**Assignment 5**

This paper provides an overview of the complexity metrics that have been developed to determine the complexity of various software modeling techniques. While the paper uses the Unified Modelling Language (UML) as an example to demonstrate different points about software modeling complexity, similar metrics can be extended to other modeling and programming languages as well.

**THE SETTING**: It is apparent that software systems are increasing in size and complexity on a daily basis. One of the most difficult aspects of this is the rise in consumer standards. As a result, various types of software applications of various sizes and complexities are being created. The need to incorporate necessary and improved functionalities, such as a Web-Based interface, adds complexity to any program such as more security and reliability, as well as the ability to run on a range of platforms and operating systems, as well as new developments including supply chain management, customer relationship management, enterprise resource planning, and others.

**THEOROTICAL METRICS**: In order to assess the complexity of the available modeling tools and techniques, Rossi and Brinkkemper developed a set of theoretical metrics. The number of objects, relationships, and property types in each diagram of the modelling language are all taken into account in these metrics. As a result, the diagram becomes more complex as the number of constructs increases. As a result, practical metrics are established that provide a more concrete sense of assessing the difficulty of modeling languages.

**PRACTICAL METRICS:** Practical metrics are favored over theoretical metrics, as mentioned above, since we do not need to use all of the constructs of any diagram at all times. Furthermore, due to poor short-term memory, humans have a propensity to decompose any problem into sub-problems. The decomposition of diagrams into smaller sub-diagrams is ignored by theoretical metrics. In any diagram, some structures are used more often than others. As a result, weights should be assigned to each build. Finally, the 80/20 rule notes that only 20% of the diagram structures can be used to specify 80% of software system solutions. As a result, functional metrics provide a more realistic image and hence a more realistic comparison between different modeling tools.

**RESEARCH**: The team performed a Delphi analysis and research that needed 29 global UML practitioners to classify the most commonly used UML diagrams. It was discovered and concluded that not all diagrams and related structures are used for every device construction, and that functional metrics are the preferred comparison metrics over all other options.

**CONCLUSION**: Theoretical metrics, we may conclude, do not include an accurate comparison of various modeling methods and languages. Developing any useful metrics for any project or modeling language, on the other hand, is dependent on a number of variables, but it is possible and useful. In recent years, as the size and complexity of software systems has increased, so has the complexity of modeling languages.