Mathematical Method and Tool for IT System Complexity Reduction
Ensuring Successful Delivery of Complex Software Projects

Adele VORSTER¹ and Josef J. LANGERMAN²
University of Johannesburg, South Africa

Keywords: IT system complexity, Project failure, Complexity reduction, Minimum viable product (MVP), Micro-services, Measure complexity.

Abstract. This study highlights the concerning high project failure rate even when the best software development methodologies and project management methods are used. Information technology systems are classified as some of the most complex artefacts that mankind produces [2]. This study aims to (1) identify a mechanism that can be used to measure software complexity; (2) analyse the correlation between complexity and the high project failure rates; and (3) conclude with a mechanism that can be used to reduce complexity. In-depth mixed method case studies were conducted. Quantitative data will be collected through a tool by measuring the system complexity. Qualitative data will be collected through interviews. The results indicate that there is a correlation between project failure and high system complexity measurements. This research proposes guidelines on how to reduce system complexity in order to increase the chances of successful project delivery.

Introduction

The Standish Group Research CHAOS Reports [1] over the past two decades have shown alarmingly high project failure rates. Figure 1 show the concerning high failure rate. An insignificantly small increase in successful delivery of software projects can be seen in 2001 after the introduction of agile software development methodologies. An investigation into the high failure rates and possible remediation is critical when considering the vast amount of financial losses incurred through failed projects.

Information technology (IT) systems are classified under the most complex creations of mankind. IT Systems have similar but never identical parts and is comparable to that of a fractal structure where patterns emerge throughout the structure [2].

Problems can be categorized as Simple, Complicated and Complex. With a simple problem, the pre-identified steps can be followed and the outcome is predictable. Complicated problems are exceedingly difficult for an untrained person or non-professional to comprehend. A trained person or professional applying appropriate formulas and methods the outcome can be achieved with a high level of certainty. Every situation of a complex problem is unique. Following steps from a completed complex problem does not guarantee the successful outcome of another complex problem [3].

You cannot control what you cannot measure [4]. It is thus crucial to be able to measure the complexity of an IT system in order to reduce the complexity.
This research paper investigates complexity reduction mechanisms to increase the probability of successful delivery of an IT software system and project.

Complexity measurement methods will be discussed. Software complexity measurement tools will be identified and discussed. The formula and tool needs to work in conjunction with each other to be able to measure the software’s complexity. A literature review will be done on a mechanism that can be utilised to reduce a software system’s complexity. The results obtained from the in-depth case studies will be summarised and the analysis presented.

To ensure successful delivery of a complex IT system the complexity needs to be reduced. This research study will conclude with guidelines and a method to measure and reduce software complexity.

**Complexity Measurement Methods**

The known and main complexity measurement formulas are McCabe’s Cyclomatic Complexity and Halstead Complexity [5]. A third method to measure system complexity will also be discussed.

**Cyclomatic Complexity**

McCabe’s Cyclomatic complexity [6] is based on graph theory and other mathematical definitions. It measures the possible independent paths through the code of a software system. Complexity v of graph G can be determined with the following formula:

\[
v(G) = e - n + 2p
\]

where:
- \( e \) = the number of edges in the graph
- \( n \) = number of notes in the graph
- \( p \) = number of connected components

**Halstead Complexity**

Complexity is measured through the operators in the IT system. Halstead’s difficulty measure can be determined with the following formula [7]:

\[
Difficulty = L / V
\]

where:
- \( L \) = Program level
- \( V \) = Program volume

The formula to determine program level \( L \) is:

\[
L = (2 + n2^*) \times \log_2 (2 + n2^*)
\]

where:
- \( n1 \) = number of unique operators
- \( n2 \) = number of unique operands

The formula to determine Program volume \( V \) is:

\[
V = (N1 + N2) \times \log_2 (n1 + n2)
\]

where:
- \( N1 \) = total number of operator occurrences
- \( N2 \) = total number of operand occurrences

**Session’s First Corollary of Software Complexity**

The relative complexity of two software systems A and B are equal to the ratio of the number of states in system A divided by the number of states in system B. Complexity of a system equals the number of states of a system raised to the power of the number of variables of the system. Thus, the complexity measure \( C \) is [5]:

\[
C = S^V
\]

where:
- \( S \) = number of states
- \( V \) = number of variables
Reducing Complexity

Measuring a system’s complexity is directly related to the reduction of the system’s complexity [5]. The functionality contained within the IT system and the coordination between its sub-systems directly affects the system’s complexity. Decreasing the functionality within a system will result in a higher coordination between the sub-systems as the excess functionality will be moved to another sub-system. Reducing the coordination between sub-systems will result in more functionality in each of the sub-systems, which causes higher functional complexity. A perfect balance between the functionality and coordination within an IT system is required to achieve reduced complexity. This balance can be determined by Simple Iterative Partitioning (SIP) [5].

Simple Iterative Partitioning

SIP is based on a synergistic partitioning methodology. It can be utilised to mathematically determine and optimize the least possible complexity of an IT system. The SIP methodology deconstructs a system into its sub-systems [5].

1) Partitioning
   Partitioning is defined by dividing a system into its sub-systems. Partitions should adhere to the five laws; (1) Partitions should be valid in terms of partitions as defined by set theory. (2) Partitions created should be logically related to the problem being addressed. (3) The amount of partitions should be in line with the problem being addressed. (4) Each partition should be more or less equal in size. (5) Interactions between different partitions should be minimal.

2) Simplification
   A system’s complexity can be further reduced by the removal of items from sub-systems or the removal of complete sub-systems.

3) Iteration
   Iteration reduces complexity by implementing each sub-system one at a time and iterating through the set rather than implementing the entire group at once. It is better to implement a complex system iteratively according to Boyd’s Law of Iteration.

Complexity Reduction through Partitioning

Using (5) a system with two variables and six states has a complexity measurement of:

\[ C = S^V \]
\[ = 6^{12} \]
\[ = 2,176,782,336 \]
\[ > 2 \text{ billion possible paths} \]

Applying SIP to partition the system into two sub-systems that each contains half the variables and the same amount of states, the complexity by using (5) becomes:

\[ C = S^V \]
\[ = 6^6 + 6^6 \]
\[ = 2(6^6) \]
\[ = 93312 \text{ possible paths} \]

This equates to a 99.996% decrease in complexity.

Complexity Measurement Tools

Various tools are available to measure the complexity of an IT system that is written in line with object oriented principles (OOP). The following were found through the search engine google.com: (1) ProjectCodeMeter (2) NDepend (3) Code Metrics (4) SourceMonitor (5) Nitriq

ProjectCodeMeter

IT system complexity can be measured and analyzed with ProjectCodeMeter. Other measurements that can be obtained through a code analysis include (1) the working hours spent developing the software, (2) cost of development, (3) productivity of the development team and (4) the maintainability of the source code.
NDepend
Visualising dependencies and associations between the code components are the main objectives of NDepend. Various software metrics such as cyclomatic complexity, lines of code (LOC), coupling and cohesion can be obtained. NDepend can be integrated into the .Net Visual Studio integrated development environment (IDE) as an add-in tool.

SourceMonitor
Analysis can be performed on IT systems developed in a range of languages such as C++, C#, C, VB.Net, Java, Delphi, VB6 and HTML. Module complexity, software size and possible software defects are presented after code analysis.

Measuring Complexity of an IT System
None of the available code analysis tools provide the ability to measure a system’s complexity against the formula presented in SIP. A derived formula can be determined from Session’s First Corollary of Software Complexity [5] and McCabe’s Cyclomatic Complexity formula [6].

McCabe Cyclomatic Complexity:

\[ M = e - n + 2p \]  \hspace{1cm} (6)

where:  
- \( e \) = number of edges
- \( n \) = number of states
- \( p \) = number of connected components

The number of total linear paths through the system can be determined by McCabe’s Cyclomatic Complexity. The total number of possible states of a system corresponds to the total number of linear paths through a system.

The revised formula for calculating complexity thus becomes:

\[ C = S^V \]

\[ = (e - n + 2p)^V \]  \hspace{1cm} (7)

where:  
- \( e \) = number of edges
- \( n \) = number of nodes
- \( p \) = number of connected components
- \( V \) = number of variables

NDepend provides all the measurements needed to calculate a system’s complexity with the derived formula.

Research Method
Four mixed method case studies were conducted in the research project. Quantitative data was collected through the NDepend source code analysis tool and the metrics were applied to the derived formula in (7). Qualitative data was collected through interviews. The qualitative and quantitative data were analysed to determine if there is a correlation between project failure rates and IT systems with high complexity measurements.
Results

Table 1. Case study results.

<table>
<thead>
<tr>
<th>Case Study</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>66,233</td>
<td>105,990</td>
<td>173,459</td>
<td>268,515</td>
</tr>
<tr>
<td>Net project’s lines of code</td>
<td>15,750</td>
<td>16,354</td>
<td>19,956</td>
<td>41,061</td>
</tr>
<tr>
<td>Types</td>
<td>659</td>
<td>782</td>
<td>2,774</td>
<td>1,608</td>
</tr>
<tr>
<td>Assemblies</td>
<td>21</td>
<td>11</td>
<td>22</td>
<td>44</td>
</tr>
<tr>
<td>Namespaces</td>
<td>74</td>
<td>65</td>
<td>17</td>
<td>132</td>
</tr>
<tr>
<td>Methods</td>
<td>4082</td>
<td>5387</td>
<td>27,066</td>
<td>135,926</td>
</tr>
<tr>
<td>Fields</td>
<td>1225</td>
<td>1208</td>
<td>649</td>
<td>1927</td>
</tr>
<tr>
<td>Max method complexity</td>
<td>16</td>
<td>8</td>
<td>1080</td>
<td>92</td>
</tr>
<tr>
<td>Average method complexity</td>
<td>1.52</td>
<td>1.62</td>
<td>2.17</td>
<td>1.62</td>
</tr>
<tr>
<td>Edges</td>
<td>120</td>
<td>74</td>
<td>109</td>
<td>118</td>
</tr>
<tr>
<td>Nodes</td>
<td>41</td>
<td>27</td>
<td>51</td>
<td>40</td>
</tr>
</tbody>
</table>

Quantitative data

<table>
<thead>
<tr>
<th>Calculated complexity values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected components</td>
</tr>
<tr>
<td>McCabe’s Cyclomatic Complexity</td>
</tr>
<tr>
<td>Number variables</td>
</tr>
<tr>
<td>Number states</td>
</tr>
<tr>
<td>C = 8N^2 + 25^2</td>
</tr>
</tbody>
</table>

Qualitative data

<table>
<thead>
<tr>
<th>Project status</th>
<th>Failure</th>
<th>Failure</th>
<th>Success</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of resources</td>
<td>4</td>
<td>29</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>Project status reason</td>
<td>Time over-run</td>
<td>Low quality software delivered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-deployment bugs</td>
<td>na</td>
<td>774</td>
<td>9</td>
<td>193</td>
</tr>
<tr>
<td>Post-deployment bugs</td>
<td>10</td>
<td>100</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>User satisfaction and value delivered</td>
<td>Yes</td>
<td>Above average</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Development methodology</td>
<td>None</td>
<td>Hybrid</td>
<td>Hybrid</td>
<td>Hybrid</td>
</tr>
</tbody>
</table>

Analysis

The following observations were made after analyzing the qualitative and quantitative data:

- **Complexity**
  A clear correlation was observable between the projects with a failed status and high complexity measurements.

- **Number of identified bugs**
  Projects with a high amount of bugs identified due to their high systems complexity resulted in the IT system being delivered with low quality and a failed delivery status.

- **Number of project resources**
  It was observed that projects with a very high number of resources allocated to it resulted in a failed project status. Further investigation into the optimal number of resources allocated to a project was required.

- **Complexity vs. software system size**
  No correlation between the software system’s complexity and size of the software system was observable. Larger projects did not have higher complexity measurements due to their size.
Key Findings

Complexity of an IT system has a drastic effect on the successful delivery of the project. Reducing the complexity during the development of the IT system is thus crucial. The following guidelines were presented to reduce complexity during development and to increase the probability of successfully delivering the project:

- **Delivery Team**  
  Communication issues arise in large teams due to the number of communication links between the members of the team [8].  
  The communication links can be determined with the following formula:  
  \[ L = \frac{n(n-1)}{2} \]  
  where \( n \) = number of team members.  
  Keeping the number of communication links optimal will reduce the complexity due to team member corroboration.

- **Minimum Viable Product**  
  Focus on the core required functionality of an IT system can be achieved by delivering a minimum viable product (MVP) [9].  
  Delivering the MVP first ensures that the team focuses on the core required functionality. This in essence reduces complexity relating to the functionality of the IT system. Glass’s Law [10] states that the complexity of a system doubles with every 25% increase in functionality.

- **Simple Iterative Partitioning and Micro-services architecture**  
  Micro-services architecture will ensure that the complexity that rises from the development and delivery of a monolithic application can be reduced [11]. SIP can be utilised to divide the IT system into its set of micro-services. The optimal amount of functionality and coordination between the micro-services can be determined through the SIP methodology [5] and formulas.

- **Development methodology**  
  Following an iterative development methodology such as Agile will ensure that the IT system is delivered with the highest possible obtainable value.

Conclusion

Traditional software development methodologies are not adequate to ensure the successful delivery of an IT system. After an industry wide alignment with agile development methodologies to compensate for the shortcomings of traditional software development methodologies, the project failure rates were still exceptionally high.

With IT systems being complex it is clear that there was no silver bullet solution to their successful delivery. It is thus crucial that the complexity of an IT system be reduced during the software development and project delivery phase.

A revised formula (7) to measure software complexity was presented and a tool to use in conjunction with this formula was identified.

The following guidelines to reduce complexity were discussed in more detail in this paper in the key findings section:

- **Delivery team assembly with the optimal amount of team members to reduce the coordination complexity that arise from a high number of communication links.**
- **Focus on delivering the minimum viable product (MVP) that contains only the important requirements and functionality.**
- **Utilise a micro-service architecture to reduce unnecessary technical complexity. The system can be divided into its sub-systems and micro-services through simple iterative partitioning (SIP).**
- **A development methodology that follows an iterative approach will assist in reducing complexity throughout the systems development.**

References


