Automated Software Quality Visualisation Using Fuzzy Logic Techniques

James Senior, Ian Allison, Jon Tepper

Abstract

In the past decade there has been a concerted effort by the software industry to improve the quality of its products. This has led to the inception of various techniques with which to control and measure the process involved in software development. Methods like the Capability Maturity Model have introduced processes and strategies that require measurement in the form of software metrics.

With the ever increasing number of software metrics being introduced by capability based processes, software development organisations are finding it more difficult to understand and interpret metric scores. This is particularly problematic for senior management and project managers where analysis of the actual data is not feasible.

This paper proposes a method with which to visually represent metric scores so that managers can easily see how their organisation is performing relative to quality goals set for each type of metric. Acting primarily as a proof of concept and prototype, we suggest ways in which real customer needs can be translated into a feasible technical solution.

The solution itself visualises metric scores in the form of a tree structure and utilises Fuzzy Logic techniques, XGMML, Web Services and the .NET Framework. Future work is proposed to extend the system from the prototype stage and to overcome a problem with the masking of poor scores.

Introduction

With the advent of new methodologies, process-driven management and new project management tools, the number of software metrics in use has increased dramatically (White et al., 2004). Software Process Improvement (SPI) programmes, like the Capability Maturity Model Integration (CMMI), require that metrics are recorded in order to achieve accreditation (Reitzig, et. al, 2003). As organisations strive toward CMMI Level 4 or 5 the number of metrics they must record increases (White et al., 2004). Core to SPI are the metrics produced from measurement of software practices and processes.

This measurement is seen by the Software Engineering Institute (SEI) as pivotal in the improvement of “management and work processes of software development and acquisition” (Goethert and Siviy, 2004). Having implemented these processes, companies often find themselves producing vast quantities of metrics to satisfy the audits they must pass in order to achieve certain levels of compliance with CMMI, but organisations then often fail to make full use of metrics because they struggle to consistently understand indicators derived from measurement data. Their investigation identified numerous challenges faced when organisations attempt to make use of software metrics. The major concerns of the SEI are:

- Information is often misunderstood or misinterpreted
- Lack of data integrity
- No base for comparison

(Goethert and Siviy, 2004)

The problems found are not specific to CMMI, which is a generic approach to achieving and assessing software capability. Rather the problems relate to the capture of metrics across multiple projects within a single organisational domain. The real potential for software metrics is in their analysis so that they can be used to help achieve the goal of any software development through managing risks, and subsequently the process improvement programme by learning to improve the processes used to develop software and hence improve quality and efficiency (Walden, 2002).

Software metrics, therefore, have struggled to penetrate into mainstream software engineering because they were not able to provide management with this kind of information and hence failed to deliver their primary objective (Fenton and Neil, 2000). Indeed, metrics are sometimes viewed as a hindrance to the developers and managers who have to compile them (Alexander, 2003). The future success of metrics lies in bringing together different areas of software development and testing and enables managers to make predictions, assessments and tradeoffs during the project lifecycle.

So, one of the challenges now facing software development organisations is organising, representing and indeed understanding the vast array of metrics currently being tracked. This is complicated by the dispersed nature of metrics, reducing the value of recording the metrics in the first place. To stem this loss of value, organisations require a system that can organise and represent metrics in one place.

Solving the problems highlighted by using an automated application would promote understanding and interpretation of the information provided by metrics, it should ideally be classified and then summarised in a format easily understood by management. “A technique which can be used to reduce this complexity is software visualization, as a good visual display allows the human brain to study multiple aspects of complex problems in parallel” (Lanza and Ducasse, 2002). Once in a familiar format, managers could spend more time making strategic decisions than having to understanding exactly what the information meant.

This paper presents an application that addresses these issues by incorporating fuzzy logic concepts in the system architecture. The work was undertaken collaboratively with a global technology organisation (known here as GlobalTech) as a “proof of concept”, with the application being a vehicle for the ideas formulated with the customer. The high level requirements agreed with the customer included the implementation of a tool capable of representing metric scores visually to senior managers, thus allowing them to make better strategic decisions. More specifically the application should allow the user to create visual representations in the form of tree structures that display the performance of metric scores for software projects. Validation reviews with the sponsors at GlobalTech provide one form of assessment of the results.

Evaluation of Current Software Metrics Visualisation

There are many software packages available that provide visualisation of software metrics. One problem that these types of application attempt to solve is the easy digestion of complicated information. However, the packages typically process source code only. Such source code analysis tools measure an application’s size, complexity, maintainability, portability etc. An example of a source code analytical tool is CodeCrawler which is used to analyse any object orientated source and produce a tree-like structure representing relationships and collaborations between objects in the system. Other examples of source code analysis tools include CodeCheck – a C++ parser, which focuses on core code-related metrics and CodeHistorian, which analyses change history and ways in which a development team codes.

Other types of decision support systems (DSS) offer a vast array of visualisation techniques which present business information to the user in an intuitive way. For example, Cognos Visualizer, gives the user a series of gauges representing the health of certain aspects of the business. Others use scorecards and indicators to track milestones, goals and deliverables to make sure that they are on track to achieve expected results (Selby, 2004). An example of the traffic light system being used in scorecards can be seen in Microsoft’s Business Scorecards Accelerator product. It focuses on providing a scorecard system using traffic lights to indicate performance of organisations. The main difficulties foreseen with the development of metric related software in this format is the arrangement of boundaries that define the colours for each type of data.

Using Fuzzy Logic to Represent Metric Data

The proposed solution to this problem of boundary definition is fuzzy logic. Fuzzy logic provides the ability for a machine to perceive the world as humans do by representing vague and ambiguous knowledge (Negnevitsky, 2005). The intention is to blur the boundaries of binary thinking by allowing the classification of objects into multiple sets. Fuzzy sets are intended to blur boundaries so that various items can belong to any set (Hopgood, 2001). Fuzzy logic has been used to try and predict error prone code modules using historical data (So et al., 2002), but not for project management metrics.

At GlobalTech, the metrics that were being recorded were detailed in the organisation's Quality Assurance Plan. Some of the metrics and their attributes are listed in table 1.

Metric	**Frequency**	**Measure**	**Format**
Total Project Hours of Effort	Weekly	Hours	float
Project Rework Percentage	Weekly	Percentage	float
Estimated Remaining Effort	Weekly	Hours	float
Earned Value	Weekly	Hours	float
Schedule Variance	Weekly	Hours	float
Cost Variance	Weekly	Hours	float
Defects by lifecycle phase	On Completion of lifecycle	Defects	integer
Frequency of Emergency Changes in Production Measure	Monthly	Change Requests	integer
Open and Closed Change Requests	Weekly	Change Requests	integer
Open and Closed corrective actions	Weekly	Corrective Actions	integer

Table 1 - Example Characteristics of Metric Data

For each metric captured by the organisation, three boundaries were defined to reflect the project success criteria. These boundaries represent performance in the form of traffic lights: green signifying 'on target – no action required'; amber signifying 'action required to meet target'; red signifying 'redemptive action required to salvage the situation'. These boundaries are configurable for each metric as some metrics might have different expectations or targets compared to others.

The calculation for the parent node – i.e. the score for the entire tree (project) – uses each metric (denoted by child nodes). The centre of gravity function method of defuzzification, that takes the balance point of all membership functions (Hopgood, 2001), was used to calculate the colours of the nodes in a metric tree by balancing the three possible traffic light colours based on their membership to each one.

For each metric there were three colour scores that represented the differing degrees of health for a given metric. The values of the boundaries for these colour scores could be between any range of numbers although they were commonly percentage values such as those shown in Table 2.

Colour Score	Ranges for scores of x
Green	$0\% \leq x < 10\%$
Amber	$10\% \leq x < 20\%$
Red	$20\% \leq x < 30\%$

Table 2 - A typical arrangement for Metric Colour Scores

These ranges for the colour scores were not static and had to be able to change easily so they could take on any value the user desired. This arrangement of ranges and colour scores lent themselves to set theory and hence later fuzzy logic, where each colour was set in the universe of discourse of metric scores.

Primarily, classical (crisp) set theory was considered as a method with which to calculate membership of a metric score to a particular metric score set. This way, a metric score would either be a member of a metric score set or not, i.e. a Boolean value; true or false. However, when calculating the overall score for the metric tree, these Boolean values were too imprecise because they did not reflect the balance of score within each set i.e. how "green" a good score was.

For example, consider two metrics that contribute to a metric tree as in Figure 1, one with a high green score ($S_{1,,}$i.e. at the good end of the green range) and the other with a high amber score (S_2, i.e. at the good end of the amber range). One would expect the overall score of the metric tree to be green because a high amber score is not far away from being a green score itself (according to the ordering of the traffic light colours). In this scenario, however, classical set theory would deem that the overall score fall on the border between Green and Amber, B_1. However, the expected result would be closer to $(S_2 – S_1) / 2 + S_1$.

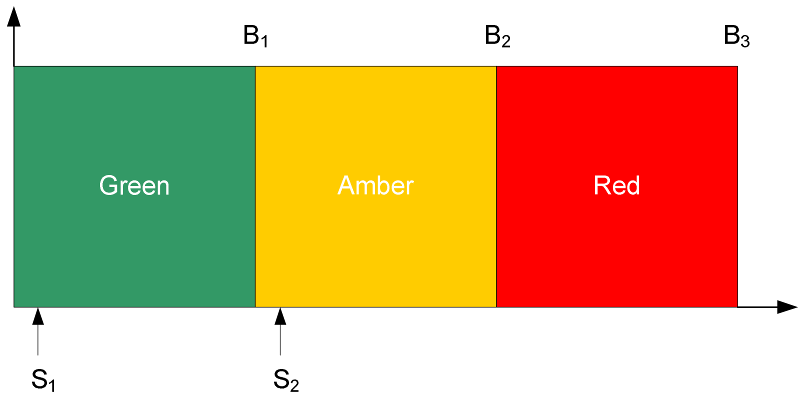


Figure 1 - A typical Metric Score scenario

Negnevitsky states "the concept of a set is fundamental to mathematics. However, our own language is the supreme expression of sets" (Negnevitsky, 2005). If the above example was to be expressed in language, then S_1 would be described as being a "very green" score and S_2 would be described as being a "very amber" score. As the user would describe these scores in this way, it follows that their expectations should be translated mathematically in the same fashion. The most appropriate method to use is fuzzy sets where degree of membership to a particular metric score set can be accurately measured.

Choosing Membership Functions for Sets

Considering Amber is between Green and Red in the metric score ranges, a triangular membership function was deemed most applicable as it provided an even score when moving either direction away from the core of the function.

For Green and Red metric scores a membership function that accentuated good and bad scores respectively was required. This meant that for Green scores, as the score tended to better, the membership became stronger more quickly. This is because scores close to being very green, i.e. to the strong side of the core of the membership function, should be scored as full membership because they were more extreme values. The same applied to Red scores but for the opposite direction of gradient membership function. The type of membership function chosen was the trapezoidal function, where either positive or negative gradients adjacent to a plateau starting at the core and ending at the opposite end of the range. An example of these types of trapezoidal functions can be seen in Figure 2.

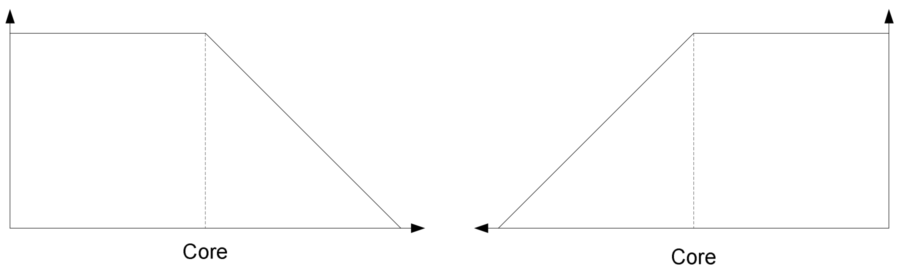


Figure 2 - Two Trapezoidal Membership functions

Other membership functions were considered including linear types but they would not give much bias towards polarised (extreme) scores, for example, there is a certain 'leeway' given to good green scores where if a certain score is achieved, say, in the top 5 percentile, it should be treated as if it has achieved a full score; i.e. a full membership. This is particularly poignant when the ranges are small and a small change in a score will result in a large change in membership.

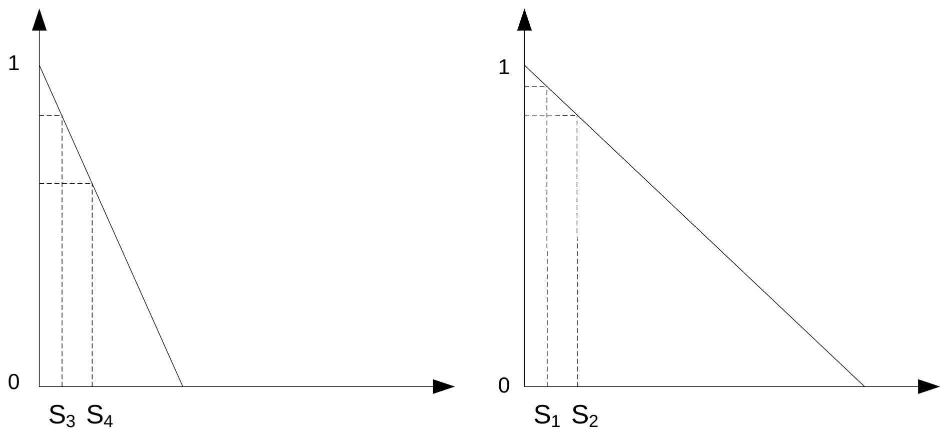


Figure 3 - Linear Membership functions with varying range sizes

In Figure 3, the graph showing the smaller range illustrates this point well, where the difference between S_3 and S_4 is small yet the difference in membership is fairly significant. For a given metric these scores are both considered good and therefore would merit full membership of the particular set. This would mean in later defuzzification they would provide extra weighting for their particular set. A linear function would be unable to achieve this kind of positive discrimination of scores.

Even less applicable for the Green and Red score sets are triangular membership functions. This is because once with a triangular function, once past the core, the better (in the case of Green) or worse (in the case of Red) the scores are, the less the degree of membership to that particular set. This point is illustrated in Figure 4 where S_1 is a better score than S_2 yet has a lesser degree of membership to Green due to the triangular membership function. The same applies to the Red Score triangular membership function where S_3 is a worse score than S_4 yet is less of a member to the Red score set.

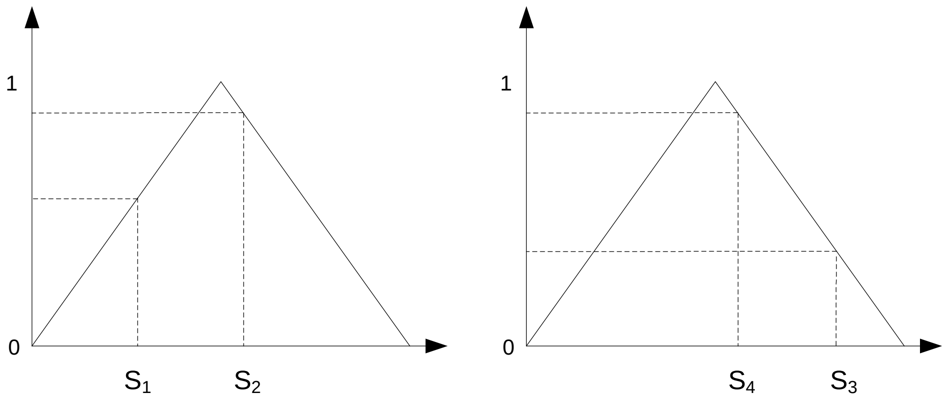


Figure 4 - Triangular Membership functions

Fuzzification

Colour sets provided by the user for each type of metric began and ended adjacent to one another as shown in Table 2. Having experimented with different membership functions it became clear that having functions that began and ended cleanly next to each other was not going to produce expected results overall scores. A case in point is Figure 5 where values were near the borders of sets (apart from extreme Green or Red scores). When fuzzified these scores would result in very low or no membership to a set. This was not desirable because it meant that their score, despite being significant in its own right, had little or no bearing on the outcome of the overall metric score.

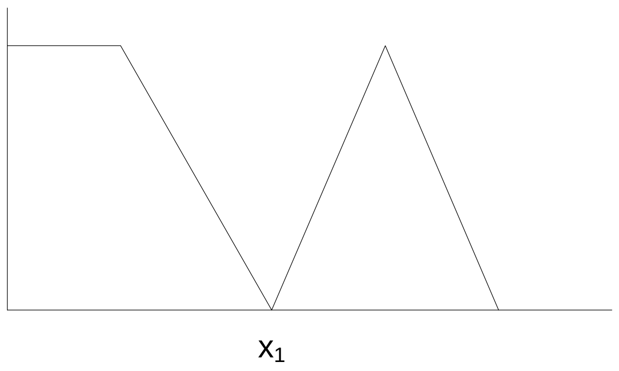


Figure 5 - A scenario where x1 has no membership to either set

The solution was to introduce an element of fuzziness to the sets. This could potentially have been done by the user when specifying traffic light colour sets that overlapped with one another. However, this was deemed confusing for the user and hence it was decided that it should be carried out after accepting the traffic light colour sets from the user.

The amount of fuzziness, i.e. the amount by which sets were to overlap, was the next important factor to consider. It was decided that the most effective way to do this was to measure fuzziness as a ratio of the difference from the core of the membership function to one of the boundaries (known as the interval). This value was then added onto the appropriate edge(s) (depending on the membership function) and accounted for the fuzziness. In Figure 6 an example is given of a trapezoidal function and a triangular function being extended in such a manner.

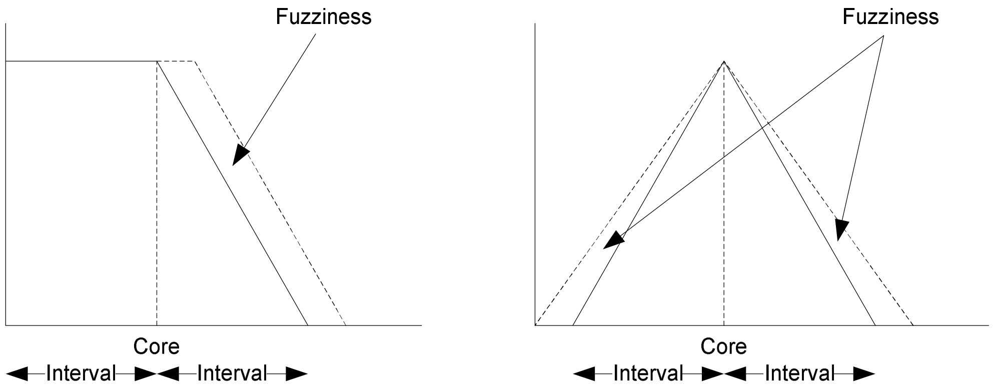


Figure 6 - Introducing Fuzziness to Trapezoidal and Triangular Membership Functions

Using Parts of an Inference Engine

Fuzzy Inference is the mapping of an input to an output using the theory of fuzzy sets (Negnevitsky, 2005). The Mamdani method is a common prescription for Fuzzy Inference, carried out in a series of steps. After attempts to build a Mamdani inference engine to solve this particular problem it was clear that not all the steps were necessary for this particular application and so his method was altered slightly. The first stage of inference was the fuzzification of results where raw data was fuzzified for each metric included in the user's metric tree. This step was definitely required.

The second and third stages of inference are where Mamdani's engine is modified, and the rule evaluation and aggregation of rule consequents were omitted from the process. The reason for this was because the problem that was being solved was simple in comparison with other rule based systems and therefore rule evaluation and aggregation of rule consequents was overkill and would not add any benefit to the solution.

For calculation of the overall metric tree colour, the aggregation of rule consequents was replaced with an aggregation of fuzzified metric scores as can be seen in Figure 9. This process had the potential to aggregate scores to over 1.0 but this was not a problem as the calculations done in defuzzification could cater for values outside of the range of 0 to 1. In this way all contributing data from any metrics included in the tree could have bearing upon the overall score in the next stage, defuzzification.

Defuzzification, step 4 in Mamdani's method, was needed as it decided the colour of each node in the metric tree. A centroid function was used to split the values in the colour sets into two equal sets and the location of the split was taken as the colour for the particular node. Depending on whether individual metrics were being evaluated or the metric tree itself, these values in the sets would be non-aggregated or aggregated values respectively. This centroid function is also known as the centre of gravity (COG) and there are several methods with which to calculate it. The most accurate centroid function is achieved by using a continuum of points from the aggregated output, however, an accurate estimate can be gained from a sample of points from the same output (Figure 7).

$$\text{COG} = \frac{\sum_{x=a}^{b} \mu_A(x)x}{\sum_{x=a}^{b} \mu_A(x)}$$

Figure 7 - Centre of Gravity Function using a sample of points along the x-axis

The process of aggregating all scores (Figure 9), provided an ideal method from which to evaluate the metric tree's overall colour score as each metric that contributed to the tree had its fuzzified values combined with others to give an overall picture of the score. From this the COG of all metrics' colour scores could be calculated and be used as the overall metric score.

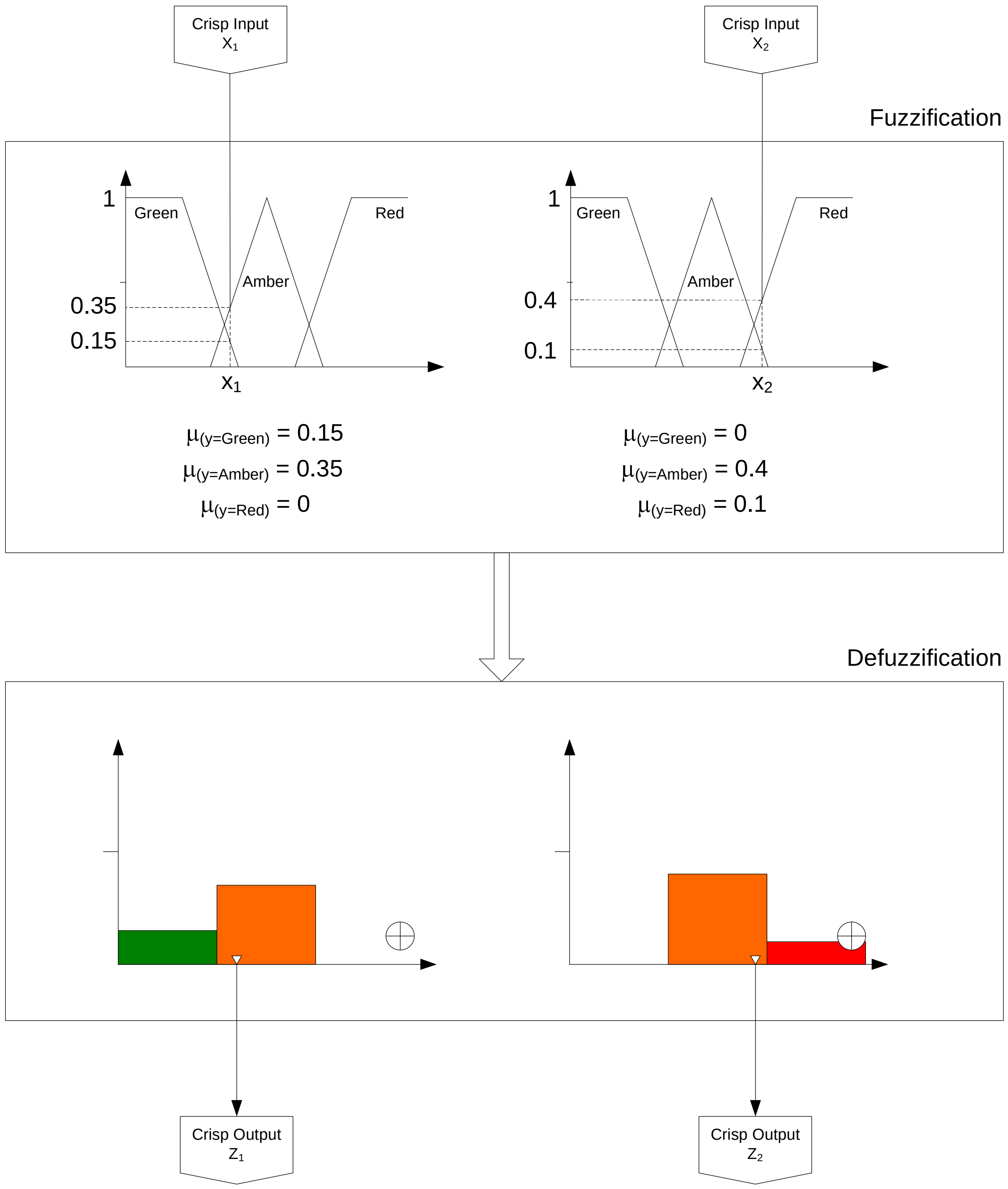


Figure 8 – The use of fuzzy principles for individual metric colours (Diagram adapted from Negnevitsky, 2005)

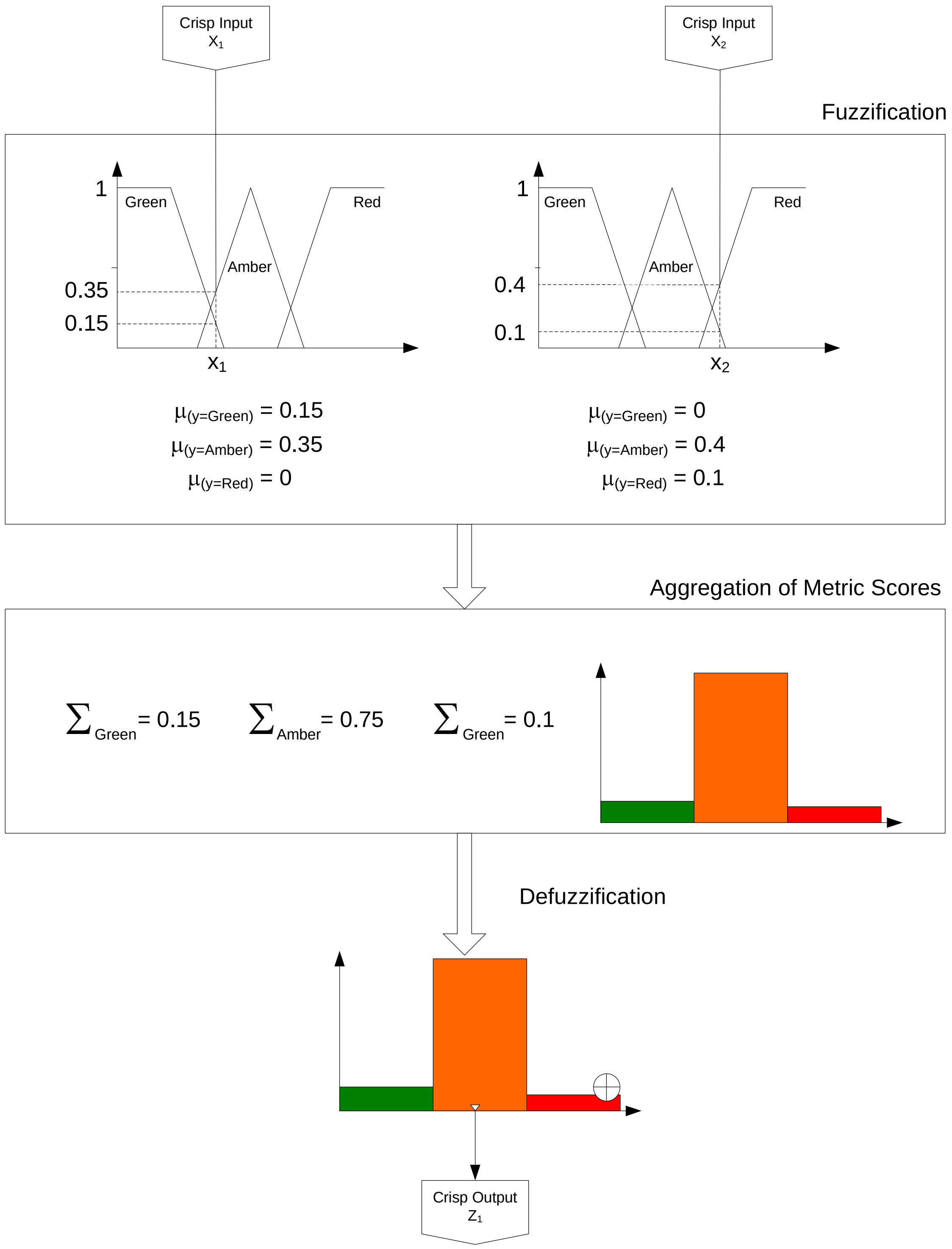


Figure 9 - The use of fuzzy principles for a metric tree's overall colour (Diagram adapted from Negnevitsky, 2005)

System Architecture and Development

Following the Extreme Programming (XP) Methodology (Beck, 2004), the application was developed in a series of three iterations. These iterations made up the first release of the application. The High Level Requirements (HLR) of the tool were captured in what is known as a "Story List", a key component of the XP methodology. User stories have a similar purpose to use cases except that they are designed to replace large requirements documents.

As is the way with XP, design is not formal and is expected to be bourn out in fragments of code and prototypes that represent small parts of the application (Jeffries, 2004). The first prototypes provided an exploratory activity into how best to solve particular sections of functionality that were anticipated to be complex due to lack of expertise in that area. These early prototypes were then evolved from simplistic, isolated sub-systems into interoperating yet loosely coupled modules that represented the most abstract form of their predecessors.

The XP stories were used to guide how the application should be arranged into separate “Engines” with responsibility for driving its particular area of functionality within the application. There were three main “engines” that delivered the core functionality and another additional set of systems that were responsible for the presentation layer:

- XGMML Engine
- XGMML Rendering Engine
- Metric Data Engine
- Metric Tree Engine

The way these different engines grouped together can be seen in the application architecture diagram in figure . There were also other systems that managed the presentation layer code but the main business logic was contained in the engines above.

Test-Driven Development (TDD) was used to ensure the quality of the product as it fits with Extreme Programming (Beck, 2003). XP encourages developers to build unit tests before coding the actual functions themselves, arguing that doing it this way round makes writing the actual code much easier and faster. This method helps the developer think about the purpose and features of individual functions and ensures code is modular and easily readable. It promotes inherently simple designs because the developer is influenced by the need to fulfil tests first and foremost and is therefore primarily driven to deliver value to the customer (Newkirk and Vorontsov, 2004).

During the development of this application the rules of TDD were adhered to and tests were coded in a separate project to that of the main code. This process helped the design to be thought out by writing tests that would confirm the ideas that had been translated from paper into the application’s code. By the end of development, 36 test modules had been written that encompassed 130 tests which in turn made 173 assertions on the application’s code, resulting in the production of 3000 lines of code. However, as seen in demonstrations and user testing the results were extremely good in that the application was reliable, producing the expected results for metric tree scores. However, it should be noted that some tests contained multiple test assertions in an attempt to simplify the test creation. This meant that for more complex modules, like the FuzzyEngine there were many more checks carried out than actually indicated here. Furthermore, where some modules involved collections of information, such as the MetricTree, the number of assertions depended on the size of collection and therefore could be unlimited depending on how the test was configured.

Having established with the customer that metric data stored by the application should be available to other applications, it was necessary to use a method with which to expose the data. Web Services were chosen as they allow communication across the Web between heterogeneous, programming-language independent applications. Web Services work by transmitting information over SOAP (Simple Object Access Protocol) in XML format (Gottschalk et. al., 2002). XML or eXtensible Markup Language can describe any type of data and more importantly can be parsed by any platform making for much more loosely coupled applications.

Data Transfer Objects (DTO) were applied as a key part of a web service implementation (Newkirk and Vorontsov, 2004). DTO’s are used to transmit all types of data from the Metric Data Engine. DTOs are representations of the database schema that present a “flattened” view where relationships between tables are hidden to the consumer of the DTO. This model helps keep the complexities of the data model away from the client, and reduces the amount of data that must move across the “wire” (Sandoz et al., 2003). In order that the DTO is platform-independent, it was created using an XML schema. This code can be used by the web service producer side of the application in turning the MetricDataSet object into a DTO before it is transmitted across the Web Service.

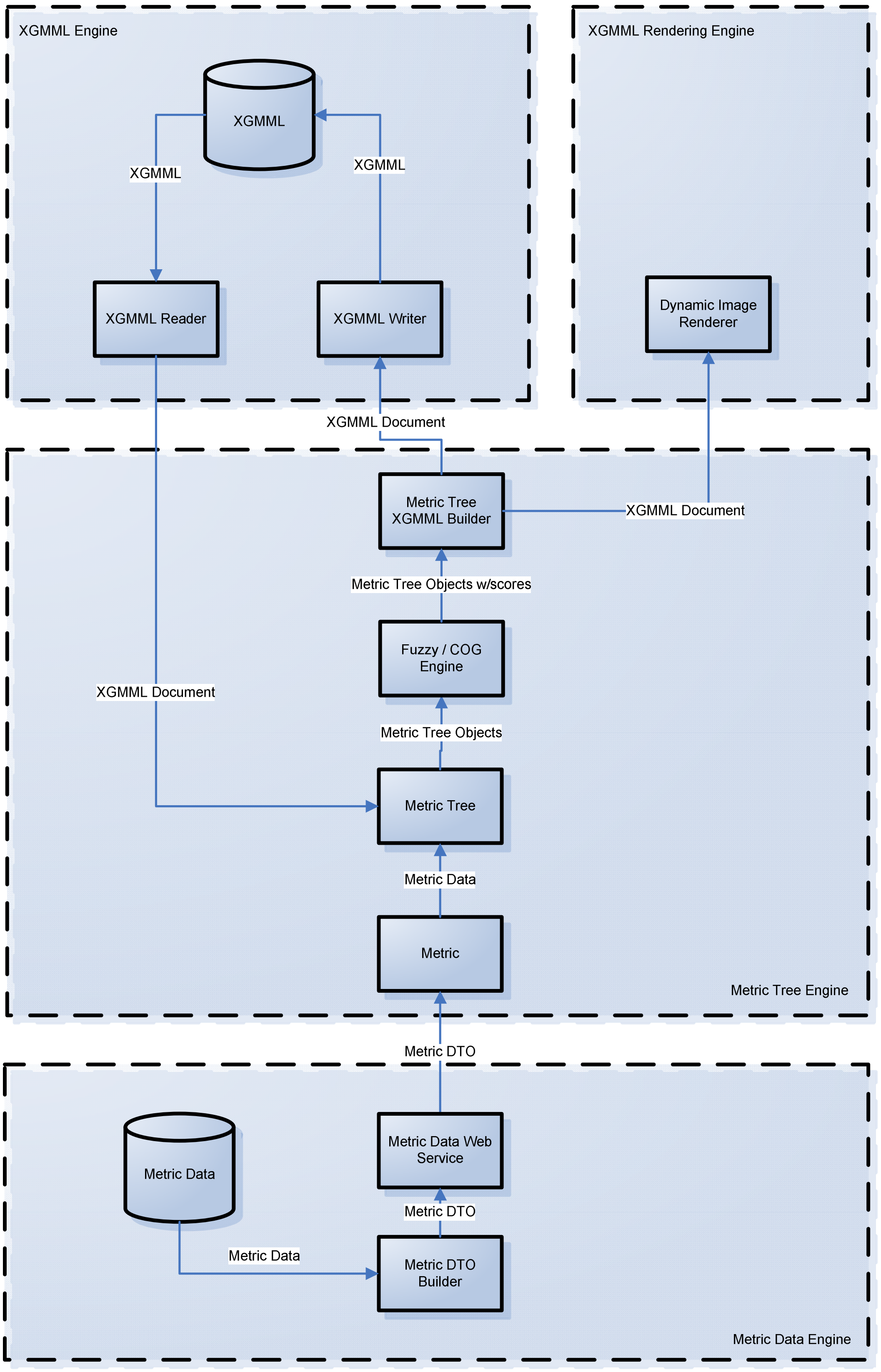


Figure 10 – System Architecture

Fuzzy Metrics Via Metric Tree Engine

The most pivotal object in the application's architecture, the MetricTree was introduced as a structure that represented a central body under which all information relating to a Metric Tree could be stored and processed. Its primary role was to record the configuration of a user's metric tree including which Metrics and Projects to include.

As much of the functionality from the Metric Tree was modularised to other areas of the application it resulted in the Metric Tree itself referencing many different types of modules. An example was the responsibility it had for initiating loading of metric data for each metric in the metric tree's configuration. This involved calling the load operations for each selected metric in the configuration. The resulting scores found from the data were then processed by the Centre of Gravity Engine (part of the fuzzy logic engine) to give an overall score for the tree.

The Fuzzy Engine was responsible for performing fuzzification of metric scores, catering for Triangular, Trapezoidal Left and Trapezoidal Right membership functions. The TrafficLightScorer module brought the Fuzzy Engine module into the context of this application, acting as a wrapper to perform fuzzification calculations for metric scores. The Traffic Light scorer passed "Fuzzy Information" such as the score to evaluate, the membership function set values, the type of membership function to calculate and lastly the amount of fuzziness to be applied.

The Centre Of Gravity Engine combined with the Fuzzy Engine and Traffic Light Scorer provided the tools required to carry out the fuzzy inference designed for this application. The engine makes use of the TrafficLightScore container, for which it carries out the centre of gravity function for three equal areas of mass; Green, Amber and Red. They are each given equal importance and the point of balance is found with the colour and actual score provided as output.

Graphical Representation via XGMML Engines

GlobalTech required that that the visualisation was in the form of a tree structure. It was therefore not ideal to store the information in a database because of the nature of a nodal structure that it is better modelled by objects rather than in a flat-relational style data store. The kind of information required to be stored was position of nodes relative to their parents and attributes of nodes like size, colour, and weight. One method that enables this kind of storage is Graph Modelling Language (GML). GML was borne out of the need to share graphs and charts between applications without the need for conversion between proprietary file formats (Himsolt, 1996). This file format provided an open standard for applications to build and distribute graph information easily. It is capable of storing any information relating to graphs, nodes and edges which makes it able to represent many different types of graphs and charts.

The XGMML version, which uses an object format with XML tags, was used to save metric trees because of the nature of the tree structure and the way that XGMML captures a metric tree configuration. It uses a series of tags to describe nodes and edges of graphs and its primary purpose is to allow exchange of different graphs between XGMML rendering applications. From the requirements of the application, the different components of a metric tree were understood and then mapped onto the XGMML 1.0 Specification (XGMML, 2005).

As each metric needed to be a traffic light colour, it was decided they should be circular and have a colour attributable to it. Lines linked metrics together joining the parent circle to the other circles representing the individual scores of metrics. The weight of each metric according to user preference was saved in the XGMML structure. Furthermore, other configurable attributes of the metric tree had to be saved, including start and end dates, granularity and projects to include in the pull of metric data. These items weren't actually part of the graphical representation but were important to the configuration of the metric tree and needed to be saved with the tree structure itself.

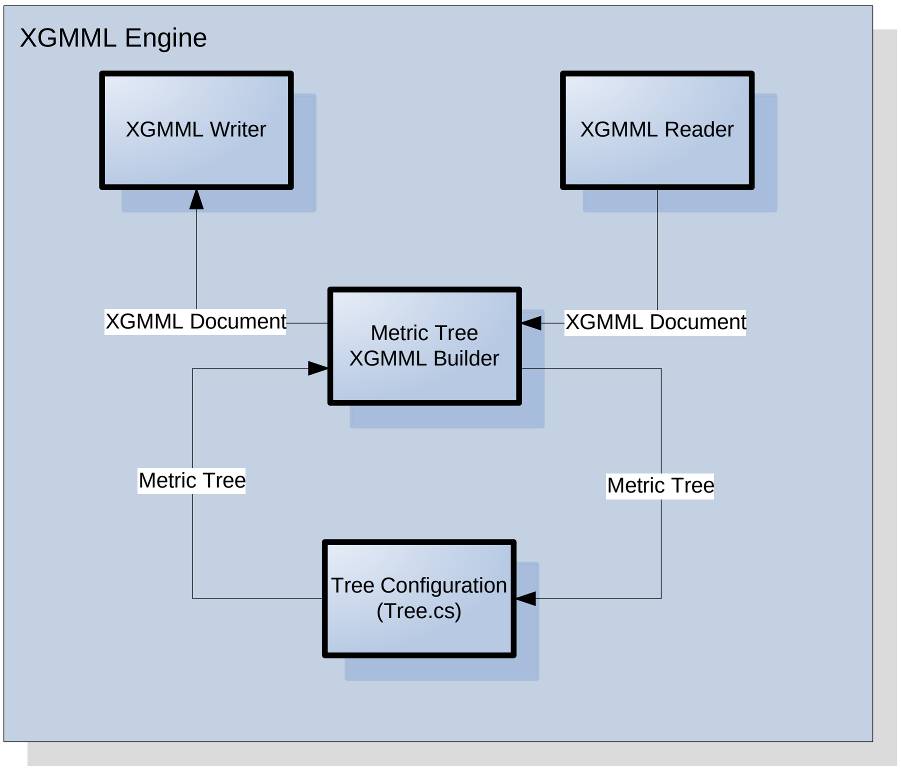


Figure 11 - The Metric Tree XGMML Builder

The XGMML engine (Figure 11 - The Metric Tree XGMML BuilderFigure 11) comprises two main modules: the writer and reader. To improve the design aspects of the functionality have been decoupled through two other modules during the iterations of the development: metric tree XGMML builder and Tree Configuration. The XGMML Writer engine was an important component of the XGMML Engine that was responsible for the creation of well-formed XGMML documents representing graphical objects in XML. The XGMML Reader parsed XML documents and produced an XGMML Document object to be used in other parts of the application.

Metric Tree XGMML Builder/Tree Configuration modules take a MetricTree object and breaks it apart into component parts: metrics, scores and configurations and the reassembles it in the form of an XGMMLDocument ready for use in other areas of the application. It is responsible for introducing a Metric Tree's graphical form. It is used in conjunction with the reading and writing of XGMML documents to disk, as the final metric tree had to be converted to an XGMMLDocument before the reading and writing modules could process it.

The drawing of XGMML documents within the application was performed by the XGMML Rendering Engine. The code generating the image sat behind a standard web page, but instead of outputting HTML content to the screen, the application instructed the page to output an image onto which the XGMML was rendered. This was achieved by setting the "Response" object's content property to "image/jpeg". .NET's drawing libraries were utilised in order to render an image, in particular objects such as "Bitmap", "Graphics", "Pen" and "Brush" were very useful. In order for the XGMML Document to be rendered, each node and each edge in the document was drawn in turn according to its elements.

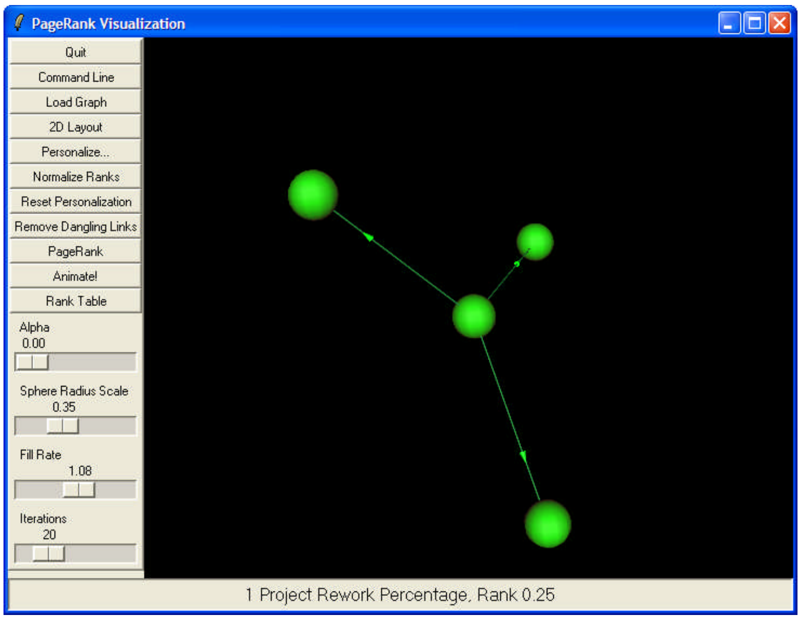
Evaluation

The success of this research is based primarily upon meeting the requirements of the customer. Whether or not this was achieved is assessed through discussion of the story tests and their pass and fail state. These and the secondary project goals are all discussed in this section. Feedback from GlobalTech Management is also presented and discussed in this section.

XGMML

This language provided a good way of storing nodal structures graphically which leant itself to the storing of metric trees. It provided all the structures necessary to capture the characteristics of a metric tree, including nodes, edges and other configuration information like which projects to pull data for. The design of the application's XGMML functionality was decoupled from the rest of the application. The XGMML Engine could be used to represent any nodal object that used the same main elements of the language that were used by the Metric Tree.

Proof of the portability of XGMML was provided in being able to load saved Metric Trees in a 3rd party application called PRViz (http://www.cs.rpi.edu/~gregod/PRViz/Web/). This application interpreted the XGMML produced by this application successfully and rendered the tree in its own XGMML engine. The result can be seen in Figure 12.



Alpha 0.00
Sphere Radius Scale 0.35
Fill Rate 1.08
Iterations 20

1 Project Rework Percentage, Rank 0.25

Figure 12 - A Metric Tree loaded in Page Rank Visualization (PRViz)

Although not a requirement, this kind of portability was considered important when saving Metric Trees. By using an open standard (XGMML) Metric Tree representations could be used in other applications that supported it.

Web Services

As the quantity of metric data grows, it would become a valuable commodity which would potentially need to be harnessed by other areas of the organisation. By exposing this area of the application through web services, other applications could obtain information required. Due to the nature of the technology and the implementation of Data Transfer Objects and the appropriate paradigm, this information is also completely heterogeneous.

If this application is used more widely in the future some considerations will need to be made around the security of the information. Restrictions will need to be implemented as to who can access it as some of the data may be sensitive.

Feedback from Sponsor

Having developed the application in close quarters with the customer using Extreme Programming, feedback was given at each weekly meeting as to the progress of the project. This close contact was invaluable as it helped to establish if the interpretation of the requirements was accurate and if not, to make corrections early on in the development process before they became more costly to change in later stages. During the meetings story tests for the week's work were passed or failed according to the test description.

Upon completion of the development, a full demonstration was performed followed by some time to receive comment and feedback which formed the basis of discussions for direction of the application in later releases. The sponsors were represented by two individuals from a team responsible for establishing measurement standards, investigating new statistic methodologies and putting a measurement "culture" in place. They were specifically responsible for identifying ways of presenting data and evaluating new methods with which to do this, with a focus on automating the process.

The original goals of the project were to propose a method by which to solve the problems identified at the organisation with the metrics programme. The work undertaken in this research project was deemed of value and that the ideas presented by the application coupled with the technologies and theories chosen to solve the problems had the potential to be used.

System Limitations

As scores for metrics are averaged before being presented, there is the chance that some poor values can be over-shadowed in certain scenarios. For example, in a situation where for a particular metric there are many good scores and only one poor score, the average will tend to the good end of the score. This would occur in scenarios where the data range for a metric configuration was large. For smaller date ranges, for instance a month, the effect would be less so due to the maximum number of data items in a series being four. The solution would require the system to provide either a "drill down" facility or some warning of variance at the top level.

Conclusion

A software solution was delivered that met the GlobalTech's requirements. Through feedback received from the organisation it can be stated that this application has successfully achieved a new way to analyse and visualise software metrics. Fuzzy Logic techniques were employed in the calculation of metric tree scores having evaluated and experimented with other methods. Appropriate parts of the Mamdani fuzzy inference engine were used to meet the needs of the customer which resulted in the moulding of the technique to the requirements. Having evaluated the use of fuzzy logic in metric analysis, GlobalTech were able to demonstrate its potential and with further development the can evaluate the concepts in a controlled project-based experiment.

The use of Extreme Programming helped to manage the interaction with the sponsors to ensure the requirements were prioritised and the tool was validated as it developed.

As this application provides the shell of a full fuzzy logic inference engine, there is the potential for the customer to introduce their own rule base to the system and use the fuzzification and defuzzification engines provided. This will be proposed at the next meeting with Senior Management.

References

Alexander, G., 2003. 21st Century Schizoid Plans. Professional Tester, 16 (Oct)

The Balanced Scorecard Institute, 2005 <http://balancedscorecard.org/>.

Beck, K., 2003. Test-Driven Development: By Example. Addison-Wesley.

Beck, K., 2004. Extreme Programming Explained, 2nd Edition. Addison-Wesley.

Fenton, N., Neil, M., 2000. Software Metrics: Roadmap, Taken From: "The Future of Software Engineering", Anthony Finkelstein (Ed.), ACM Press.

Goethert, W., Siviy, J., 2004. Applications of the Indicator Template for measurement and Analysis. Software Engineering Institute, Carnegie Mellon University.

Gottschalk, K., (et al), 2002. Introduction to Web services architecture. IBM Systems Journal, 41(2).

Himsolt, M., 1996. GML: A Portable Graph File Format, <http://infosun.fmi.uni-passau.de/Graphlet/GML/gml-tr.html>, (5th November 2004).

Hopgood, A., 2001. Intelligent Systems for Engineers and Scientists, Second Edition, CRC Press.

Lanza, M., Ducasse, S., 2002. Understanding Software Evolution Using a Combination of Software Visualization and Software Metrics. University of Berne, Switzerland: Software Composition Group.

Negnevitsky, M., 2005. Artificial Intelligence – A Guide to Intelligent Systems, Second Edition, Addison-Wesley.

Newkirk, J., Vorontsov, A., 2004. Test-Driven Development in Microsoft .NET, Microsoft Press.

Reitzig, Miller, Kile, et. al, 2003. Achieving CMMI Level 2 in the Configuration Management Process Area Using IBM Rational Software Solutions, Rational Software.

Sandoz, P., (et al), 2003. Fast Web Services. < http://java.sun.com/developer/technicalArticles/WebServices/fastWS/>. (1st April 2005)

Selby, R., 2004. Software Requirements Metrics Provide Leading Indicators in Measurement-Driven Dashboards for Large-Scale Systems. <http://sunset.usc.edu/cse/pub/event/2004/COCOMO/files/WedAM/Wed_AM_02_Selby.pdf>, (11 Nov 2004).

So, S., Cha, S., Kwon, Y., 1998. Empircial evaluation of a fuzzy logic-based software quality prediction model. Advanced Insitute of Science and Technology, Korea. Department of Electrical Engineering and Computer Science.

Walden, D., 2002. A Business Case for CMMI-Based Process Improvement, General Dynamics – Advanced Information Systems.

White, Longenecker, Leidig, et. al, 2003. Applicability of CMMI to the IS Curriculum: A Panel Discussion, EDSIG.

XGMML, 2005 as published at http://www.cs.rpi.edu/~puninj/XGMML/.