methodology. 2009 *International Conference on Interoperability for Enterprise Software and Applications China*, Beijing, 21-22 April 2009, 34-38.

[22] Zhu, Z.Y. and Dai, S.H. (2009) J2EE-Based Enterprise ERP System Design and Implementation. 2009 2*nd IEEE International Conference on Computer Science and Information Technology*, Beijing, 8-11 August 2009, 509-512.

[23] Chris, P. (2009) Learn to Program. 2nd Edition, Pragmatic Bookshelf.

[24] Andreas, S. (2013) An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education*, **13**, Article No. 19. https://doi.org/10.1145/2534973

[25] Samantha, M.A, Chong, S.C. and Kennedy, D.G. (2014) A Comparison between Evaluation of Computer Based Testing and Paper Based Testing for Subjects in Computer Programming. *International Journal of Software Engineering and Applications*, **5**, 57-72. https://doi.org/10.5121/ijsea.2014.5105

[26] MITRE (2013) System Design and Development. MITRE Corporation, Bedford, MA.

[27] Creswell, J. (1994) Research Design: Qualitative and Quantitative Approaches. Sage Publications, New York.

[28] George, D. and Mallery, P. (2013) IBM SPSS Statistics 21 Step by Step: A Simple Guide and Reference. 13th Ed, Pearson, Upper Saddle River, NJ.

[29] Central Bank of Sri Lanka (2014) Annual Report 2014.

[30] Sekaran, U. and Bougie, R. (2013) Research Methods for Business A Skill-Building Approach. 6th Edition, Wiley, New York.

[31] Board of Investment (2012) BOI. http://www.investsrilanka.com/

[32] Srinivasan, D.D. and Gopalaswamy, R. (2006) Software Testing: Principles and Practice. Pearson Education, India.

[33] Al-Hossan, A. and Al-Mudimigh, A.S. (2011) Practical Guidelines for Successful ERP Testing. *Theoretical and Applied Information Technology*, **27**, 11-18.