

Measuring Complexity In X++ Code

Anders Tind Sørensen

Kongens Lyngby 2006
IMM-B.Eng-2006-42

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-B.Eng-2006-42: ISSN none

Summary

Almost from the beginning of software development there has been a wish of being able to measure the quality of the program code. One aspect that affects several areas of software quality is the complexity of the code. Limiting the code complexity can lead to more testable code, provides faster bug-fixing and makes it easier to implement new features. The purpose of this project has been to find and implement relevant complexity metrics for the programming language X++, which is a part of the Microsoft Dynamics AX ERP system.

After some investigation the following ten metrics were selected: Source Lines Of Code, Comment Percentage, Cyclomatic Complexity, Weighted Methods per Class, Depth of Inheritance Tree, Number Of Children, Coupling Between Objects, Response For Class, Lack of Cohesion in Methods and Fan In. They represent some of the most established measures available and are a combination of traditional metrics and metrics designed specifically for object-oriented languages.

Each of the chosen metrics was implemented as stipulated in the theory. Since X++ contains special language features (e.g. embedded SQL) that the original authors did not describe, it was necessary to find out what the original intend of the metric was, and then derive a reasonable solution.

The metrics has been integrated into the existing Best Practice tool, which allows developers to check that their code adheres to certain non-syntax rules. This way they can immediately determine if the complexity values of their code is outside acceptable ranges and hence may need changes to reduce complexity.

In addition to the Best Practice checks, the metric values can be extracted as raw data for statistical purposes. It is also possible to directly generate statistics on a team/module level.

Acknowledgements

I would like to thank the following people:

- **Michael Fruergaard Pontoppidan** for being my mentor at Microsoft.
- **Knud Smed Christensen** for being my mentor at DTU.
- **Hans Jørgen Skovgaard** for suggesting this exiting topic.
- **Ola Mortensen** for review of report.
- **Morten Gersborg-Hansen** for review of report.
- **Johannes C. Deltorp** for review of report.
- **Betina Jeanette Hansen & Victor** for love and moral support.

Resumé

Siden software udviklingens begyndelse har der eksisteret et ønske om at kunne måle kvaliteten af en programkode. Et af de aspekter der påvirker flere områder af softwarekvaliteten er programkodens kompleksitet. Ved at begrænse kompleksiteten kan man få en mere testbar kode og det bliver hurtigere at rette fejl og tilføje nye funktioner. Formålet med dette projekt har været at finde og implementere relevante kompleksitetsmålemetoder til programmeringssproget X++, som er en del af ERP systemet Microsoft Dynamics AX.

Efter nogle undersøgelser blev følgende ti målemetoder valgt: Source Lines Of Code, Comment Percentage, Cyclomatic complexity, Weighted methods per Class, Depth of Inheritance Tree, Number Of Children, Coupling Between Objects, Response For Class, Lack of Cohesion in Methods og Fan In. Disse metoder repræsenterer nogle af de mest etablerede målinger tilgængelige, og er en kombination af traditionelle metoder og metoder der er designet specifikt til objektorienterede sprog.

Hver af de valgte målemetoder er blevet implementeret som teorien foreskriver. Da X++ indeholder specielle sprogkonstruktioner (f.eks. indlejret SQL) som de oprindelige forfattere ikke har beskrevet, blev det nødvendigt at finde ud af hvad det oprindelige formål med målingen var, og ud fra dette aflede en fornuftig løsning.

Målemetoderne er blevet integreret med det eksisterende Best Practice værktøj, som tillader udviklere at kontrollere at deres programkode opfylder visse ikke-syntaks regler. På denne måde kan de med det samme se hvis kompleksitetsmålingerne af deres kode overskrider nogle grænseværdier og ændringer i koden derfor kan være nødvendige.

Ud over at indgå i Best Practice kontrollerne, kan værdierne fra kompleksitetsmålingerne også trækkes ud som rå data til statistiske formål. Det er også muligt direkte at generere statistikker på team/modul niveau.

Contents

Chapter 1	Introduction	3
Chapter 2	Project planning.....	3
2.1	Schedule	3
2.2	Development method	3
2.3	Security procedures	3
Chapter 3	Complexity and metrics	3
3.1	Measurements & Metrics	3
3.2	Complexity.....	3
3.3	Metrics in Object-Oriented systems	3
Chapter 4	Functional specification	3
4.1	Abstract	3
4.2	Overview & Justification	3
4.3	Target Market.....	3
4.4	Pillars.....	3
4.5	High Level Requirements	3
4.6	Overview Scenarios	3
4.7	Personas	3
4.8	Assumptions & Dependencies	3
4.9	Use Cases.....	3
4.10	Functional Requirements	3
4.11	Error Conditions	3
4.12	Notifications	3
4.13	Fields table.....	3
4.14	Reports	3
4.15	Testability.....	3
4.16	Translation & Localization.....	3
4.17	Performance, Scalability & Availability (Client Apps).....	3
4.18	Setup (Client Apps).....	3
4.19	Security & Trustworthy Computing	3
4.20	Extensibility & Customization.....	3
4.21	Technology Configurations & Platform Considerations	3
4.22	Sustainability Concerns	3
4.23	Supportability Concerns.....	3
4.24	Upgrade & Maintenance	3
4.25	Monitoring & Instrumentation (i.e. Watson & SQM).....	3
4.26	Usability	3
4.27	Dev & Test Estimates	3
Chapter 5	Design.....	3
5.1	Basic class design.....	3
5.2	Integration with the Best Practice tool.....	3
5.3	Metric statistics.....	3

Chapter 6	Implementation	3
6.1	Project	3
6.2	Base classes	3
6.3	Integration with BP	3
6.4	Metric implementations	3
6.5	Statistics generation	3
Chapter 7	Test.....	3
7.1	Unit tests	3
7.2	Functional test.....	3
7.3	Adherence to own rules	3
Chapter 8	Analysis of results.....	3
8.1	Results overview	3
8.2	Details	3
8.3	Comparison of selected modules.....	3
8.4	Comparison by team	3
Chapter 9	Metric evaluation	3
Chapter 10	Future improvements	3
10.1	Open issues	3
10.2	New ideas	3
Chapter 11	Conclusion.....	3
Chapter 12	Bibliography	3

Appendix A: Project diary

Appendix B: Source code

Appendix C: Setup instructions

Appendix D: CD with source code and the MBS Functional Specification

Chapter 1 Introduction

The ERP system Microsoft Dynamics AX contains the powerful programming language X++. This language enables users and vendors to create their own business objects and functions. When writing the code, it can be interesting to measure just how “good” quality the code is. According to [McConnell04] “good” code has the characteristics of being both maintainable and testable. Complexity has a very high impact on both the testability and maintainability of code, since developers who can easily understand how the code works, will be less prone to make errors.

The purpose of this project is to clarify which form of complexity analysis (eg. cyclomatic complexity, number of lines, lines with comments etc.) will be relevant to X++ code. The most relevant measurements should then be implemented for X++. A part of the project will be to design a solution that has the right level of integration with any existing tools inside Dynamics AX.

The target audience for this report is people with basic knowledge about developing in Dynamics AX.

Chapter 2 Project planning

This chapter contains information relevant for the planning and execution of the project. Please note that although this section was created in the beginning of the project it also contains information added at the end of the project.

2.1 Schedule

The shown project schedule was created to get an overview of how the project should elapse. The project is rated to 10 weeks, but due to a lot of holidays in the period, it actually lasted a little longer. Please note that the week numbers are the official European week numbers, and not the internal DTU.

Week	Milestone	Report	Design	Coding
18	Start 1/5	Project planning, Theory	Relevant metrics	
19		Theory	Integration with existing tools	Test of existing tools
20		Dev + syntax	Basic solution structure	Metrics Framework
21			All functionality	Non-OO metrics
22	Non-OO metrics 4/6			Non-OO metrics
23		Test Non-OO		OO metrics
24				OO metrics
25	OO metrics 25/6	Implementation		OO metrics
26		Test + analysis of results		GUI stuff
27	Code complete 9/7	Finalize report		
28		Finalize report		
29	Hand-in 17/7			

An up to date project diary can be found in Appendix A. This shows that all milestones were met on or ahead of time. The Non-OO metric implementations were completed by May 31st and the rest of the implementations were completed by June 20th.

2.2 Development method

For this project I will use the Test-Driven Development (TDD) method, since this is becoming more and more common at Microsoft. TDD is a part of what is called eXtreme Programming (XP), and the main goal of TDD is not testing software, but helping the programmer during the development process by having clear and unambiguous program requirements. These requirements can be expressed in the form of tests, and when all tests succeed the program is complete.

When coding, the steps are:

- Write a test that specifies a small functional unit.
- Ensure that the test fails, since you haven't built the functionality yet
- Write only the code necessary to make the test pass
- Refactor the code, ensuring that it has the simplest design possible for the functionality built to date

This is somewhat different from the traditional approach of first implementing and then testing, but gives the benefit of more testable code since it has been targeted towards testing right from the beginning. When adding new features later in the product cycle, one can always run the collection of tests, to ensure that new functionality will not break any existing functionality.

For a full explanation of TDD and its advantages/disadvantages, please refer to [Newkirk04].

2.3 Security procedures

As the project period is very limited, it will be very critical to lose work from system breakdown or theft of equipment. All the material for the project is stored in a single folder on a laptop. At the end of every working day a backup of the contents will be written to a CD that will be kept separate from the computer. Once a week a backup of the CD will be saved on a separate server.

Since there is only one contributor of material on this project, it will not be a problem with conflicting versions of documents or source code. However, every document (including the source code) will have a version number and a last-changed date, to have a common reference for review purposes.

Chapter 3 Complexity and metrics

This chapter provides the reader with some theory regarding the field of software metrics and complexity. A number of metrics will be introduced, including their definition and use.

3.1 Measurements & Metrics

Measurement has a long tradition in natural sciences. At the end of the 19th century the physicist Lord Kelvin formulated the following about measurement: *“When you can measure what you are speaking about, and express it into numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.”*

As the software development process matures, there is a bigger need to be able to evaluate the software being created. As Lord Kelvin stated, this means that we must have numerical values which describe the properties of the software. Many authors have proposed desirable characteristics that these software metrics must possess: The value must be computed in a precise manner; it should be reproducible; it must be intuitive and it should provide some useful feedback to the user of the measure to allow him to get a better understanding of how to make improvements. Also, a measure should be well suited for statistical analysis.

3.2 Complexity

The word “complexity” is defined by [Encarta] as “the condition of being difficult to analyze, understand, or solve”. Software complexity can be defined from a developer’s view, as the complexity involved in developing and maintaining a software program. Figure 3-1 shows that software complexity has three varieties: computational, psychological and representational. The most important of these are probably the psychological, which is composed of problem complexity, programmer characteristics and structural complexity.

Problem complexity reflects the difficulties in the problem space. The only measures of this are subjective, as it will depend heavily on the observer’s insight into the problem area. Also the programmer’s characteristics are hard to measure objectively, although some sources argue that it can be measured using IQ and personality tests.

The software literature has, due to the above problems, been focused primarily on developing structural complexity metrics which measures the internal program characteristics. An internal attribute of a product can be measured in terms of the product itself. All information that is needed to quantify the internal attribute is available from a representation of the product. Therefore, internal attributes are measurable during and after creation of the product. Internal attributes do however not describe any externally

visible qualities of the product, but they can be used to get an estimate of some external characteristics, such as testability or maintainability.

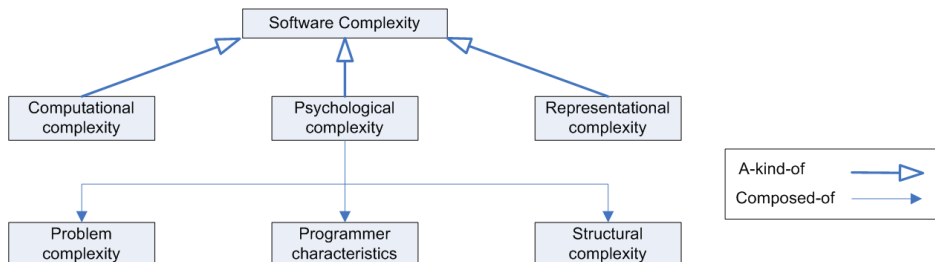


Figure 3-1 Classification of software complexity. Adapted from [Sellers96].

3.2.1 Effects on software quality

Figure 3-2 shows some of the elements that software quality consists of. The structural complexity can have a direct impact on how easy the product will be to maintain, because to maintain, one must first understand how the existing code works, then make the required modifications and lastly verify that the changes are correct.

The lower the complexity, the more maintainable a system is, and thus it decreases the time needed to fix bugs and speed up the integration/development of new features. Also, the complexity will have an indirect influence on the reliability because the easier it is to test a system the more errors are likely to be discovered before they reach the customer. This will contribute further to the perceived quality of the product

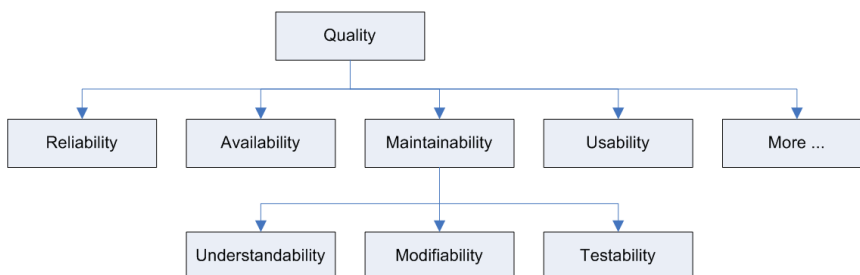


Figure 3-2 Hierarchy of software quality

3.3 Metrics in Object-Oriented systems

Traditional metrics have been applied to the measurement of software complexity of structured systems since the early seventies. Many sets of metrics have been proposed, and some have been established as de-facto standards, while some have only been used for special purposes and programming languages.

Although Object-Oriented (OO) systems have things in common with structured systems (e.g. basic algorithms), there are architectural differences that must be considered when measuring OO systems. For example, in OO systems there is a focus on peer-to-peer relationship rather than a hierarchical structure for control flow. Also, the presence of inheritance structures and the effect it can have on the system's complexity cannot be described by any of the traditional metrics, hence there was a need to develop new metrics that would better support the system's special properties.

One application of metrics in both types of systems is in terms of a threshold value or alarms. An alarm would occur whenever the value of a specific internal metric exceeds some predetermined threshold. Values that are not within the acceptable range should be used to draw attention to a particular anomalous part of the code. For many of the metrics the alarm levels cannot be global absolute values, but are dependent on the particular development environment and language constructs.

3.3.1 Traditional metrics

In this section some traditional code metrics are described. These have been chosen based on how commonly they are mentioned in literature and based on review of what other metric tools are using.

Note that in some of the theoretical descriptions of the metrics several ways to solve a problem is discussed. Which method is actually chosen for the X++ implementation will be stated in the Functional Specification.

3.3.1.1 Size (LOC/SLOC)

The size of the code is probably the oldest method of measuring how hard the code is to understand, and the measurement hereof is mentioned in more than ten thousand research papers. The size can be measured in many ways, where most of them include some counting of the physical lines of code, e.g. how many "Carriage return/Line feed" characters exists. Since most modern languages allows comments and blank lines in the code, this Lines of Code (LOC) count has been further specialized as Source Lines of Code (SLOC), where blank lines and comment-only lines will not be taken into account. SLOC can both be counted at the module (class) and method level.

The problem with SLOC is, that it can be difficult to use to compare code written in different languages, since the syntax may influence how much code is needed for a given operation. Also, some languages can have more than one statement on each line, which makes it hard to compare with more simple languages. The programmer's personal coding

style can also affect the outcome of SLOC, as there is usually more than one way to write the needed code.

Despite these problems SLOC is still an easy-to-understand metric that gives a good hint of the amount of effort that will be required to understand how a piece of code works. SLOC can also be valuable for the management as size measurements can be used in connection with resource allocation and estimation.

3.3.1.2 Comment Percentage (CP)

Comments in source code assist developers and maintainers in understanding the code. The Comment Percentage metric can be calculated as the total number of lines with comments divided by the total lines of code less the number of blank lines.

[Rosenberg97] states that NASA Software Assurance Test Center has found that a comment percentage of about 30% is most effective. Other authors suggest numbers ranging from 10% to 20%, but it will depend highly on the level of the programming language and the complexity of the computational problem.

3.3.1.3 McCabe Cyclomatic Complexity (V(G))

According to [Sellers96], the most established measure of module complexity is the Cyclomatic Complexity, which was introduced by Thomas McCabe in 1976.

Cyclomatic Complexity is computed using a graph that describes the control flow of a module, as shown on Figure 3-3. A module corresponds to a single function or subroutine and has a single entry and exit point. The nodes of the graph correspond to the commands of the module. A directed edge connects two nodes if the second command might be executed immediately after the first command. There are a couple of different definitions for the Cyclomatic Complexity, but the most common is:

$$V(G) = e - n + 2$$

where G is a program's flow graph, e is the number of edges (arcs) in the graph and n is the number of nodes in the graph.

The word "cyclomatic" comes from the number of fundamental cycles in a connected, undirected graph. A strongly connected graph is one where each node can be reached from another node by following directed edges in the graph. The cyclomatic number in graph theory is defined as $e - n + 1$. Program control flow graphs are not strongly connected, but they become strongly connected when a "virtual edge" is added, connecting the exit node to the entry node. Adding one to the graph theory definition to represent the virtual edge makes the Cyclomatic Complexity equal to the maximum number of independent cycles through the directed acyclic graph. Note that $V(G)$ is not the number of test paths through the code, since there are often additional paths to test [Sellers96].

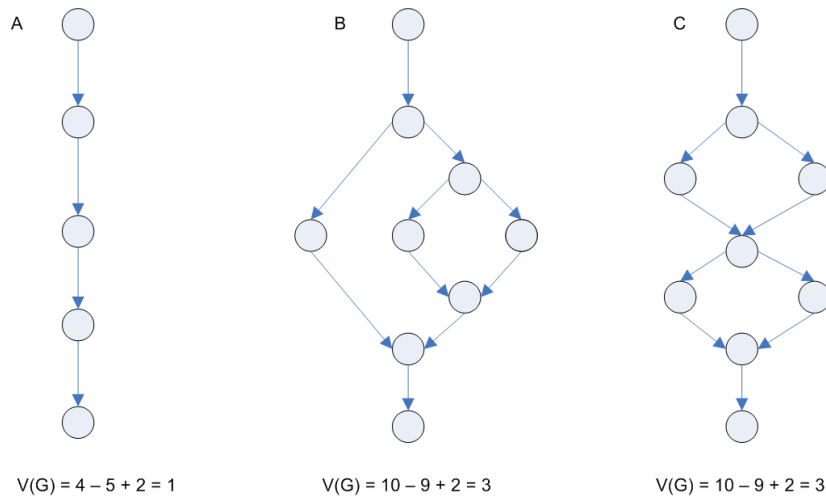


Figure 3-3 Control flow graph with sequence (a), nested `if` (b) and sequential `if` (c)

Figure 3-3 shows three examples of control flow graphs and what their Cyclomatic Complexity numbers are. As can be seen, a sequential program with no branches will always have $V(G) = 1$, no matter how many nodes the program consists of. It does not matter how any branches are structured: (b) has two nested `if`s whereas in (c) they are ordered sequentially, but still they have the same complexity number. Some argue, that intuitively (b) is of greater complexity than (c), but this is not the case when using the $V(G)$ formula.

According to [McCabe96] there are several practical ways of computing the Cyclomatic Complexity. Of course one could create a complete control flow graph with all the nodes and edges and apply the $V(G)$ formula directly. This approach can however require a great amount of computational work, since we are actually only interested in the decisions in the graph and not all the individual nodes. Instead we can take advantage of that most programming language constructs has a direct mapping to the control flow graph, and there by adds a fixed amount to complexity. I.e. an `if` statement, `for` statement, `while` statement and so on, are binary decisions, and therefore add one to complexity.

Boolean operators will either add one or nothing to complexity, depending on whether they have short-circuit evaluation semantics that can lead to conditional execution. For example the C++ operator `&&` will add one, since the second part of the `&&` statement will only be evaluated if the first part is true. Note that many implementations do not take these short-circuit Boolean operators into account. If these are suppressed it means that the Cyclomatic Complexity number will not be equal to the number of paths in the code, and thereby can not be directly interpreted as a measure of the number of test paths needed to fully cover the code. No matter which approach is taken, the important thing when calculating complexity from source code is to be consistent with the interpretation of language constructs in the flow graph.

As with Boolean operators, there are also different opinions on how to treat multiway decision constructs (like `switch`). Some argue, that since the `switch` statement only evaluates one expression, the entire structure should only add one to the complexity. Also, there is a discussion if the complexity contribution of the `switch` statement is exactly the number of case-labeled statements, even in the case where several case labels apply to the same program statement (fall-through). [McCabe96] recommends that the `switch` statement only contribute with the number of edges out of the decision node, so that fall-through case labels will not add to the complexity.

Values

A common application of the Cyclomatic Complexity is to compare it against a set of threshold values. Table 3-1 shows such a set. As stated in section 3.2, it will depend very much on the programmers experience and insight in the problem the code solves, how well these threshold values apply, but [McCabe96] finds these guidelines appropriate.

Cyclomatic complexity	Risk evaluation
1-10	Simple module without much risk
11-20	More complex, moderate risk
21-50	Complex, high risk
> 50	Un-testable module

Table 3-1 V(G) values

3.3.1.4 Function points (FP)

Function points are an ISO recognized software metric to size an information system based on the functionality that is perceived by the user of the system, independent of the technology used to implement the system. It is thereby probably the only metric that is not restricted to code.

In FP, system size is based on the total amount of information that is processed, together with a complexity factor that influences the size of the final product. The complexity factor is based on these weighted items:

- Number of external inputs
- Number of external outputs
- Number of external inquiries
- Number of internal master files
- Number of external interfaces

The weights assigned to each item depend on the specific system being developed. This is also one of the main arguments against FP, that two systems might not get the same measurement, as the weights are a matter of individual interpretation.

3.3.2 OO metrics

In this section, special metrics applying to Object-Oriented systems will be described. The majority of the included metrics has been proposed by [Chidamber91].

3.3.2.1 Weighted methods per class (WMC)

A traditional metric suite for Non-OO systems often includes the Number of methods, which is a simple count on how many methods a given code file contains. [Chidamber91] introduces the Weighted Methods per Class (WMC) metric, which is the sum of the complexities of the methods in a class. The complexity they mention can in principle be calculated in a variety of ways, but for most applications the Cyclomatic Complexity $V(G)$ will be used. Some also sets the complexity per method to a fixed value of 1, which is allowed according to the definition, thus making $WMC = \text{Number of methods}$.

The number of methods and the complexity of methods in a class is an indicator of how much time and effort will be required to develop and maintain the class. The larger number of methods in a class, the greater is the potential impact on its children, since the children will inherit all the methods defined in the parent class. Also, classes with a large number of methods are likely to be very application specific, which can limit the possibility of reusing the class.

There are some problems in calculating WMC, since the metric does not specifically state which type of methods to include (private, public, protected etc.). Also, it does not distinguish class attributes (i.e. the “get” and “set” methods) from regular methods, so there will be added one to the WMC count for each attribute.

Different limits for the WMC have been used in various metric tools. One way is to set the WMC to a fixed maximum number, e.g. 50. Another way is to specify that a maximum of 10% of classes can have more than 20 methods. This allows some large classes but the majority of classes should be small.

3.3.2.2 Response for a Class (RFC)

The metric Response for a Class (RFC) counts the number of methods (both internal and external) in a class that can be potentially used by another class. If a large number of methods can be invoked in response to a message to a class, the testing and debugging of the class can become more complex, since it will require a greater level of understanding from the tester or developer.

In [Chidamber91] RFC is defined as the number of distinct elements in RS ($RFC = |RS|$), where the response set RS is expressed by:

$$RS = \{M_i\} \cup_{\text{all } n} \{R_{ij}\}$$

where $\{M_i\}$ = set of all methods in the class and $\{R_{ij}\}$ = set of methods called by $\{M_i\}$. The response set can also be expressed as the number of local methods plus the number of remote methods.

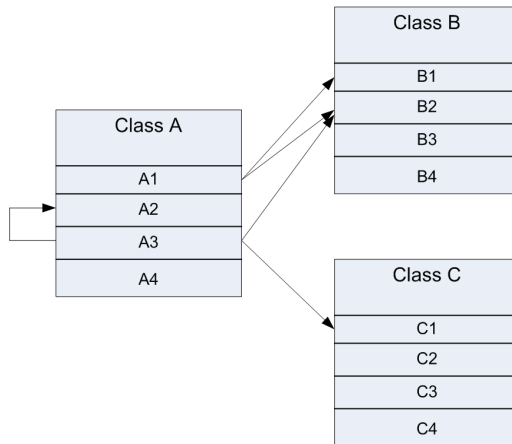


Figure 3-4 RFC example illustration

In Figure 3-4 is shown classes A, B and C each containing four methods. The arrows show method calls/usage from class A. The response set for the figure with regards to class A is calculated as:

$$\begin{aligned}
 RS &= \{A1, A2, A3, A4\} \cup \{B1, B2\} \cup \{A2, B2, C1\} \\
 &= \{A1, A2, A3, A4, B1, B2, C1\}
 \end{aligned}$$

From the above set will RFC equals 7, since it is calculated as the number of distinct elements in the response set.

3.3.2.3 Lack of Cohesion in Methods (LCOM)

Cohesion measures to which degree the methods of a class are related to each other. A cohesive class performs one function whereas a non-cohesive class performs two or more unrelated functions. Correct object-oriented designs maximize cohesion since it promotes encapsulation. A non-cohesive class might need to be refactored into two or more smaller classes. Cohesion also has an impact on complexity, since well grouped functionality will be easier to understand and maintain.

The original object-oriented cohesion metric was proposed by [Chidamber91] and measures the inverse cohesion. They define Lack of Cohesion in Methods (LCOM) as the number of pairs of methods on disjoint sets of instance variables (called P), reduced by the number of method pairs acting on at least one shared variable (called Q). If $P > Q$ then $LCOM=P-Q$ else $LCOM=0$. When LCOM equals zero it indicates that it is a cohesive class, where as a number greater than zero indicates that the class may be split into two or more classes.

For example, in class X of Figure 3-5, there are two pairs of methods accessing no common instance variables (f,g and f,h), while one pair of methods (g and h) shares variable E. This gives a LCOM of $2 - 1 = 1$.

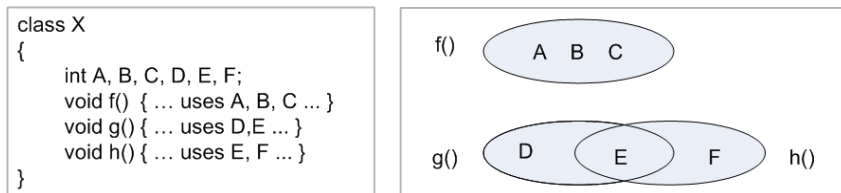


Figure 3-5 LCOM example illustration

This original definition of LCOM has received a great deal of criticism from various authors. Among these are the facts that LCOM gives a value of zero for very different classes, that, since it is defined on direct variable access, it's not well suited for classes that internally access their data via properties, and that the resulting value of LCOM in some cases will depend on the number of methods in the class.

To overcome the above-mentioned problems, several sources have suggested alternative interpretations/methods for calculating LCOM. [Sellers96] proposes LCOM* defined as $(m - \sum(mA)/a) / (m-1)$, where m =number of methods in the class, a =number of variables (attributes) in the class and mA =number of methods that access a variable. LCOM* decimal values will vary between 0 and 2, where 0 indicates high cohesion and 2 is extreme lack of cohesion.

[Hitz95] changes the definition of LCOM to measure the number of connected components in a class. A connected component is a set of related methods and class-level variables. Methods $a()$ and $b()$ are related if they both access the same class-level variable, or $a()$ calls $b()$ or $b()$ calls $a()$. The "Improved LCOM" (ILCOM) equals the number of connected groups of methods. $ILCOM=1$ indicates a cohesive class, which is the "good" class. $ILCOM \geq 2$ indicates a problem, where the class should be split into several smaller classes. $ILCOM=0$ happens when there are no methods in a class which is also a "bad" class.

No matter which of the LCOM definitions one may choose, they all measure cohesion between methods and data. In some cases data cohesion is not the right kind of cohesion. Some argue that a class groups related methods, not necessarily data. If classes are used as a way to group auxiliary procedures that does not work on class-level data, the cohesion will be low. Although this is still a good cohesive way to code, it is not cohesive in the "connected via data" way. A class that provides only storage will also get a low data-cohesion, if it does not act on the data it stores.

3.3.2.4 Coupling Between Objects (CBO)

CBO is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non-inheritance classes that a class depends on, i.e. classes that are used either through local instance variables or used as parameters to the methods of the class being measured.

Excessive non-inheritance coupling between classes prevents reuse, since a more independent class will be easier to reuse in another context. If a class has a high CBO it will also be more sensitive to changes in other parts of the design and therefore maintenance is more difficult. Also, strong coupling will make a class harder to understand or change by itself, if it is related to other classes. Designing systems that have the weakest possible coupling between modules, but where one still adheres to the general rules of the object's responsibility, can thus reduce complexity.

3.3.2.5 Depth of inheritance tree (DIT)

Many authors of OO metrics literature note the need to measure a system's inheritance structures. This is due to the fact that the deeper a class is in the hierarchy the greater the number of inherited methods will be, making it more complex. The most common of these inheritance measures is the Depth of Inheritance Tree (DIT) metric that counts how many ancestors (parent, grand-parent etc.) a class has.

In many OO based languages all classes inherit from some super class often called Object. This will result in all user created classes having a minimum DIT of 1, although some authors argue that Object should not be included when computing the DIT metric.

A recommended value for DIT is 5 or less, although some sources allow up to 8. The reason for these values is that very deep class hierarchies are complex to develop and comprehend.

3.3.2.6 Number of Children (NOC)

The number of children is the number of immediate subclasses to a class in the hierarchy. It is thereby a measure of how many subclasses are going to inherit the methods of the parent class. [Chidamber91] states that it is generally better to have depth than breadth in the class hierarchy, since it promotes reuse of methods through inheritance. However, if a class has a large number of children, it may require more testing of the methods of that class and hence will increase the testing time.

3.3.2.7 Fan-In / Fan-Out

Fan-Out is another name for the CBO metric. Fan-In measures the number of other classes having a reference to the class. Since Fan-In in particular is a system metric, it requires knowledge of all classes in the program, and cannot be measured by just evaluating the source code of a single class. Despite the possible implementation problems, Fan-In can be a very useful metric since it gives an indication of how high impact a change in the class can potentially have. The more who uses the class, the more caution and testing should be exercised when making a modification.

Chapter 4 Functional specification

Microsoft Business Solutions (MBS) has created document templates for documenting all steps in the software development process, right from the initial Quick Specification (describing idea/concept of the functionality) to the final test specification. This helps to ensure that when all sections of the template has been filled out, all aspects of the respective step will have been taken into consideration and nothing has been forgotten.

This chapter contains the sections from the MBS Functional Specification template that I have filled out. Please refer to the CD (Appendix D) for the complete specification document with descriptions of the sections included.

Product: Microsoft Dynamics AX 4.01

Feature name: BP Complexity Check

4.1 Abstract

The main goal of the feature is to supply the developer with measurements of how complex the code is.

4.2 Overview & Justification

When handing over code between teams, it is vital that the new developers quickly can understand the functionality of the code, and how the code is related to and affects other parts of the system. Also, Independent Software Vendors (ISV) must be able to understand the existing code in order to extend the functionality. It has been shown in various studies that the complexity of a piece of code has a great impact on the maintainability, understandability and testability of the code.

The new Complexity Check tool will provide developers with information of how well the code performs in connection with complexity- and other OO metrics. It can also be used for finding candidates among old parts of the code that may need rewriting to live up to the current coding standards.

The Best Practice (BP) framework already contains functionality for checking different rules when a class/method is compiled. It will thus be natural for the new tool to be based on the BP framework since developers are already familiar with this and since it will save some development time.

4.3 Target Market

This tool will both be targeted towards internal use and as well as Dynamics AX developers in all markets.

4.4 Pillars

MBS Pillar	Release Theme	Functionality Description
1. Best TCO	Low maintenance	It will decrease the Total Cost of Ownership by providing information that can result in lower maintenance and testing time

4.5 High Level Requirements

Number	Category	Requirement
0010	Required	The developer must be able to select if the complexity check will be included in the BP check
0020	Required	The complexity checks must support all language constructs in Dynamics AX version 4.0
0030	Required	Must support both traditional and OO based metrics
0040	Required	Outputs should be in the form of BP suppressible warnings and info.
0050	Required	Output from BP must be in both human- and machine-readable format so it can be post-processed automatically.
0060	Required	Results of the complexity checks should be included in the generation of the Best Practice Excel sheet.
0070	Optional	It should be possible to create statistics on all metric values, and not only those who causes BP warnings.

4.6 Overview Scenarios

Simon is developing a new feature in Dynamics AX. During the development of the actual code, he has set the compiler output level to 4, to enable automatic best practice checks when he compiles the code. Also, he has enabled check of the complexity best practice rules. This helps him to limit the complexity of the code he writes, by pointing out classes or methods where certain criteria are not met. By reducing the complexity, debugging or finding errors in the code at a later point in time will become much easier, as he can quickly understand what the code does and what impact any changes might have on other classes.

4.7 Personas

No.	Persona Name	Role	Comments
1.	Simon	System Implementer	All developers in general. Will only use Simon as persona in the use cases.
2.	Ivar	Inexp. VAR Sys implementer	
3.	Isaac	ISV Biz App Dev	
4.	Mort	IT Systems developer	

4.8 Assumptions & Dependencies

No.	Description	Type
1.	The new feature will (partly) be build on top of the existing Best Practice tool.	Dependency

4.9 Use Cases

With basis in the high level requirements and general domain knowledge, six separate use cases have been identified for the new tool. The use cases are listed in Figure 4-1 and the following sections will go through the details.

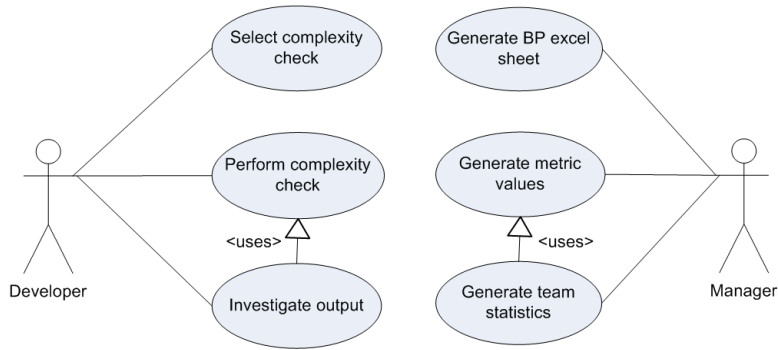


Figure 4-1 Use case diagram

4.9.1 Use Case 1: Select complexity check

4.9.1.1 Goals

Number	Goal
0101	Enable the developer to select if the complexity check should be performed as part of the Best Practice checks

4.9.1.2 Pre-conditions

Number	Pre-condition
0201	Must have a developer license to Dynamics AX
0202	The Dynamics AX client should be opened

4.9.1.3 Post-conditions

Number	Post-condition
0301	The user's selection is saved in the database

4.9.1.4 Basic Flow

Step Number	Action	Reaction
0401	Open the BP setup form, by selecting the menu Tools\Options... and clicking on the Best Practices button.	The "Best Practice parameters" form opens.
0402	In the tree expand the nodes "Best Practice checks", "Specific checks" and "Classes".	Tree expands to make the new complexity tree node visible.
0403	User checks/unchecks the complexity tree	Tree node gets checked/unchecked

Step Number	Action	Reaction
	node.	
0404	The user clicks the "OK" button to save the changes.	Changes to selection gets saved

4.9.2 Use Case 2: Perform complexity check

4.9.2.1 Goals

Number	Goal
0101	To perform the BP complexity check and have violations reported

4.9.2.2 Pre-conditions

Number	Pre-condition
0201	Must have a developer license to Dynamics AX
0202	The Dynamics AX client should be opened
0203	The complexity check option must be selected (use case 1)

4.9.2.3 Post-conditions

Number	Post-condition
0301	The complexity check has been performed and any violations to the complexity limits have been reported.

4.9.2.4 Basic Flow

Step Number	Action	Reaction
0401	User right-click on a class in the Application Object Tree (AOT) and selects Add-ins -> Check best practices	The best practice complexity check will output its results to the "Best practices" tab of the compiler output window.

4.9.2.5 Variations (Sub Flows)

Step Number	Condition	Action	Reaction
0401a	Compiler output level has been set to higher than 3.	User performs an action that will cause the class to be compiled. This can be that he has edited the source code of a class and selects "Save" in the editor.	The class will be compiled followed by a best practice check as in flow 0401.

4.9.3 Use Case 3: Investigate output

4.9.3.1 Goals

Number	Goal
0101	Enable the developer to see where the metric violation occurs

4.9.3.2 Pre-conditions

Number	Pre-condition
0201	The Dynamics AX client should be opened
0202	Must successfully have completed Use Case 2

4.9.3.3 Post-conditions

Number	Post-condition
0301	The code that has violated the metric is visible

4.9.3.4 Basic Flow

Step Number	Action	Reaction
0401	Once the Best Practice has completed and the Compiler output window has opened, switch to the Best Practices tab	The Best Practice tab opens.
0402	For each of the errors/warnings in the grid, double click on the line.	The code for the class/method that contains the metric violation will be shown in the MorphX Editor form.

4.9.3.5 Extensions (Alternative Flows)

Step Number	Condition	Action	Reaction
0402a	No Best Practice violations	None, since the code has passed the BP checks	None

4.9.4 Use Case 4: Generate BP Excel sheet

4.9.4.1 Goals

Number	Goal
0101	To have the output from the complexity check included in the Excel workbook, when using the CheckBestPractices startup command

4.9.4.2 Pre-conditions

Number	Pre-condition
0201	Must have a developer license to Dynamics AX

4.9.4.3 Post-conditions

Number	Pre-condition
0301	Any warnings or errors from the complexity check will appear in the Excel workbook

4.9.4.4 Basic Flow

Step Number	Action	Reaction
0401	Dynamics AX is started with the following parameter -startupcmd=CheckBestPractices_<excel file>	All classes in the AOT are compiled and the selected best practice checks are performed. The results are then grouped and inserted into the Excel template workbook.

4.9.5 Use Case 5: Generate metric values

4.9.5.1 Goals

Number	Goal
0101	Enable developers and managers to view metric values for a selected TreeNode and its subnodes.

4.9.5.2 Pre-conditions

Number	Pre-condition
0201	Must have a developer license to Dynamics AX
0202	The Dynamics AX client should be opened
0203	Cross references must be generated for the entire AOT

4.9.5.3 Post-conditions

Number	Post-condition
0201	Metric values have been generated and are viewable in a form.

4.9.5.4 Basic Flow

Step Number	Action	Reaction
0401	Open the new form "Metric results"	The "Metric results" form opens.
0402	Select or manually enter the path to an AOT TreeNode from where the generation must commence.	Start path has been selected
0403	User click the "Start generation" button	Metric values are generated for the selected TreeNode and all its subnodes. Afterwards the grid in the form is refreshed with the new data.

4.9.5.5 Extensions (Alternative Flows)

Step Number	Condition	Action	Reaction
0403a	The path given is not a valid TreeNode	User click the "Start generation" button	The error message "Invalid path to TreeNode" is shown

4.9.6 Use Case 6: Generate team statistics

4.9.6.1 Goals

Number	Goal
0101	Enable developers and managers to view metric values for a selected TreeNode and its subnodes.

4.9.6.2 Pre-conditions

Number	Pre-condition
0201	Must have a developer license to Dynamics AX
0202	The Dynamics AX client should be opened
0203	Use case 5 "Generate metric values" must have completed with success

4.9.6.3 Post-conditions

Number	Post-condition
0201	Metric statistics per prefix/team has been generated and is viewable in a form.

4.9.6.4 Basic Flow

Step Number	Action	Reaction
0401	Open the new form "Metric results"	The "Metric results" form opens.
0402	Switch to the "Team statistics" tab	The "Team statistics" tab is opened.
0403	Select or manually enter the filename/path to a text file containing combinations of teams and prefixes.	Filename has been entered
0404	User clicks the "Generate team statistics" button	Statistics (average, minimum, maximum and occurrences) are generated for the metric values, using the selected filename as input. Afterwards the grid in the form is refreshed with the new data.

4.9.6.5 Extensions (Alternative Flows)

Step Number	Condition	Action	Reaction
0404a	The filename is not valid	User click the "Generate team statistics" button	The error message "Invalid filename" is shown

4.10 Functional Requirements

This section describes which metrics has been chosen and clarifies any open issues from the theory section.

4.10.1 Chosen metrics

Since X++ is a highly Object-Oriented language, both traditional and OO metrics should be used. In the table below can be seen which metrics must be implemented in the new complexity metrics tool. Please refer to section 3.3 of this report for a detailed description of the individual metrics.

Metric	Level	Measures	Acceptable range
SLOC – Source lines of code	Method	Size	[1;40]
CP – Comment percentage	Method	Complexity	[10%;100%]
V(G) – Cyclomatic complexity	Method	Complexity	[1;10]
WMC – Weighted methods per class (1)	Class	Size and complexity	[1;50]
DIT – Depth of inheritance tree	Class	Size	[0;8]
NOC – Number of children	Class	Coupling/Cohesion	[0;10]
CBO – Coupling between objects	Class	Coupling	[0;20]
RFC – Response for class	Class	Communication and complexity	[1;50]
LCOM - Lack of Cohesion in Methods	Class	Internal cohesion	[1]
FI – Fan In	Class	Coupling	[1;50]

Computational notes:

(1) Only methods (both private, public and protected) specified directly in a class are included so any methods inherited from a parent are excluded. V(G) will be used as the complexity number in WMC calculation.

As can be seen in the table, mostly the metrics proposed by [Chidamber94] (WMC, DIT, NOC, CBO, RFC, LCOM) has be chosen for the OO part. Although many other metrics could have been included, the ones proposed by [Chidamber94] has, since their invention, been implemented in many metrics tools, so some statistical data will be available for comparing the X++ code with other systems. Among the users of these metrics is NASA's Software Assurance Technology Center, which has found them quite useful. The Fan-In has been included due to its unique ability to find classes that is not referenced from any other classes (potentially dead code).

The SLOC, CP and V(G) metrics has been chosen because they are relatively easy to understand, and although they are not directly aimed at OO systems, they still plays an important part in evaluating method complexity. The Function Point metric described in the theory section has not been included since it has a somewhat vague definition and is not restricted to code only.

4.10.2 Elements from the AOT to check

In Dynamics AX there is a distinction between “pure” code classes, and classes concerning the graphical representation of data. They are separated into the two Application Object Tree (AOT) nodes called “Classes” and “Forms”. Forms are mostly used to view/edit data, and the controls on the forms are in most cases bound directly to fields from a data source. Both classes and forms can contain general methods, but on forms, each control and field on the data source has their own “methods” node. Since it is vital to limit method complexity no matter what type of object the methods is attached to, the method-level metrics (V(G), SLOC, CP) will be calculated for all methods.

In X++, classes have a special method named `ClassDeclaration`. This method contains all class-level variables and the specification of the class (private/public + inheritance), but no real code. This method should not be included in the method-level metrics, since it is a class definition and not a regular method.

The class-level metrics will however only be calculated for the “pure” classes. This is because on forms, a lot of the work is done by using the visual designer to set various properties and not by creating code constructs. This means that the metric algorithms will be really difficult to apply to forms without redefining the meaning of the metrics.

4.10.3 Handling methods within methods

As oppose to many other Object-Oriented languages, the X++ syntax gives access to creating methods within other methods (referred to as “embedded methods”) like in C. None of the algorithms for computing the metrics (this goes for both traditional and OO) has taken this special case into account.

One of the main arguments for using embedded methods is that it can limit the use of the embedded functionality to a specific method. It can however be argued, that if it is necessary to have embedded methods to accomplish some functionality, then the outer and the embedded method has higher coherency with each other than with the rest of the methods in the class, and thus should be separated out in their own class. The use of embedded methods is not yet considered a direct violation to the best practices however it is generally not recommended when creating new functionality.

```

class A
{
    public void methodX()
    {
        int subMethodZ()
        {
            If (something)
                dothis;
            else
                dothat;
        }

        subMethodZ();
        subMethodZ();
    }

    public void methodY()
    {
        anotherCall();
        anotherCall();
    }
}

```

Figure 4-2 Use of embedded method

```

class A
{
    public void methodX()
    {
        methodZ();
        methodZ();
    }

    private int methodZ()
    {
        If (something)
            dothis;
        else
            dothat;
    }

    public void methodY()
    {
        anotherCall();
        anotherCall();
    }
}

```

Figure 4-3 Use of private method

Figure 4-2 shows a class which uses an embedded method and Figure 4-3 shows its equivalent class where the embedded method has been rewritten as a private method. Converting from an embedded to a private method can be somewhat tricky, since an embedded method has access to its outer method's variables. However, having more parameters in the new private method can solve this issue.

There are basically two approaches for dealing with embedded methods in the metrics calculation: Either to see the embedded method as just a code block within the outer method or to handle them as any other private method. If we "cut" out the code to convert it to a private method, no complexity penalties will be given to a method that has embedded methods, since calls to other methods do not contribute to the Cyclomatic Complexity count. One could argue that this is intuitively incorrect since methods with embedded methods will be of greater size and thus likely will require more effort to understand.

Using the first approach, where the embedded method is just considered a code block, will result in methodX of Figure 4-2 having a higher complexity count ($V(G)=2$) than the methodX of Figure 4-3 ($V(G)=1$), since the "if" in the embedded method will be included in the count for methodX. If we however look at the sum of method complexities for the class, using the "code block" approach, it will actually result in a lower total complexity than the "cut" approach ($V(G)=3$ vs. $V(G)=4$), since the private methodZ will add 2 where the embedded methodZ only will add 1 to the total $V(G)$. This issue can be solved by letting the "constructor" of the embedded methodZ add one to $V(G)$ of methodX, the same way as a normal method always has a $V(G)$ of one. This will result in methodX of Figure 4-2 having $V(G)=3$, methodX of Figure 4-3 having $V(G)=1$ and both having a total class $V(G)$ of 4.

Another advantage of using the “code block” approach is that measurement of SLOC and CP will also be more understandable and consistent than if we were to split the method into two parts. The downside is that we need to recognize the embedded method “constructors”, so we cannot use simple text search to find the code constructs (like “if”, “while”) for the V(G) count. Since this is only a minor problem, the “code block” method will be used in the implementation.

4.10.4 Handling SQL statements

Besides having embedded methods, X++ has another special language feature, which is the ability to have SQL statements directly in the code. Like with embedded methods, none of the sources discusses how to address this.

In Table 4-1 is given examples of SQL statements representing different combinations of keywords. The V(G) column suggest how much each statement should contribute to the Cyclomatic Complexity. The reasoning behind the suggested numbers will be explained below.

Case	V(G)	Example
1	0	Select t1 where t1.f1 == x;
2	0	Select t1 where t1.f1 == x && t1.f2 == y t1.f3 == z;
3	1	while select t1 where t1.f1 == x && t1.f2 == y t1.f3 == z
4	1	Select t1 where t1.f1 == x join t2 where t2.f1 = t1.f1;
5	2	while select t1 where t1.f1 == x && t1.f2 == y join t2 where t2.f1 = t1.f1 && t2.f2 == z
6	0	delete_from t1 where t1.f1 == x && t1.f2 == y;
7	0	update_recordset t1 setting f1 = x where t1.f1 == y && t1.f2 == z;
8	0	insert_recordset t1 (f1, f2) select f11, f22 from t2 where t2.f1 == y;

Table 4-1 Calculation of Cyclomatic Complexity for SQL statements

As can be seen in the above table, the basic `select where` does not add anything to the complexity of the method. This is because it can be compared to retrieving a single object from a regular function (e.g. `a=method1()`;) which does not add to the complexity.

A `while` in front of the `select` will add one, since it will result in loop like a regular `while` or `for` statement.

The Boolean operators `&&` and `||` in the SQL statements do not add one to V(G), as opposed to when they occur in normal expressions. The reason for this is that the SQL statement is executed by the Object Server, and all the elements of the Boolean operators will always be evaluated, so they can not be seen as short-circuit operators. Also, they can be considered as just being parameters to a function.

The reason why the `join` also adds one is that it will result in an additional value being returned. If we were to obtain the same without using the `join`, we would have to use a nested `while select` statement, which also would have added one to the complexity. However, if an `exists` or `notexists` keyword is in front of the `join`, then nothing should be added, since no value then will be returned by the SQL statement.

The keywords `delete_from`, `update_recordset` and `insert_recordset` in case 6-8 can be seen as bulk commands. This is equivalent to regular function calls with parameters, and thus they do not add anything to $V(G)$.

4.10.5 Handling Switch-statements

As described in section 3.3.1.3, there are different opinions on how to handle `switch` statements when calculating the Cyclomatic Complexity. The solution suggested by [McCabe96] will be adapted in the implementation, so `switch` statements add the number of edges out of the decision node to the complexity count. Following this approach, the code represented on the next page will result in $V(G)=3$.

```
switch(a)
{
    case 1:
        doOne();
        break;
    case 2:
    case 3:
        doTwoThree();
        break;
    default:
        doSomething();
        break;
}
```

Please note, that even if we were to remove the “break;” from the code, it would still result in the same complexity, although the first cases would fall through and result in all the code being executed. The reason behind this is that a test would still require min. 3 different test paths to verify its correctness, no matter if the “break;” were there or not.

4.10.6 Handling break and continue

In X++, keywords “break” and “continue” can be used within loops to either jump out of the loop or to immediately go to the top of the loop. It is quite common to use “break” and it is reasonably easy to understand when appearing in code, but the use of “continue” is not that widespread and the use of it might lead to confusing code and is generally not well seen in an object oriented language such as X++.

The keywords will most often appear as the result of a branch operation like “if”, since otherwise the code below the keyword would be superfluous as it never would be executed. The branch before the keyword will have added one to the Cyclomatic complexity, and since the branch and break/continue can be seen as one path through the

code, the actual keyword will not need to add additionally to the Cyclomatic Complexity count.

4.10.7 Handling Try-catch statements in V(G)

Error handling in X++ is done by surrounding code blocks by a try-catch statement. These statements can be seen as binary decisions, since the “catch” part is only executed if a certain (error) condition is met. As there can be more than one `catch` in the error handling statement, each of the error types being caught will add one to the Cyclomatic Complexity number.

4.10.8 Handling macros

In X++ macros can be defined the same way as in C. A macro is basically just a piece of text that gets replaced in the source-code. Macros are usually used as a convenient way of defining constants, but some macros also contains more complex code.

If the source code of a method is obtained by calling the `AOTgetSource` function on an AOT node, the raw code without the macros expanded will be returned. If we however use the `SysScanner` class to get the tokens, the macros will be expanded and any text from the macro will be included in the tokens.

When calculating SLOC and CP, the macros should not be expanded, since one should not get a line count penalty for declaring constants, which can make the source code a lot more readable. In the V(G) calculation however, the macros should be expanded so all branch keywords in the macro (if any) can be evaluated and included in the Cyclomatic Complexity count. Although one could argue that the macro is just a method, having application functionality outside well-defined objects is not in line with the Object-Oriented philosophy. Also, hiding functionality in a macro can make it very difficult to use unit tests to verify that the functionality works as intended. A “real” method should instead be added to an object, so the function can be tested and verified as normal.

4.10.9 Types to include in Coupling Between Objects

As described in the theory section, the original definition for CBO states that it is a count of the number of distinct classes that a class has references to. In X++ however, the definition of a class is somewhat fluent, since classes can be divided into Class and Form objects. Also, tables, extended data types and enumerations can be considered as a kind of classes, since instances of these can be created directly in the code. As the purpose of CBO is to identify classes which are coupled to a lot of other objects, the term “distinct classes” in the definition of CBO will for X++ be interpreted as “distinct object types”, so both regular classes, tables, forms, extended data types and so on, all will add to the CBO count.

The CBO metric can be used to evaluate how sensitive a class is to changes in other objects, and since the basic data types like `int` and `str` cannot be changed by the developers, they will not be included in the CBO count. Also, the table fields will not add to

the CBO count, since these can be seen as just being methods/properties on the table, so no matter how many fields of a table is referenced, the entire table will only contribute with one to the count.

4.10.10 Calculation of LCOM

[Etzkorn97] compares some of the known interpretations of the LCOM metric, to find out which one is most suitable. Their conclusion is that the LCOM as defined by Li and Henry is properly the most accurate. They also states that the one proposed by [Hitz95] is the same just calculated with basis in graph theory instead. Furthermore they have concluded that the measure should not include inherited variables, but that any constructor methods should be included in the calculations. The implementation will adhere to their conclusions and use LCOM as defined by [Hitz95].

One thing [Etzkorn97] does not take into consideration is static methods. Per definition a static method can not operate on instance variables, so a class with two independent static methods will always have $LCOM \geq 2$, which indicates that it should be split into two separate classes. In X++ it is however common practice to group related static functions in a single class. Also, many classes have a static method "description", which is used for reflection purposes. To avoid getting a misleading LCOM, the implementation should not evaluate static methods.

Another issue is abstract methods. They can not contain any code, and thus will always cause $LCOM > 1$ if they are included in the count. To avoid this problem, abstract methods will not be included in the calculation of LCOM.

4.11 Error Conditions

The new feature contains no error conditions or option boxes.

4.12 Notifications

All of the below mentioned notifications will appear in the Best Practices tab in the Compiler Output window as warnings. They all have the developer as the recipient, have no special requirements nor do they have any performance considerations. The notifications will only appear if the complexity metrics have been enabled in the Best Practices parameters window.

Notification name	Source Lines of code
Trigger condition	When BP check is run and the number of Source Lines of a class is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The number of Source lines (SLOC) of [Class name] is higher than [Recommended]: [Value]
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended value for SLOC [Value] - The SLOC of the class, i.e. 438
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Comment Percentage
Trigger condition	When BP check is run and the Comment Percentage of a class is lower than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The Comment Percentage (CP) of [Class name] is lower than [Recommended]: [Value]
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended value for CP [Value] - The comment percentage of the class, i.e. 11%
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Cyclomatic complexity
Trigger condition	When BP check is run and the Cyclomatic Complexity of a method is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The Cyclomatic Complexity (V(G)) of [Method name] is higher than [Recommended]: [Value]
Replacement variable definitions	[Method name]- Name of the method that is evaluated [Recommended] - The recommended value for V(G) [Value] - The V(G) of the method, i.e. 12
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Weighted Method for Class
Trigger condition	When BP check is run and the Weighted Methods for Class number of a class is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The Weighted Methods for Class (WMC) number of [Class name] is higher than [Recommended]: [Value]
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended value for WMC [Value] - The WMC number for the class, i.e. 55
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Depth of Inheritance Tree
Trigger condition	When BP check is run and the Depth of Inheritance Tree of a class is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short	The Depth of Inheritance Tree (DIT) of [Class name] is higher than [Recommended]: [Value]

format)	
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended value for DIT [Value] - The DIT value, i.e. 8
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Number of children
Trigger condition	When BP check is run and the Number of children of a class is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The Number Of Children (NOC) of [Class name] is higher than [Recommended]: [Value]
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended value for NOC [Value] - The NOC, i.e. 25
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Coupling Between Objects
Trigger condition	When BP check is run and the Coupling Between Objects metric of a class is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The Coupling Between Objects (CBO) metric for [Class name] is higher than [Recommended]: [Value]
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended value for CBO [Value] - The CBO value for the class, i.e. 15
Special requirements	None

Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Response For Class
Trigger condition	When BP check is run and the Response For Class value of a class is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The Response For Class (RFC) value of [Class name] is higher than [Recommended]: [Value]
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended max. value for RFC [Value] - The RFC value of the class, i.e. 20
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Lack of Cohesion in Methods
Trigger condition	When BP check is run and the Lack of Cohesion in Methods value for a class is higher than a set threshold value.
Recipient(s)	The developer
Notification content - alert message (short format)	The Lack of Cohesion in Methods (LCOM) value for [Class name] is higher than [Recommended]: [Value]
Replacement variable definitions	[Class name]- Name of the class that is evaluated [Recommended] - The recommended max. value for LCOM [Value] - The LCOM value of the class, i.e. 3
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

Notification name	Fan In
Trigger condition	When BP check is run and the Fan In of a class is zero.
Recipient(s)	The developer
Notification content - alert message (short format)	The Fan-In of [Class name] is zero.
Replacement variable definitions	[Class name]- Name of the class that is evaluated
Special requirements	None
Performance and scalability considerations	None
Configuration options	Complexity metrics can be enabled/disabled from in the Best Practices parameters window.

4.13 Fields table

Fields	Data Type	Def. Value	Req. (Y/N)	Edit (Y/N)	Save to Templ. (Y/N)	Size constraint in DB	Output Format	Help text
SysBPPParameters. CheckComplexity	NoYes	No	N	Y	N			Check class complexity metrics

4.14 Reports

The functionality will report its results through the compiler output window's "Best Practices" tab or the "Metric results" form, from which all the information can be sent to a printer. Also, the Excel workbook generated (use case 4) can be printed or by other means post-processed to form reports.

4.15 Testability

Using TDD will improve the potential for making the code testable through automation. Also, most of the new functionality will be non-GUI oriented algorithms, which makes ideal candidates for automated tests. To improve the testability the numeric values for each metric should be obtained directly from a property on the classes.

4.16 Translation & Localization

No special considerations.
All texts must be defined as labels, like in the rest of the application.

4.17 Performance, Scalability & Availability (Client Apps)

The new feature should be fast, since it may be run every time a class is compiled (if compiler is 4). The goal regarding performance is, that a best practice run on the entire AOT, where only the complexity metric has been selected, should take no more than 45 minutes on a 3 GHz computer with 1 GB RAM.

This feature should have no impact on the scalability of Dynamics AX, since it only operates on metadata and thus is independent on the amount of company specific information in the database.

4.18 Setup (Client Apps)

User must have a regular Dynamics AX client deployed. In order to use the functionality, a developer license must be installed.

4.19 Security & Trustworthy Computing

This tool will be used by developers who already have access to make changes in all parts of the application X++ code, so it will not add any additional security risks to the program.

4.20 Extensibility & Customization

For some metrics the threshold (alarm) values are more or less static no matter who are using them. Other threshold values will depend on specific company policies, which could state that no methods over a certain size limit are allowed. Due to this, the threshold values should be changeable on a company level.

Extensibility of the feature will happen through normal the layering strategy, where other can add new classes in their own layer. To make it easy to add new metrics, the BP complexity tool must be able to automatically find out which metrics exists across the layers. This can be done by creating a super class from which all metric classes must inherit.

4.21 Technology Configurations & Platform Considerations

The tool runs in the Dynamics AX client program, and thus it will have the same platform limitations as the rest of the client. It will be written to support the syntax of Dynamics AX 4.0 and if any significant changes are made to the syntax in future releases, it will need to be adjusted accordingly.

4.22 Sustainability Concerns

One of the goals of this tool is to help developers make code more understandable, and thereby provide for an easier handover of code between teams. The code of this tool

should itself comply with the new complexity checks, so the Sustained Engineering Team (SE) should be able to quickly understand and work through the code to resolve any issues or errors that might occur after it has been released.

4.23 Supportability Concerns

Both customers and support will probably have no knowledge of how the various metrics are computed. If they are to validate the measures, it is important that the help file contains information of how to manually compute each metric. The theory section from the report can be more or less directly used as help text, although it might need to be joined with the functional specification to be more practical applicable.

4.24 Upgrade & Maintenance

The feature will not have any negative effect on upgrade or maintenance, since it only requires one new field in the database and doesn't make any vital changes to the existing functionality.

4.25 Monitoring & Instrumentation (i.e. Watson & SQM)

Dynamics AX uses Watson¹ as default, so any errors that occur within Dynamics AX (and hence in the new tool) will be reported automatically. Evaluation of usage-tracking is not within the scope of this report.

4.26 Usability

By building upon the existing BP framework, it will provide a recognizable user experience, since most developers are already familiar with the terminology and usage of the BP tool.

4.27 Dev & Test Estimates

Due to the use of TDD as the development method, the below mentioned Developer Hours will include the time needed to produce the unit tests during development. Test Hours will be used for running tests on large amounts of data and analyzing the results.

No.	Feature Area Description	UE FTE Hours	UA FTE Hours	Dev Hours	Test Hours
1.	Traditional metrics			50	15
2.	OO metrics			60	15

¹ For more info about Dr. Watson go to <http://support.microsoft.com/default.aspx?scid=kb;EN-US;308538>

Chapter 5 Design

The chapter provides an overview of the solution structure, and how the various components are linked together. Many of the basic decisions were made in the functional requirements (section 4.10), so the purpose of the solution design is to create an appropriate program structure that will fulfill these requirements.

5.1 Basic class design

All metrics share certain common properties, no matter how they are computed and what level (class/method) they operate on. They all perform their computations on an AOT `TreeNode`, which will represent either a method or a class. Also, all of them must be able to return a string that states if the element violates the acceptable value ranges, and thus causes a best practice warning.

Figure 5-1 shows how these common properties are gathered in an abstract class called `CodeMetricBase`. The classes `CodeClassMetric` and `CodeMethodMetric` are also abstract and are used to divide the metrics into two groups, according to the level they operate on. All metrics will hence get their own class which then inherits from one of these two sub-base classes.

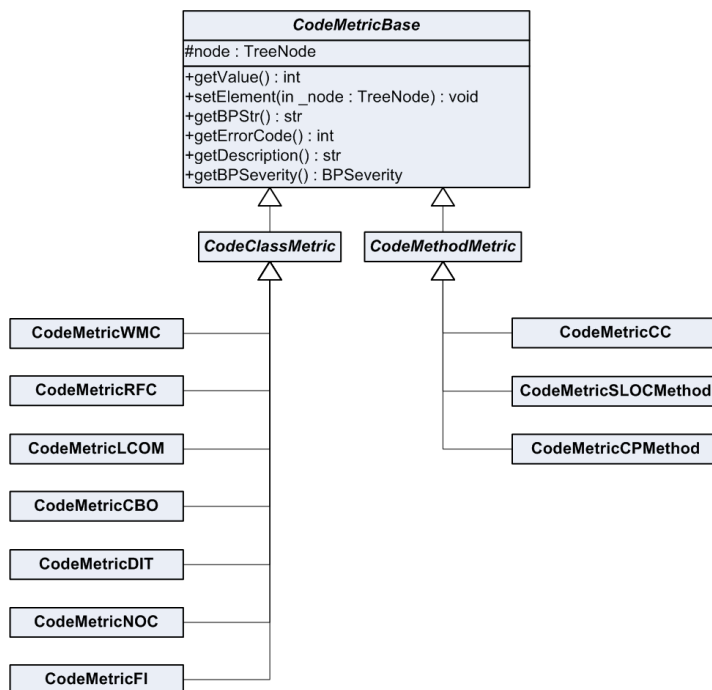


Figure 5-1 Basic class structure

5.2 Integration with the Best Practice tool

In the existing best practice tool, each kind of AOT object has its own corresponding BP class, which will check for problems that apply to the specific object type. They all inherit from the class `SysBPCheckBase`, which contains common functionality needed by all checks.

When a user starts a check of the best practices on a given node, the class `SysBPCheck` is responsible for iterating through all child nodes. Based on the type of the child node, `SysBPCheck` passes the `TreeNode` to the correct implementation of `SysBPCheckBase` and calls the `check` method on the BP class. Due to performance considerations, only one instance of each of the BP classes is created. A list with these instances is kept in `SysBPCheck`, so the classes can be used whenever needed.

To add the new complexity checks, new functionality must be added to the BP classes `SysBPCheckClassNode` (checks “pure” code classes) and `SysBPCheckMemberFunction` (checks methods, no matter of what their parent’s type is). As stated in the functional specification, it is important that it will be easy to add new complexity checks in the future. To avoid hard coding the names of the metric classes, reflection can be used to find the available metrics. That way new metrics will automatically be included in the checks, if they just inherit from either `CodeClassMetric` or `CodeMethodMetric`. Since it can take some time for the reflection API to actually find the correct implementations, the two BP classes will each hold a list with instances of the appropriate metric classes, so only one lookup will be needed.

5.3 Metric statistics

The reason that the `CodeMetricBase` in Figure 5-1 has an abstract method called `getValue` is to accommodate for the generation of metric values as described in Use Case 6. Also, all metric implementations must override the method `getDescription` which should return the short name (eg. “V(G)” or “WMC”) of the metric. This name will, along with the treenode path and value, be inserted into a table when the treenode is processed. After all values have been generated, they can be viewed in a grid on a form. The user can then use the grid’s build-in filter and sorting functionality to find any interesting data.

There was a wish from the managers to get some statistical values (average value, minimum value, maximum value, number of occurrences) for each metric per team and module. Internally, Microsoft has a list of which classes belongs to which team. This list is based on the prefix (first letters) or postfix (last letters) of the object names. If more than one prefix matches the object name, then it’s the longest one that has the best match, and if both a prefix and a postfix match then postfixes are considered as the best match.

To generate the statistical team/prefix values, the list needs to be supplied as an ASCII text file. Each line should consist of a team and prefix par, separated by semicolon, like this:

```
SCM Tech;Sys
```

Chapter 6 Implementation

This chapter explains how the different parts of the new complexity tool have been implemented. Only fragments of the source code will be shown here. Please refer to Appendix B for the complete source code listing.

6.1 Project

A new Private Project has been created in Dynamics AX, to hold all objects that are created or modified as part of the new tool. By creating a new project it is easy to create backups of the code, as one can merely export the private project as an xpo file. It also makes it easier for others to quickly install the tool, by just importing a single file.

The new project called “Complexity” has the following folder structure and contents:

- **Complexity** (Main project folder)
 - **Modified** (Existing AOT objects that has been modified)
 - **New** (New objects)
 - **Metric Framework** (Base classes and enumerations)
 - **Metric Implementations** (Implementations of all ten metrics)
 - **Other** (Support code)
 - **Statistics** (Objects for creating statistics on the metrics)
 - **Extended data types** (Extended data types for statistics tables)
 - **Test** (Test main folder)
 - **Test dummy classes** (Nonsense classes for test purposes)
 - **Unit tests** (Unit tests for the new objects)
 - **Unit test helper classes** (Classes for initialization of common functionality)

6.2 Base classes

As mentioned in the design chapter, the basic metric framework consists of three abstract classes: `CodeMetricBase`, `CodeClassMetric` and `CodeMethodMetric`. Although class- and method level metrics are basically the same, they have individual needs.

Many of the class level metrics needs to have information about which other methods or classes a class has references to. In Dynamics AX this sort of information is called “cross references”. Cross references for a `TreeNode` can be generated using a build-in function, and will be saved either to a single temporary table (which only exists as long as the temporary table variable is in scope) or to a “real” set of tables. Using the temporary table is somewhat faster than using the real tables, since writes to the database is avoided. In principle, the temporary cross references could just be created in the individual metric classes, but since it is needed for more than one metric, it is more effective if it can be passed as a parameter to the classes implementing `CodeClassMetric`. The UML diagram of Figure 6-1 shows the extra methods which have been added to `CodeClassMetric` to accommodate for the cross references issue.

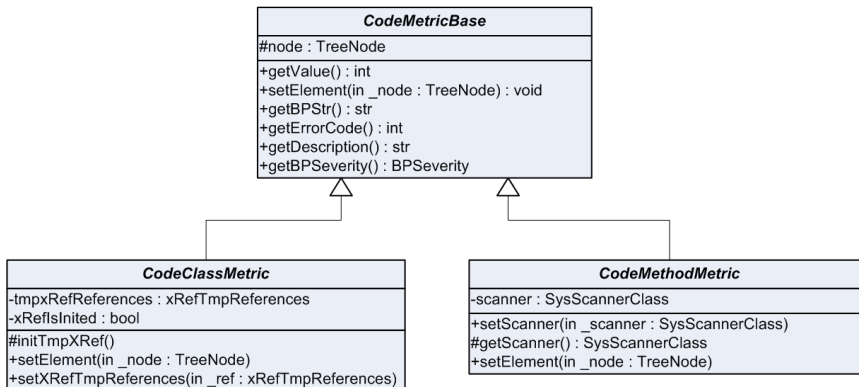


Figure 6-1 Base classes

The private variable `xRefIsInited` in `CodeClassMetric` is set to `false` whenever a new `TreeNode` is passed to the class. This is done by overriding the method `setElement` from `CodeMetricBase`. When a class inheriting from `CodeClassMetric` needs to use the cross references, it just calls the protected method `initTmpXref` to make sure that it has been properly created. The source code for that method can be seen below. It uses the class `xRefUpdateTmpReferences` to perform the actual update.

```

protected void initTmpXRef()
{
    xRefUpdateTmpReferences tmpUpdate;

    if (!xRefIsInited)
    {
        //Create tmp references for the entire class
        tmpUpdate = new xRefUpdateTmpReferences();
        tmpUpdate.fillTmpxRefReferences(node);
        tmpxRefReferences = tmpUpdate.allTmpxRefReferences();

        //Set the flag to true
        xRefIsInited = true;
    }
}
  
```

The method level metrics can have a need to use a scanner class to get the tokens of the source code. When the metrics are checked as part of a best practice check, the `SysScannerClass` have already been created, and since it might take a little time to create, it would be beneficiary if the instance could be passed to implementations of the `CodeMethodMetric` class, like with the cross references on the class level. Figure 6-1 shows which methods have been added to optimize the use of the `SysScannerClass`.

6.3 Integration with BP

This section describes how the new tool integrates with the standard best practice tool.

6.3.1 Enabling the complexity checks

It can vary between the various VARs (Value Adding Reseller) and ISVs which best practice checks they want to use, so each of the checks can be switched on and off. These settings are saved in the table `SysBPPParameters` and can be edited in the form `SysBPSetup`.

A new YesNo field called `CheckComplexity` has been added to `SysBPPParameters`. To display it in the form, the following code has been added to the method `buildSelectionTree` (where `tmpNode` is the parent node "Classes"):

```
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckComplexity),  
parameter.CheckComplexity);
```

Figure 6-2 shows the modified form, where the complexity node has been added below the "Classes" node. Although the complexity metrics are both on class and method level, only one checkbox has been added. This could be divided into two, which would allow having only method level checks turned on, but for simplicity only a single checkbox is used.

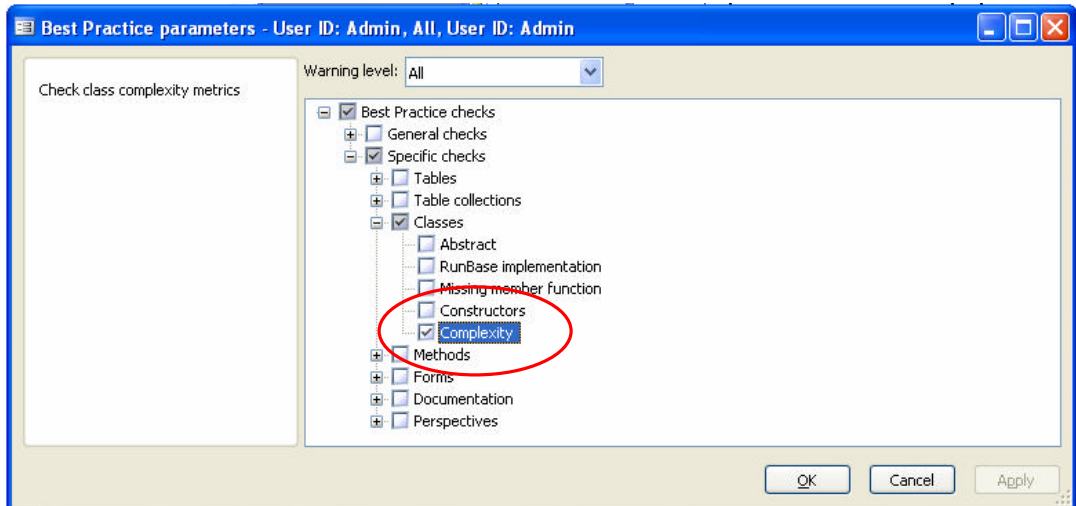


Figure 6-2 Best Practice parameters

6.3.2 Loading the metric classes

As stated in the design section, it is important that the names of the metric implementation classes are not hard coded in the BP tool, which would make adding new metrics more complicated. Since it can take a little while to find the classes which inherit from `CodeClassMetric` or `CodeMethodMetric`, a new `List` have been added to both

`SysBPCheckClassNode` and `SysBPCheckMemberFunction`. The instances of the metric implementations in these lists will then be reused to avoid too much overhead creating classes.

To fill the lists, a new utility class called `ClassInstanciator` has been created. So far, this class only has a single static method `createSubClassInstances`. It takes a variable of type `classId` as parameter and returns a `List` containing instances of classes that implements the class with the specific id.

The utility class is used in the `new` method of classes `SysBPCheckClassNode` and `SysBPCheckMemberFunction`, as shown below (from `SysBPCheckClassNode`):

```
protected void new()
{
    super();

    //Create a list that will hold instances of the metric classes
    codeClassMetricList = ClassInstanciator::createSubClassInstances
        (classNum(CodeClassMetric));
}
```

6.3.3 Performing the checks

Figure 6-3 gives an overview of which steps are involved in performing the BP complexity checks. Please note that although the figure is for class level metrics, most of it also applies to method level metrics.

The `check` method on `SysBPCheckClassNode` will be called from `SysBPCheck` (not shown). If the complexity check has been enabled (see previous section), the method `checkComplexity` is called. The source for this method is listed below:

```
void checkComplexity()
{
    CodeClassMetric codeMetric;
    ListEnumerator enum;
    str errMessage;
    xRefUpdateTmpReferences tmpUpdate;
    xRefTmpReferences tmpxRefReferences;
    ;

    //Create tmp references for the entire class (for optimization)
    tmpUpdate = new xRefUpdateTmpReferences();
    tmpUpdate.fillTmpxRefReferences(sysBPCheck.treeNode());
    tmpxRefReferences = tmpUpdate.allTmpxRefReferences();

    //Loop through all the metric classes that are available
    enum = codeClassMetricList.getEnumerator();
    while(enum.moveNext())
    {
        //Cast as CodeClassMetric
        codeMetric = enum.current();
    }
}
```

```

//Pass the tree node of the method to check
codeMetric.setElement(sysBPCheck.treeNode());

//Pass the tmp references already generated
codeMetric.setXRefTmpReferences(tmpxRefReferences);

//Perform the check
errMessage = codeMetric.getBPStr();

//If the errMessage is not empty then add a new BP message
if (errMessage != '')
{
    //Find out what to do with the message
    switch(codeMetric.getBPSeverity())
    {
        case BPSeverity::Info:

sysBPCheck.addInfo(codeMetric.getErrorCode(),0,0,errMessage);
                break;
        case BPSeverity::Warning:

sysBPCheck.addWarning(codeMetric.getErrorCode(),0,0,errMessage);
                break;
        case BPSeverity::Error:

sysBPCheck.addError(codeMetric.getErrorCode(),0,0,errMessage);
                break;
    }
}
}
}
}

```

The first thing this method does is to create the temporary cross references for the current `TreeNode` (which is a class-type node). Then the following is done for each of the class level metric implementations: First, the current `TreeNode` is passed on to the metric class by using the `setElement` method and the cross references are passed on by using the `setXRefTmpReferences` method. Then the method `getBPStr` on the metric implementation is called, and if it results in an error message, the static method `getBPSeverity` on the metric object is called to determine if an info, warning or error should be added to the list of the best practice deviations.

In `SysBPCheckClassNode` the deviations are added by using one of the methods `addInfo`, `addWarning` or `addError` from the class `SysBPCheck`. For the memberfunction checks however, the method `addSuppressableWarning` or `addSuppressableError` on `SysBPCheckMemberFunction` is used to add the deviations. Doing this enables the developers to suppress the warning or error in the code should they wish to do so.

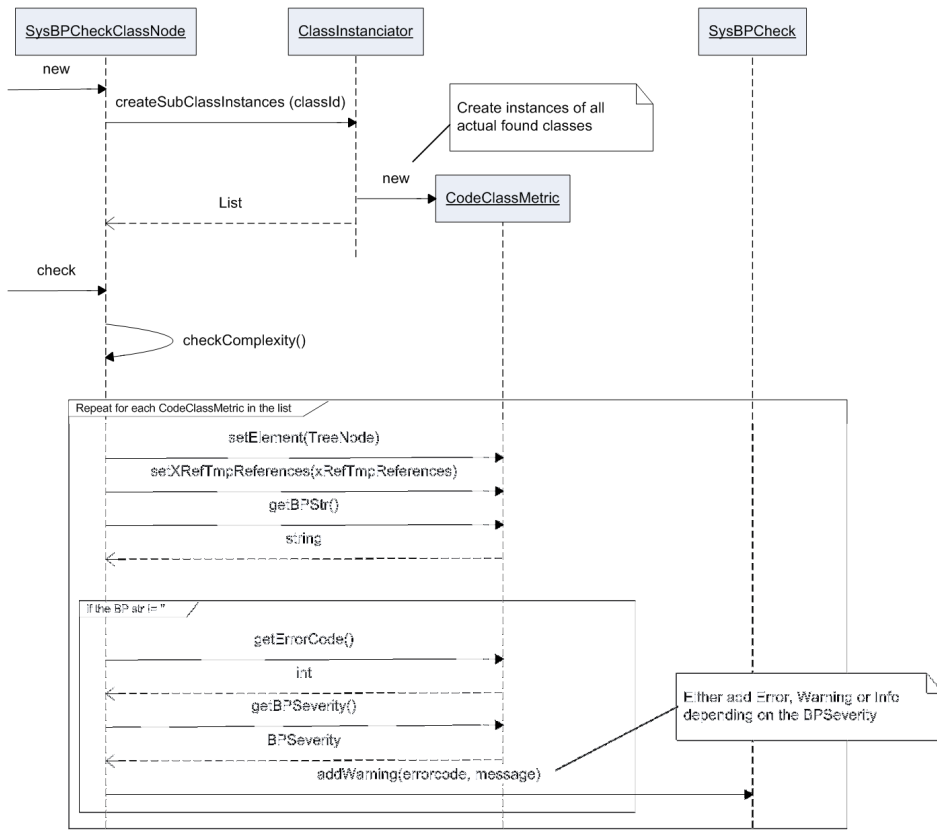


Figure 6-3 Sequence diagram for checking class node

6.4 Metric implementations

In the following sections, the code for calculating each of the ten metrics will be explained in depth.

Although the calculation part is different for each metric, the basic structures of the methods are more or less the same for all implementations. They all override three methods from `CodeMetricBase`, which basically just returns constants. The code shown below is from the class `CodeMetricSLOCMethod`, but could be from any of the implementations:

```
public str getDescription()
{
    //Source Lines of Code
    return 'SLOC';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetricSLOCMethod;
}

public BPSeverity getBPSeverity()
{
    //Warning
    return BPSeverity::Warning;
}
```

Another method that looks more or less the same is `getBPStr`. Normally this starts with calling the class' `getValue` method. Then it compares the resulting value with a predefined threshold limit from a local macro, and if necessary creates a string with the best practice message.

```
public str getBPStr()
{
    str ret;
    int slocVal;

    //Get the value for SLOC
    slocVal = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (slocVal > #MaxSLOCValue)
        ret = sprintf('The number of Source lines (SLOC) of method %1 is %2
            (Max. recommended %3)',node.nodeName(),int2str(slocVal)
            ,int2str(#MaxSLOCValue));

    return ret;
}
```

In some metrics, the main computational function is a static method which takes some kind of parameter. The reason for this is that the metric computation is used as a part of another metric. The overridden instance method `getValue`, will then create the required parameter object and then call the static method. Below is shown an example of this from the V(G) calculation.

```
int getValue()
{
    //Return the value for V(G) for the source code
    return CodeMetricVGMethod::calcVG(this.getScanner());
}
```

6.4.1 SLOC (CodeMetricSLOCMethod)

Obtaining the source code for a method is very simple, since it is just a matter of calling the method `AOTGetSource` on the current `TreeNode`. To calculate the number of source code lines, all comments must be removed from the code. When this is done, one can simply count the number of carriage returns (`'\n'`), less the number of blank lines. The primary method for calculating SLOC is shown below:

```
public static int calcSLOC(str sourcecode)
{
    int sloc;
    TextBuffer textBuffer;
    str cfcode;
    str line;
    ;

    //Create TextBuffer and fill with comment-free source code
    cfcode = CodeMetricSLOCMethod::removeComments(sourcecode);
    textBuffer = new TextBuffer();
    textBuffer.setText(cfcode);

    //Get first line
    line = textBuffer.nextToken(false, '\n');

    //Loop through lines
    while(line)
    {
        //If the line is not blank then increase SLOC
        if(strrem(line, ' ') != '')
            sloc++;

        //Read next line
        line = textBuffer.nextToken(false, '\n');
    }

    return sloc;
}
```

The above method uses a `TextBuffer` to read through the lines of comment-free source code. Each time a new non-blank line is fetched the SLOC count is increased.

To be able to remove comments from the source code, a new class `SourceCodeChunker` has been created (see Figure 6-4 for an overview of the class). As the name suggests, its job is to scan through some code to provide chunks of source code and comments. Each time the method `moveNext` is called, it will fetch the next available source code and/or comment. So, to remove comments from a piece of code, one simply can keep calling `moveNext` and `currentCodeChunk`, until `moveNext` returns false. The method `removeComments` in `CodeMetricSLOCMMethod` does exactly that, as shown in the code snippet below:

CodeMetricSLOCMMethod	SourceCodeChunker
<pre>+getValue() : int +calcSLOC(in sourcecode : str) : int +removeComments(in sourcecode : str) : str +getBPStr() : str +getErrorCode() : int +getDescription() : str +getBPSeverity() : str</pre>	<pre>-source : str -sourcelen : int -fromPos : int -linecount : int -currentCode : str -currentComment : str -startLineComment : int -startLineComment : int +new(in sourceCode : str) +moveNext() : bool -findMinPos(in vals : container) : int -findStrEnd(in sourceCode : str, in startPos : int, in quote : str) : int -resetOutput() -scanForCommentsAndQuotes() : int +codeStartLine() : int +commentStartLine() : int +currentCodeChunk() : str +currentCommentChunk() : str +lineCount() : int</pre>

Figure 6-4 Overview of `CodeMetricSLOCMMethod` and `SourceCodeChunker`

```
public static str removeComments(str sourceCode)
{
    str cfcode = ''; //Comment-free code
    SourceCodeChunker chunker = new SourceCodeChunker(sourceCode);
    ;

    //Get all code chunks
    while(chunker.moveNext())
        cfcode += chunker.currentCodeChunk();

    //Return the comment-free code
    return cfcode;
}
```

6.4.1.1 SourceCodeChunker

When scanning through the source code for comments there are a number of scenarios that need to be taken into consideration:

- **Two kind of comments**
Multi-line comments starts with `/*` and ends with `*/`.
Single-line comments starts with `//` and ends when a newline `\n` character is reached.
- **Comments in comments**
Comments might include other comments.
- **Comment-characters inside strings**
When regular comment character sequences appears inside a string (enclosed by either `"` or `'`), they should not be treated as comments.

The example method shown below (extracted from one of the unit tests) illustrates the above mentioned challenges. It furthermore includes escaped characters and quotes (also in combination with verbose strings starting with `@`), which can all pose problems when trying to find the end of a string.

```
/*Starting comment
   Comment line 2
   // */
int MyMethod()
{
    int a; //Comment here
    str s= '/* hello */ // " \' ';
    /*comment*/ int c; //Line ends with comment
    s=@'hello \';
    ;
    if (a==1)
        this.doSomething();

    //Only comment line /* more comments */
}
```

The `getNext` method of the `SourceCodeChunker` (see next page) starts by clearing the private output variables. It then calls the private method `scanForCommentsAndQuotes`, which finds the first occurrence (position) of one of the following character(s): `/*`, `//`, `'`, `"`.

If a multi-line or single-line comment is found, then the variable `currentCode` is set to contain all code from the last known end-position to the newly found position. Then the end of the comment is found by searching for either `*/` or a newline character, the comment is extracted, and the end-position is saved.

If the start of a string is found (double or single quote character), then the method `findStrEnd` is called, to find the position where the string ends. Then `scanForCommentsAndQuotes` is called again, starting at the end of the string. This will keep repeating until a comment has been found or the end of the code is reached.

```

public boolean moveNext()
{
    int scanPos;

    //Reset the output variables
    this.resetOutput();

    if(fromPos < sourcelen)
    {
        //Scan for comments and strings
        scanPos = this.scanForCommentsAndQuotes();

        //Repeat until we have found a comment
        while(scanpos > 0 && currentComment == '')
        {
            switch(substr(source,scanpos,#commentLength))
            {
                case '/*':
                    //Start of multi line comment found, so insert the
text and search for comment end
                    currentCode += substr(source,fromPos,scanPos-frompos);
                    fromPos = strstr(source,'*/',scanPos,sourcelen -
scanPos)+#commentLength;
                    currentComment = substr(source,scanPos,frompos-
scanPos);
                    break;

                case '//':
                    //Start of multi line comment found, so insert the
text and search for line end
                    currentCode += substr(source,fromPos,scanPos-frompos);
                    fromPos = strstr(source,'\n',scanPos,sourcelen -
scanPos) > 0 ? strstr(source,'\n',scanPos,sourcelen - scanPos) :
sourcelen +1;
                    currentComment = substr(source,scanPos,frompos-
scanPos);
                    break;

                default:
                    //All text until the next quote pos will be included,
regarding if it is a comment
                    scanPos = this.findStrEnd(source,
scanPos+1,substr(source,scanpos,1),substr(source,scanpos-1,1));
                    currentCode += substr(source,fromPos,scanPos-
fromPos+1);
                    fromPos = scanPos + 1;
            }

            //Rescan
            scanPos = this.scanForCommentsAndQuotes();
        }

        if (currentComment == '')
        {

```

```

        //No comments was found, so we must copy the last part of the
sourcecode to the currentCode
        currentCode += substr(source,fromPos,sourceLen-fromPos+1);
        fromPos = sourceLen;
    }

    //Add to the linecount
    lineCount += StringUtil::CountOccurrences(currentCode,'\n');
    startLineComment = lineCount;
    lineCount += StringUtil::CountOccurrences(currentComment,'\n');

    return true;
}

return false;
}

```

6.4.2 CP (CodeMetricCPMethod)

To calculate the comment percentage, three numbers are needed: Total number of lines in source code, number of blank lines and number of lines containing comments. By using the `SourceCodeChunker` these can be obtained quite easily.

The method `calcCP` of class `CodeMetricCPMethod` (see code on next page) starts by initializing a new `SourceCodeChunker`. Then it keeps calling the `moveNext` method of the chunker, until it returns false and the end of the source code is reached. Each time a comment is fetched, the position of it is evaluated to find out if the comment is on the same line as a previous comment, and thus if it should add to the number of comment lines.

To find the number of blank lines in a code chunk, all spaces are removed from it and the number of '\n\n' character sequences is counted. Since no standard functionality for counting occurrences in a string exists, a new class `StringUtil` with the static method `CountOccurrences` has been created. This method just uses the build-in method `strscan` to find the wanted sequence, and each time this happens, an integer variable is increased.

```

public static int calcCP(str sourceCode)
{
    int cp;
    int newlinesInComment;
    int linesWithComments;
    int blankLines;
    int lastCommentLine;

    str tmp;

    SourceCodeChunker chunker = new SourceCodeChunker(sourceCode);
    ;

    //Loop through code/comment chunks
    while(chunker.moveNext())
    {
        if (chunker.currentCommentChunk() != '')

```

```

    {
        newlinesInComment =
StringUtil::CountOccurrences(chunker.currentCommentChunk(), '\n');

        if(chunker.commentStartLine() > lastCommentLine)
            linesWithComments += newlinesInComment + 1;
        else
            linesWithComments += newlinesInComment;

        lastCommentLine = chunker.commentStartLine() +
newlinesInComment;
    }

    //Remove spaces from the source code chunk
    tmp = strrem(chunker.currentCodeChunk(), ' ');

    //Add the number of blank lines in the chunk
    blankLines += StringUtil::CountOccurrences(tmp, '\n\n');
}

//Calculate CP
if((chunker.lineCount() - blankLines) > 0)
    cp = (linesWithComments / (chunker.lineCount() - blankLines))*100;

return cp;
}

```

6.4.3 V(G) (CodeMetricVGMethod)

The primary method of the class `CodeMetricVGMethod` is `calcVG`. This static method takes an instance of a `SysScannerClass` as an argument, and returns the Cyclomatic complexity value.

The scanner class provides a simple way of obtaining tokens from the source code of a `TreeNode`. For each token, both a symbol number and the actual text can be retrieved. All symbol numbers have been predefined in the macro `TokenType`s, which makes it relatively easy to decode the numbers.

As described in the theory section and in the functional specification, it is not necessary to build the entire control flow graph when calculating $V(G)$, so a scan for certain combinations of symbols/keywords will be enough to find the loops and branches. Table 6-1 gives a list of the keyword combinations that are scanned for. Each of the combinations will add one to the complexity count. Please note that although the table shows the actual keywords, the symbol number is used instead in most cases.

Keyword combination	Conditions
?	Not inside a SQL statement
&&	
<st_end> <type> <identifier> (The <type> must be a simple datatype, void or the name of a TreeNode object.
<st_end> if	
<st_end> while	
<st_end> for	
<st_end> case	
<st_end> default	
<st_end> try	
else if	
join <identifier>	Must not be prefixed by “exists” or “notexists”

Table 6-1 List of keywords

The <st_end> denotes the beginning of a new statement. This is actually found by searching for the end of a previous statement or the beginning of a new block, indicated by the symbols “{”, “}” or “;”

To prevent the `calcVG` method from becoming too big and complex, many of the symbol combination checks has been split out into separate static functions. Since up to four symbols are needed for detecting the combinations, a list of the four previous read symbols and strings are preserved. These historical values can then be passed on to the functions as needed. Below is an example of the function that determines if the current symbol (`symbol_1`) is within a SQL statement. The parameter `isSQL` will normally be the result of the last call to the function.

```
public static boolean isSQLStatement(boolean isSQL, int symbol_1, int
symbol_2)
{
    boolean ret = isSQL;
    ;

    if (isSQL && (symbol_1 == #LEFTBR_SYM || symbol_1 == #SEMICOLON_SYM))
    {
        //The SQL statement has ended
        ret = false;
    }
    else if(!isSQL)
    {
        //Its the first word of an expression
        if(CodeMetricVGMethod::isStatementBeginEnd(symbol_2))
        {
            //Its a SQL symbol
            switch(symbol_1)
            {
                case #SEARCH_SYM: //select
                case #DELETE_SYM: //delete
```

```

        case #UPDATE_SYM: //update_recordset
        case #INSERT_SYM: //insert_recordset
            ret = true;
        }
    }
    //while select
    else if(symbol_2 == #WHILE_SYM && symbol_1 == #SEARCH_SYM)
        ret = true;
}

return ret;
}

```

6.4.4 WMC (CodeMetricWMC)

In the functional specification it was decided that the Weighted Methods for Class metric should use the Cyclomatic Complexity. So, since $V(G)$ is already implemented, calculating WMC can simply be done by looping through all methods on a class and summing up the complexities.

Below is shown the overridden method `getValue` of class `CodeMetricWMC`:

```

int getValue()
{
    CodeMetricVGMethod vgMetric = new CodeMetricVGMethod();
    int sumVG = 0;
    TreeNode child;
    ;

    //Loop through all child methods
    child = node.AOTfirstChild();
    while(child)
    {
        if (child.treeNodeName() != 'classDeclaration')
        {
            //Pass the method to CodeMetricCCMethod
            vgMetric.setElement(child);

            //Get the value
            sumVG += vgMetric.getValue();
        }

        //Get next child method
        child = child.AOTnextSibling();
    }

    //Return sum of complexities
    return sumVG;
}

```

6.4.5 DIT (CodeMetricDIT)

The implementation of the Depth of Inheritance Tree metric uses the build-in `DictClass`. The `DictClass` can provide a number of different metadata of a “pure” code class: if it is an abstract class, which static and object methods it has and, what’s most interesting in this case, which class it directly extends. To find the total depth of the tree, we must keep iterating through the parent classes until the top class is reached. The depth is initialized to one since all classes implicit inherit from `object`. This however, means that if `object` is explicitly stated then we should not add an extract to the count. The source code for the `getValue` function is listed below.

```
public int getValue()
{
    DictClass dict = new DictClass(node.applObjectId());
    int depth = 1; //All classes inherit from Object
    ;

    //Repeat as long as we can go up in the hierarchy
    while(dict.extend())
    {
        //Increase depth if its not object
        if (dict.extend() != classNum(object))
        {
            depth++;
        }

        //Create a DictClass for the parent
        dict = new DictClass(dict.extend());
    }

    return depth;
}
```

6.4.6 NOC (CodeMetricNOC)

Obtaining the Number Of Children can also be done by using the `DictClass`. This is a matter of creating a new instance of the `DictClass` and then calling the method `extendedBy`. It will return a list of all classes that extends the class, both direct and indirect descendants. Each of the nodes in the list is then examined further, and if it is a direct descendant then one is added to the NOC count. The `getValue` method of `CodeMetricNOC` is shown below.

```
int getValue()
{
    DictClass dict;
    DictClass subDict;
    Enumerator enum;
    int noc = 0;
    ;
    //Create a new dict class
    dict = new DictClass(node.applObjectId());
    //Get an enumerator containing all subclasses
```

```

enum = dict.extendedBy().getEnumerator();

//Loop through all subclasses
while(enum.moveNext())
{
    subDict = new DictClass(enum.current());

    //If the class in an immediate child then increase the count
    if (subDict.extend() == node.appObjectid())
        noc++;
}
return noc;
}

```

6.4.7 CBO (CodeMetricCBO)

To find the amount of Coupling Between Objects the temporary cross references table is used. All references of type `xRefReference::Read` are evaluated, and the name of the object is taken from either the `ParentName` field or, if there is no parent, the `name` field. To make sure each object is only counted once, the found object names are kept in a `Map`. A `Map` is like a hash table where the key field can be of an arbitrary type. The map is then queried to see if the name already exists, otherwise it is inserted into the map. This way, when all records in the table have been processed, the CBO count equals the number of elements in the map. The `getValue` method of `CodeMetricCBO` is shown below:

```

int getValue()
{
    xRefTmpReferences thisReferences;
    Map map;
    str typeName;
    ;

    //Make sure xRef is updated for this class
    this.initTmpXRef();

    //Create a map for holding the type names
    map = new Map(Types::String,Types::String);

    //Get the paths of the objects used
    thisReferences.setTmpData(tmpxRefReferences);
    while select thisReferences where thisReferences.Reference ==
xRefReference::Read
    {
        //Get the type name (path)
        if (thisReferences.ParentName == '')
            typeName = thisReferences.name;
        else
            typeName = thisReferences.ParentName;

        //If the type does not already exists in the map then insert it
        if (!map.exists(typeName))
            map.insert(typeName,typeName);
    }
}

```

```

    //CBO = number of distinct types
    return map.elements();
}

```

6.4.8 RFC (CodeMetricRFC)

Computation of the Response For Class is somewhat similar to CBO, since it also uses the temporary cross references table and a map. In this case the map just holds objectname\methodname, and uses references of type `xRefReference::Call`. As the response set should include the class' own methods, an additional loop has been added, where the `DictClass` is used for iterating through the class' methods and inserting their names into the map.

6.4.9 LCOM (CodeMetricLCOM)

The LCOM metric as defined by [Hitz95] is the number of connected components in a class. Figure 6-5 shows an example of how the functions in a class might be connected to each other. The class has three variables (`a,b,c`) and four methods (`f,g,h,x`). The arrows in the figure represent usage/call of other variables or methods. Note, that when determining which components are connected in the LCOM metric, the direction of the relation does not matter.

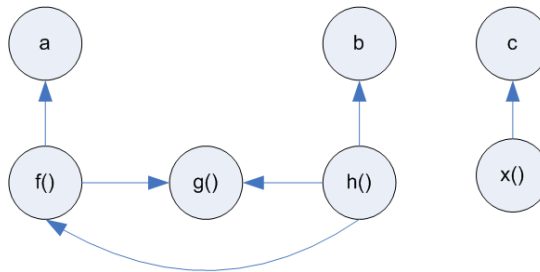


Figure 6-5 LCOM directed graph

It is intuitively clear that the LCOM of Figure 6-5 must be two, since there are two sets of components. To implement this distinction in code however, some kind of undirected graph algorithm is needed to detect how many separate sub-graphs the graph is made from. The Depth First Search (DFS) graph algorithm is ideal for this purpose, as it will traverse through all nodes in the graph and record each node's parent node. After the DFS has completed, the LCOM number will be equal to the number of nodes without a parent node.

Since there were no existing classes in Dynamics AX for representing graphs, the three new classes shown in Figure 6-6 have been implemented.

GraphNode	GraphEdge	GraphUndirected
-data : anytype -color : int -timeDiscovered : int -timeFinished : int -parent : GraphNode +getData() : anytype +setData(in _data : anytype) +getColor() : int +setColor(in _color : int) +getParent() : GraphNode +setParent(in _parent : GraphNode) +getTimeDiscovered() : int +setTimeDiscovered(in _timeDiscovered : int) +getTimeFinished() : int +setTimeFinished(in _timeFinished : int)	-node1 : GraphNode -node2 : GraphNode +getNode1() : GraphNode +setNode1(in _node1 : GraphNode) +getNode2() : GraphNode +setNode2(in _node2 : GraphNode)	-nodes : List -edges : List -dfsTime : int +addEdge(in node1 : GraphNode, in node2 : GraphNode) : GraphEdge +addNodes(in data : anytype) : GraphNode +getNodes() : List +findNodeOnData(in findData : anytype) : GraphNode +findEdge(in node1 : GraphNode, in node2 : GraphNode) : GraphEdge +getListOfNeighbours(in node : GraphNode) : List -DFS(in node : GraphNode) +runDFS() +nodesWithoutParent() : int

Figure 6-6 Graph classes

A `GraphNode` is simply a data container which can carry some payload (`data`). In addition it has some instance variables that are needed when performing the DFS routine. A `GraphEdge` connects two nodes. The endpoints of the edge are simply called `node1` and `node2`, since no specific direction is needed. The `GraphUndirected` contains a list of nodes and edges and has methods for executing the DFS (`runDFS`). Since the implementation of the DFS is standard text book material from [Cormen01] it will not be further explained here. The only two non-standard methods that `GraphUndirected` has are `findNodeOnData`, which will search for a node in the graph based on the node's data, and `nodesWithoutParent`, which will return the number of nodes that has no parent.

To build the graph, the `getValue` method of `CodeMetricLCOM` uses the temporary cross references. Each reference is treated as follows: If it is the definition of a class level variable or the definition of a non-static method then a new node is added to the graph. If the reference is a call to an internal method or if it is a read/write of a class level variable, then a new edge is added. The code for the `getValue` method is listed below.

```
int getValue()
{
    GraphUndirected graph = new GraphUndirected();
    xRefTmpReferences thisReferences;

    str graphNodeVal;
    GraphNode fromGraphNode;
    GraphNode toGraphNode;

    //Make sure xRef is updated for the class
    this.initTmpXRef();

    thisReferences.setTmpData(tmpxRefReferences);
    while select thisReferences order by Reference
    {

        //Declaration of class level variables so add node
        if(this.isClassLevelVar(thisReferences))
        {
            graphNodeVal = thisReferences.name;
            graph.addNode(graphNodeVal);
        }
    }
}
```

```

//Definition of class method so add node
else if(this.isMethodDef(thisReferences))
{
    graphNodeVal = thisReferences.Path;
    graph.addNode(graphNodeVal);
}
//Call to class method so add edge
else if(this.isInternalMethodCall(thisReferences))
{
    fromGraphNode = graph.findNodeOnData(thisReferences.Path);
    toGraphNode = graph.findNodeOnData(node.treeNodePath() + '\\\'
+ thisReferences.name);

    //If toGraphNode is null then it is a call to an inherited
method, else it is a regular internal method call
    graph.addEdge(fromGraphNode,toGraphNode);
}
//Read or write of variable
else if(thisReferences.Reference == xRefReference::Read ||
thisReferences.Reference == xRefReference::Write)
{
    //If the variable can be found as a node, then it must be a
class-level variable
    toGraphNode = graph.findNodeOnData(thisReferences.name);
    fromGraphNode = graph.findNodeOnData(thisReferences.Path);

    graph.addEdge(fromGraphNode,toGraphNode);
}
}

//Start a Depth First Search on the graph
graph.runDFS();

//LCOM = the number of connected components = the number of sub-graphs
return graph.nodesWithoutParent();
}

```

6.4.10 FI (CodeMetricFI)

As the only of the chosen metrics, Fan-In is a system level metrics. As mentioned in the theory section 3.3.2.7, computing this metric requires that all relations in the code have been established. In the other class level metrics, the temporary references were used, but since they are only created on a per-class basis, we need to use the full-blown cross reference tables here. The problem with using these tables is that we cannot be sure that they are up-to-date or if the cross references have been created at all. Of course, the entire cross references could be created each time the FI metric is computed, but since this operation would take 3-4 hours to complete each time, this is not a feasible solution. Due to this, it is assumed that the cross references are up-to-date when the `CodeMetricFI` needs it, and it will then be up to the users to make sure that this is so.

When the cross references are at hand, it is quite simple to find out which other classes have references to a class. As can be seen in the source code below, it can be done by

building a select statement that finds the references where the path includes the class' treenode path.

```

int getValue()
{
    xRefReferences  xReferences;
    xRefPaths      xPaths;
    xFromPaths     xFromPaths;
    toLikePath     toLikePath;

    str            typeName;
    Map            map;
    ;

    //Create a map for holding the type names
    map = new Map(Types::String,Types::String);

    //Add \* in the end of the path for node to find, and double the amount
of \
    //This is needed to make the "like" work correctly
    toLikePath = strReplace(node.treeNodePath() + '\\*', '\\\\', '\\\\\\');

    /* Since Fan-In is a system-level measure, we need to use x-ref from
the normal tables,
    and not from the temporary xref
    */
    while select xFromPaths
    join xReferences where xFromPaths.RecId == xReferences.xRefPathRecId
    &&
        (xReferences.Reference == xRefReference::Declaration
    ||
        xReferences.Reference == xRefReference::Call)
    join xPaths where xPaths.RecId == xReferences.referencePathRecId &&
        (xPaths.Path == node.treeNodePath() ||
        xPaths.Path like toLikePath
        )
    {
        //Get the name of the class/form/table
        typename = SysTreeNode::applObjectPath(xFromPaths.Path);

        //Insert the found type(class) name into the map if it's not
already there
        //and if it is not the class itself
        if (!map.exists(typeName) && typeName != node.treeNodePath())
            map.insert(typeName, typeName);
    }

    //FI = number of other types having a reference to this class
    return map.elements();
}

```


6.5 Statistics generation

The following sections provide details of the Tables, Classes and Form used to implement the metric statistics.

6.5.1 TmpCodeMetrics (table)

This table is used for storing the raw values for each of the measurements being made. Since the metrics operates on application code with are shared between all companies in the system, the data in the `TmpCodeMetrics` table is not saved per company.

To follow the Best Practice guidelines, new extended data types have been created for each of the fields. By using the extended data types we make sure that the same logical type of information stored in different tables also will have the same format, length, display adjustment and so on.

Field name	(Extended) Data type	Default value
TreeNodePath	TreeNodePath (str 400)	''
Metric	Metric (str 10)	''
Value	MetricValue (int)	0

6.5.2 TmpCodeMetricsTeamStat (table)

This table stores statistics values summed up per Metric, Team and Prefix.

Field name	(Extended) Data type	Default value
Metric	Metric (str 10)	''
Team	TeamName (str 25)	''
Prefix	PrefixName (str 50)	''
Occurences	MetricOccurences (int)	0
MinValue	MetricValue (int)	0
MaxValue	MetricValue (int)	0
ValueSum	MetricValue (int)	0
AverageValue	MetricAverage (real)	0.0

6.5.3 CodeMetricGenerator (class)

The purpose of this class is to get the metric values for all classes/methods from a given starting point, and insert the values into the table `TmpCodeMetrics`. The main static method is called `generateMetrics`, and takes a start `TreeNode` as parameter. This method starts by creating two lists containing instances of the available class- and method-level metrics. This is done by using the `ClassInstanciator::createSubClassInstances` method, the same way as in the BP classes. Then a `TreeNodeTraverser` is used to iterate through all the treenode's subnodes. Depending of the type of the node, either the `doMethodMetric` or the `doClassMetric` static method is called with the appropriate metric list as argument.

```

public static void generateMetrics(TreeNode startnode)
{
    //Create lists with instances of CodeMethodMetric/CodeClassMetric
    classes
        List codeMethodMetricList =
ClassInstanciator::createSubClassInstances(classNum(CodeMethodMetric));
        List codeClassMetricList =
ClassInstanciator::createSubClassInstances(classNum(CodeClassMetric));

    TreeNode treeNode;
    TreeNodeTraverser treeNodeTraverser;

    #avifiles
    SysOperationProgress simpleProgress;
    ;

    //Create a progress indicator
    simpleProgress = SysOperationProgress::newGeneral(#aviUpdate,
'Metrics', startnode.AOTchildNodeCount());

    //Traverse the startnode
    treeNodeTraverser = new TreeNodeTraverser(startnode);
    while (treeNodeTraverser.next())
    {
        //Get the current node
        treeNode = treeNodeTraverser.currentNode();

        //Increment and set text on progress
        simpleProgress.incCount();
        simpleProgress.setText(treeNode.treeNodePath());

        //Perform different actions depending on the type of TreeNode
        switch (treeNode.handle())
        {
            case classnum(MemberFunction):
                if (treeNode.treeNodeName() != 'classDeclaration')
                    CodeMetricGenerator::doMethodMetric(treeNode,
codeMethodMetricList);
                break;
            case classnum(ClassNode):
                CodeMetricGenerator::doClassMetric(treeNode,
codeClassMetricList);
                break;
        }
    }

    //Done!!
}

```

The functionality of methods `doMethodMetric` and `doClassMetric` are very similar to the `checkComplexity` method of the `BPCheckMemberFunction` and `BPCheckClassNode`, since in both cases all metric classes in the list is looped through, and passed the `TreeNode` and the scanner or cross references. The common job of actually retrieving the value and

inserting the information into the table is handled by the static method `saveInDB`, which must have the metric instance and the path to the `TreeNode` passed on. The source code for `saveInDB` is shown below:

```
public static void saveInDB(CodeMetricBase codeMetric, TreeNodePath path)
{
    TmpCodeMetrics tmpCodeMetrics;
    ;

    //Perform the check
    tmpCodeMetrics.Value = codeMetric.getValue();

    //Add standard info and insert into the table
    tmpCodeMetrics.Metric = codeMetric.getDescription();
    tmpCodeMetrics.TreeNodePath = path;
    tmpCodeMetrics.insert();
}
}
```

6.5.4 CodeMetricTeamStatGenerator, CodeMetricStatItem (class)

The purpose of the class `CodeMetricTeamStatGenerator` is to group the raw data from the table `TmpCodeMetrics` per metric/team/prefix. The information is saved in a datastructure as shown in Figure 6-7. It consists of an outer map, where the key is the name of the metric. Inside that map is another map, which has the prefix name as key. This inner map stores elements of the class `CodeMetricStatItem`. The reason for using the Map datastructure, is that it allows for fast lookups, which is a necessity since there might be a lot of raw data to be processed (currently some 419.000 records).

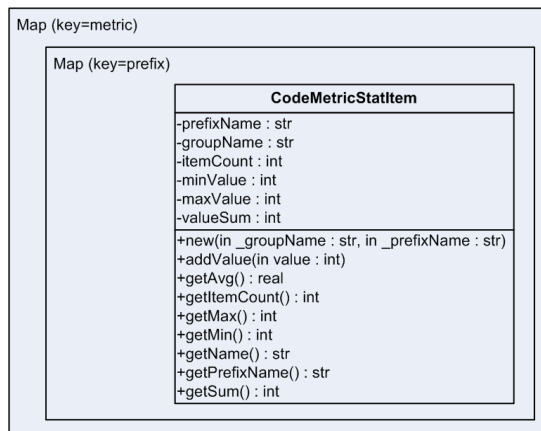


Figure 6-7 Temporary storage of team statistics

To start the generation of team statistics, the static method `statByTeam` on `CodeMetricTeamStatGenerator` must be called with the filename of the file containing combinations of team and prefix names (as explained in section 5.3). The information from

the file is loaded into a Map by the method `loadPrefixMap`. This map is then passed on to `initStatMap` which will use it to initialize the data structure from Figure 6-7.

For each record in the `TmpCodeMetrics` table, the prefixmap is searched to find the best matching prefix. This prefix, along with the metric name, can then be used to lookup the correct `CodeMetricStatItem` from the data structure. The method `addValue` on the item will then be called with the measured value, so the `minValue`, `maxValue`, `valueSum` and `itemCount` can be updated.

```
public static Map statByTeam(str _teamFileName)
{
    //Load map with prefix/team pairs from file
    Map teamPrefixMap =
CodeMetricTeamStatGenerator::loadPrefixMap(_teamFileName);

    //Get map to hold maps of CodeMetricStatItems per team per metric
    Map statMap =
CodeMetricTeamStatGenerator::initStatMap(teamPrefixMap);
    Map metricMap;
    CodeMetricStatItem statItem;

    str path = '';
    str team;
    str prefix;

    TmpCodeMetrics result;

    #avifiles
    SysOperationProgress simpleProgress;
    ;

    //Create a progress indicator
    select count(value) from result;
    simpleProgress = SysOperationProgress::newGeneral(#aviUpdate,
'Statistics', result.Value);

    //Loop through all records in tmpCodeMetrics to decide which
prefix/metric map they should be added to
    while select result order by TreeNodePath, Metric
    {
        if (result.TreeNodePath != path)
        {
            //Save the path
            path = result.TreeNodePath;

            //Find the team name from prefix map
            prefix = CodeMetricTeamStatGenerator::findPrefix(path,
teamPrefixMap);
        }

        //Increment and set text on progress
        simpleProgress.incCount();
        simpleProgress.setText(path);
    }
}
```

```

//Get the map for the metric (ie. SLOC)
metricMap = statMap.lookup(result.Metric);

//Get statItem from prefix
statItem = metricMap.lookup(prefix);

if (statItem != null)
{
    //Update item
    statItem.addValue(result.Value);
}
}

return statMap;
}

```

6.5.5 CodeMetricResults (form)

The form `CodeMetricResults` displays data from the two statistics tables. It has two tabs: Raw data (`TmpCodeMetrics` table) and Team Statistics (`TmpCodeMetricsTeamStat` table).

On the “Raw data” tab a generation of values can be started by selecting a start node from the drop down and the pressing the button “Start generation”. When the button is pressed the form method `startGeneration` will be called. This starts by deleting all data from the `TmpCodeMetrics` table, and then calls `CodeMetricGenerator::generateMetrics` which does the actual work. When that has completed the grid’s data source is refreshed to show the new values. Figure 6-8 shows what the “Raw data” tab looks like (in this case a user filter has been applied to the grid).

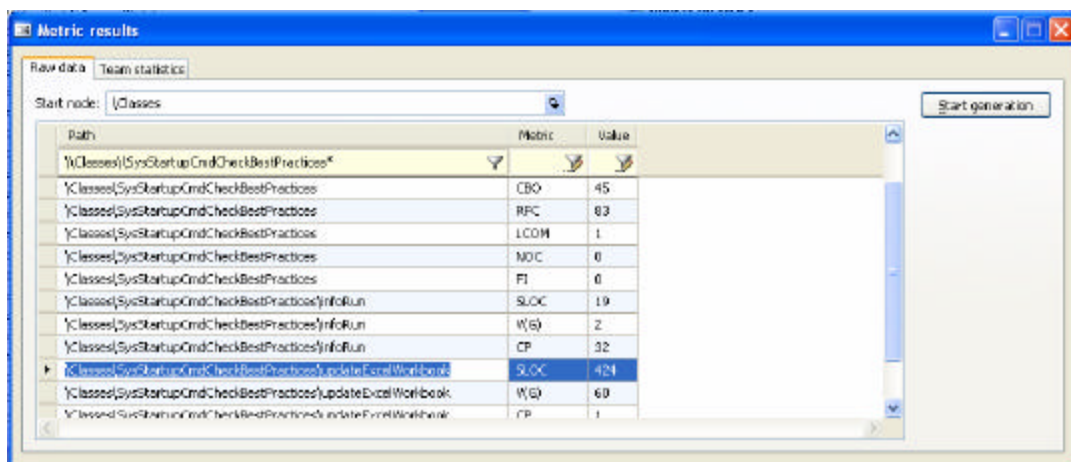


Figure 6-8 Metric results - Raw data

Figure 6-9 shows the “Team statistics” tab of `CodeMetricResults`. This has a button that will trigger generation of the team/prefix statistics, based on the selected prefix file. After the `CodeMetricTeamStatGenerator::statByTeam` method returns, the mentioned data structure is inserted into the `TmpCodeMetricsTeamStat`, by looping through the items in the two nested maps, and the grid is refreshed to show the new data.

The screenshot shows a window titled "Metric results" with two tabs: "Raw data" and "Team statistics". The "Team statistics" tab is active, displaying a table with the following data:

Metric	Team	Prefix	Occurrences	AverageValue	MinValue	MaxValue
Y(G)						
Y(G)	Project	Proj	9711	2,56	1	98
Y(G)	SQM	Purch	2715	2,32	1	52
Y(G)	Project	PurchCreateOrderForms...	3	1,33	1	2
Y(G)	Project	PurchLineType_Project	33	1,64	1	4
Y(G)	Project	PurchTableForm_Project	5	1,60	1	2
Y(G)	Project	PurchTableType_Project	48	1,23	1	3
Y(G)	SQM Tech	Query	1	1,00	1	1
Y(G)	SQM Tech	Queue	5	1,20	1	2
Y(G)	SQM Tech	Random	15	3,40	1	14
Y(G)	AID	Release	601	4,36	1	108

On the right side of the window, there is a button labeled "Generate team statistics".

Figure 6-9 Metric results - Team statistics

Chapter 7 Test

This chapter describes what has been done to verify that the new tool work as intended and that the metrics are computed correct according to the theory. Please refer to Appendix C for instructions of how to install the tool.

7.1 Unit tests

As mentioned in section 2.2, Test Driven Development has been used for this project. This has led to the development of 21 unit test classes with a total of 58 test methods. As the complexity project contains 25 non-test non-form classes, tests for four classes are missing: `CodeMetricBase`, `CodeClassMetric`, `CodeMethodMetric` and `SysBPCheckBase`. The first three framework classes are abstract, and such it is impossible to instantiate them directly. Although `SysBPCheckBase` is not declared abstract and thus could be instantiated, it does not make sense to test it directly. However, the methods that the four classes contain have all been indirectly tested, since they are used by some of the classes which have been tested.

No unit tests have been created for the form `CodeMetricResults`, since it can be very difficult to write code that tests how the graphical user interface works. The methods on the form rely on functionality from the “pure” code classes, which have already been tested, so the primary purpose of a test of the form is to confirm that the code classes and methods are invoked correct.

Not every class method has been given its own test method. This is because some of the methods rely on data being setup, and as such it does not make sense to do a stand-alone test. One example of this is the method `testAddValue` from the test class `CodeMetricStatItemTest`, which is shown below. This tests the interaction between the `addValue` method and the “get” methods like `getAvg`.

```
void testAddValue()
{
    //Create new item
    CodeMetricStatItem statItem = new
CodeMetricStatItem('group', 'prefix');

    //Check that no values are added, and that the initialize values are
correct
    this.assertEquals(0,statItem.getItemCount(),"Zero items should be
added");
    this.assertEquals(0,statItem.getAvg(),"Average should be 0");
    this.assertEquals(0,statItem.getMax(),"Max value should be 0");
    this.assertNotEqual(0,statItem.getMin(),"Min value should not be 0");
    this.assertEquals(0,statItem.getSum(),"Sum should be 0");

    //Add the first value
    statItem.addValue(100);
```

```

    //Check that the correct values are computed
    this.assertEquals(1,statItem.getItemCount(),"One item should be
added");
    this.assertEquals(100.00,statItem.getAvg(),"Average should be 100");
    this.assertEquals(100,statItem.getMax(),"Max value should be 100");
    this.assertEquals(100,statItem.getMin(),"Min value should be 100");
    this.assertEquals(100,statItem.getSum(),"Sum should be 100");

    //Add another value
    statItem.addValue(200);

    //Check again
    this.assertEquals(2,statItem.getItemCount(),"Two items should be
added");
    this.assertEquals(150.00,statItem.getAvg(),"Average should be 150");
    this.assertEquals(200,statItem.getMax(),"Max value should be 200");
    this.assertEquals(100,statItem.getMin(),"Min value should be 100");
    this.assertEquals(300,statItem.getSum(),"Sum should be 300");
}

```

During the development two Dynamics AX XUnit tools have been used: One is the XUnitToolbar and the other is the XUnitTestBrowser. The toolbar is very handy for repeating running a single test, while the test browser can be used for running all the unit tests at once. Figure 7-1 shows the result of all the unit tests, where it can be seen that all 58 tests have completed with success.

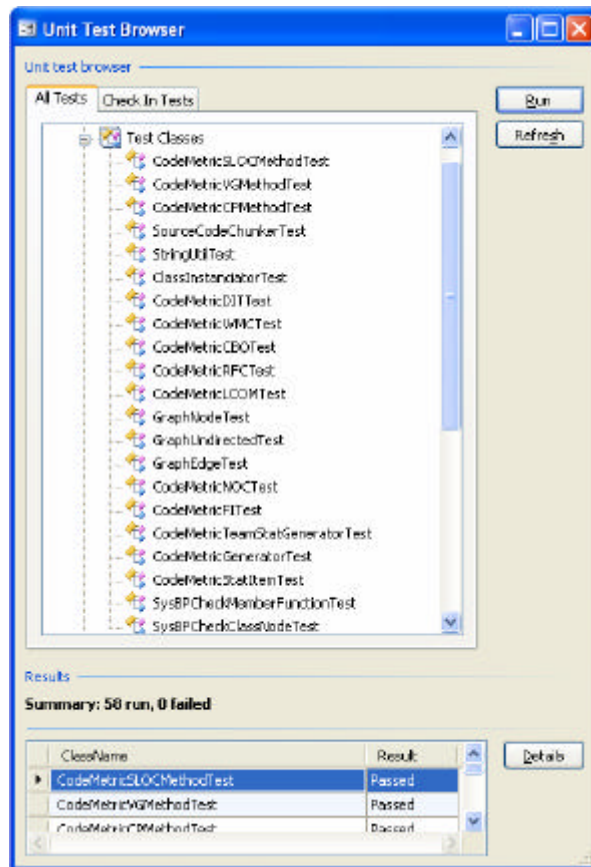


Figure 7-1 Results of unit tests

7.2 Functional test

To find out if the functionality of the new tool is correct, all items stated in the functional specification must be verified. The table below shows how each of the high level requirements have been fulfilled or implemented.

Number	Solution
0010	The developer is able to select if the complexity check will be included in the BP check by selecting/deselecting the "Complexity" node in the tree in the "Best Practices parameters" form.
0020	The complexity checks supports all language constructs in Dynamics AX version 4.0
0030	Both traditional (method level) and OO based (class level) metrics have been implemented.
0040	Outputs are shown in the best practice tab in the compiler output window.
0050	The output from BP can be machine post-processed by using the error code to distinguish between the metrics.
0060	The warnings generated by the BP Complexity checks will automatically be included in the Best Practice Excel sheet, since this just selects all warnings from the <code>SysCompilerOutput</code> table.
0070	Metric values can be extracted by using the "Metric results" form. Values can be grouped per metric/team/prefix level.

Of all the items in the functional specification, there is one that has not been fulfilled, and that has to do with the performance of the tool. It was the goal that a complexity-only best practice run on the entire AOT should take no more than 45 minutes on a 3 GHz computer with 1 GB RAM. A test of this has showed that it takes around 3½ hours to run on a laptop with a 1,6 GHz processor and 1 GB RAM, so although it was a somewhat slow processor, the requirement is not likely to be met. This speed requirement was set by me as a qualified guess, so further end-user investigation is needed to find out if the current speed is fast “enough” or if optimizations of the tool are necessary.

7.3 Adherence to own rules

As stated in the functional specification, the code for the new tool should of course not generate any complexity best practice warnings, errors or info messages. To verify this, a best practice check has been started from the top node of the new project. However, doing this revealed a lot of best practice deviations, 987 in total. These will be explained further below.

The vast majority of the deviations come from the following areas:

- Use of single quoted texts and constants, mostly in the unit tests (361 deviations).
- Classes prefixed with `CodeMetricDummy` resulted in 115 deviations. This is intentional since they are used to test that the metrics checks works correctly.
- 207 of the deviations came from the modified objects, but not from the methods that have been added or changed as part of this tool.
- Missing labels and help texts on the two tables `TmpCodeMetrics` and `TmpCodeMetricsTeamStat` and the form `CodeMetricResults` (52 deviations).
- 56 warnings because the test methods are not directly referenced by anyone.
- 77 messages of how to set method availability to private
- 93 misc. info messages

The last 26 messages are violations of the LCOM metric. All of the unit tests which have more than one method (14) will get `LCOM > 1`, due to the way the unit test framework is constructed. Each of the test methods in a `xUnitDevTest` class must be independent of each other, so the data cohesion of the class is of course very low. Having a `LCOM > 1` indicates that the test classes should be split into multiple smaller classes. Although this could be done, the semantic coherence of the unit test classes is still valid so there is actually no need to split them up.

`LCOM` is 5 and 2 for the two new classes `GraphNode` and `GraphEdge`. As mentioned in the theory (section 3.3.2.3), `LCOM` has a problem with classes that acts purely as data containers, which is exactly what `GraphEdge` and `GraphNode` do, so these can also safely be ignored.

Each of the 10 metric implementations also gets `LCOM` deviations, because of the overridden methods like `getDescription`, which does not operate on any instance variables. These methods are semantic correct so they can also be ignored.

The conclusion to this test is that the new tool adheres nicely to the best practice rules, both the existing and the new complexity related rules.

Chapter 8 Analysis of results

To extract some statistics about the various metrics, I have used the new form “Metric results” to start a generation of metrics for all objects in the entire AOT. This took around 3½ hours to complete on a regular Laptop with 1GB RAM, and resulted in more than 419.000 measurements being inserted into the `TmpCodeMetrics` table.

8.1 Results overview

An overview of the results can be seen in Table 8-1. The three method level metrics CP, SLOC and V(G) have been run on a total of 127.650 methods. As mentioned in the functional specification, the method level metrics are not limited to “pure” class methods, but also include methods on Tables, Forms and so on. The class level metrics WMC, DIT, NOC, CBO, RFC, LCOM and FI have been executed on 5.162 pure code classes.

Metric	Range	Count	Avg.	Max.	Violations	
					Count	%
SLOC	[1;40]	127.650	13,04	1.152	6.709	5,26
CP	[10;100]	127.650	4,41	98	101.684	79,66
V(G)	[1;10]	127.650	2,54	358	3.912	3,06
WMC	[1;50]	5.162	35,98	1.280	950	18,40
DIT	[0;8]	5.162	2,34	8	0	0,00
NOC	[0;10]	5.162	0,73	344	42	0,81
CBO	[0;20]	5.162	21,43	232	1.903	36,87
RFC	[1;50]	5.162	36,84	548	1.095	21,21
LCOM	[1]	5.162	4,75	123	3.496	67,73
FI	[1;50]	5.162	6,36	5.137	76	1,47

Table 8-1 Results overview

8.2 Details

This section will go into detail with the results for each of the metrics implemented. Some of the details of the results have been found by creating various SQL queries against the `TmpCodeMetrics` table, while others come from the table `TmpCodeMetricsTeamStat`, which contains statistics based on the prefix/postfix of the object names.

8.2.1 SLOC

On average, the number of source lines per method is 13 which is well under the maximum recommended limit of 40. The reason for this low average number may be due to the fact that Dynamics AX has a lot of methods (28%) with only 4 lines of code, and thus helps drive the average down. These methods are most likely getters/setters which are used to expose class level variables to the public.

A total of 6.709 methods are over the limit, and more than 1.100 methods have a SLOC count of more than 100. Nearly all of these methods at the same time have a V(G) of more than 10, so many of these methods are definitely candidates for a rewrite or at least a thorough inspection.

8.2.2 CP

Nearly 80% of the methods contain less than 10% comments and thus will create a Best Practice warning.

A total of 93.469 methods (73%) do not have any comments at all. Approx. 1/3 of the methods without any comments are probably used as simple getters/setters of class instance variables or are returning a constant, as they have only have 5 or less source lines. However more than 3.000 methods are especially critical, as they both violates SLOC (>40) and still have no comments.

8.2.3 V(G)

As a whole, the average Cyclomatic complexity is acceptable. Even if we do not take the many getters/setters into account, it is still only 3,19. Only little more than 3% of the methods are violating the constraint.

In Table 8-2 is shown the top 15 with regards to the Cyclomatic complexity, which all have more than 10 times the recommended limit. These objects also have a really high SLOC, especially method setAllocationDimension in class COSCalculationRun, which is the owner of the overall highest SLOC value (28 times the SLOC limit!).

V(G)	SLOC	Path
358	644	\Classes\LedgerSIEExportFile\getSRU
247	476	\Classes\SysContextMenu\verifyItem
173	625	\Forms\SysNetCSSEditor\Methods\updateProperties
170	1152	\Classes\COSCalculationRun\setAllocationDimension
155	428	\Classes\WebFormHtml\initVersion
150	755	\Data Dictionary\Tables\InventItemBarcode\Methods\findItemDimensions
139	709	\Classes\CustVendSettle\settleNow
128	823	\Forms\SysNetCSSEditor\Methods\loadProperties
122	550	\Classes\smmSalesManagementQueries\defaultQuery
117	480	\Classes\XBRLProcessor\importLinkbase
114	225	\Forms\SysNetHTMLEditor\Methods\runTool
111	657	\Data Dictionary\Tables\InventSum\Methods\findSum
110	109	\Classes\SysSpellChecker\wordLanguageId
108	644	\Classes\ReleaseUpdateDB39_PBA\updatePBValidationRules
102	476	\Classes\CCAdoSqlScanner\tokenStr

Table 8-2 Methods with V(G) > 100

8.2.4 WMC

The average sum of complexities per class is 35,98 which is an adequate number. Around 18% of the classes exceed the limit for WMC of 50, and 349 classes have as WMC greater than 100 and should therefore be further evaluated to see if it is possible to separate some of their functionality out into other classes.

8.2.5 DIT

The Depth of Inheritance Tree is the only metric where no classes violate the constraint of a maximum depth of eight. In Figure 8-1 a histogram for the DIT values can be seen. No classes have a DIT of zero since they all explicitly inherit from Object. 74% (3.808) of the classes inherit from something other than Object, and only one of these has the maximum depth allowed. These numbers all indicate that in Dynamics AX the use of inheritance is well thought out as many classes inherit, but the trees do not get excessively deep.

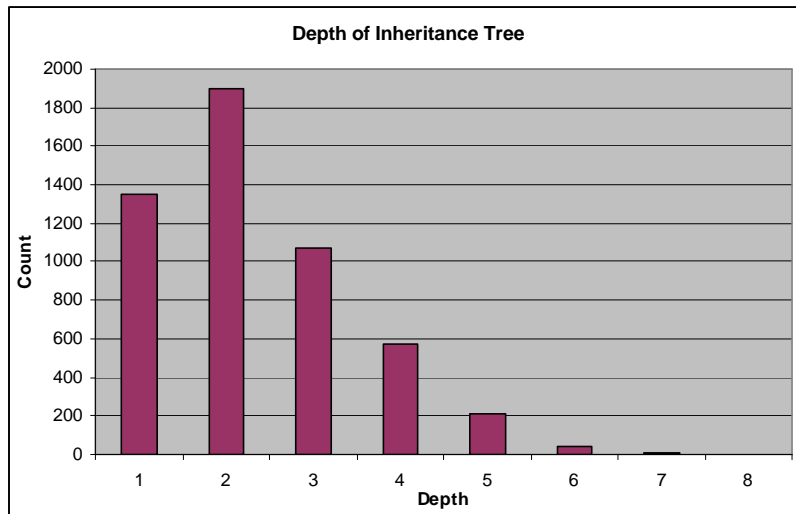


Figure 8-1 Histogram for the DIT metric

8.2.6 NOC

17% (881) of the pure code classes are being extended by another class. 531 are only extended by one or two classes, while others are being heavily used. In Table 8-3 the top ten most extended classes is shown. Many of these are part of the AX framework so it is quite natural that they are heavily used.

NOC	Path
344	\Classes\RunBase
209	\Classes\RunBaseBatch
140	\Classes\SysCOMBase
64	\Classes\AxInternalBase
58	\Classes\SysConsistencyCheck
44	\Classes\ImageListAppl
40	\Classes\VendOutPaymRecord
39	\Classes\RunBaseReport
37	\Classes\VendOutPaym
35	\Classes\SysWizard

Table 8-3 Top 10 NOC

8.2.7 CBO

On average, the Coupling Between Objects is too high, and 36% of all the classes has exceeded the maximum allowed value for CBO. This indicates that a lot of references to other classes are needed to perform a function, and that the various modules/components rely very much on each other.

A suggestion to solve this issue might be to make greater use of Façade patterns, so the modules can have a much sharper distinction between them. This way, developers do not need to know exactly how the referenced modules are internally structured they just need to know which methods to call.

8.2.8 RFC

In Figure 8-2 a histogram for RFC is shown. As can be seen, around 50% of the classes can potentially invoke less than 25 distinct methods. 21% violates the maximum RFC of 50, and 10% of the classes has a RFC of 80 or more.

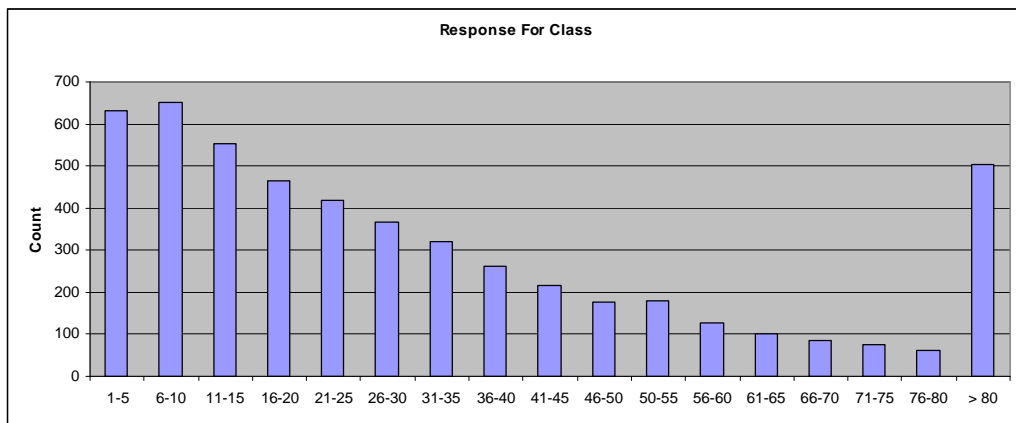


Figure 8-2 Histogram of RFC

8.2.9 LCOM

The average value for LCOM is more than 4 times higher than the recommended value. In theory this should mean, that there should be four times as many classes, as all classes with a LCOM > 1 should be split into smaller classes. This might not be the case, as coherence can be something else than the data cohesion that the LCOM metric measures. Each of the classes with LCOM higher than one should however be manually inspected by someone with both domain- and programming knowledge, to asses if the coherence is sufficient.

8.2.10 FI

On average, each class is being referenced by 6 other classes. This relatively high number is due to some classes having an extremely high FI. The most used is the `Global` class which really brings up the average. In Table 8-4 is shown the ten highest classes with regards to FI. Please note that the number of Fan-Ins per class is counted not only from the pure code classes, but also from other objects like Forms and tables.

FI	Path
5137	\Classes\Global
1339	\Classes\RunBase
1036	\Classes\Dialog
926	\Classes\DialogRunbase
916	\Classes\DialogField
588	\Classes\SysQuery
451	\Classes\RunBaseBatch
437	\Classes\ClassFactory
411	\Classes\Box
368	\Classes\RunbaseProgress

Table 8-4 Top 10 FI

8.3 Comparison of selected modules

In Table 8-5 the results (average values) are compared by module (prefix). Please note that this is not a complete list of the modules, since objects in Dynamics AX are split into 422 modules based on the object name prefix. All the selected modules have lots of classes, so a fair average value can be obtained.

As can be seen DSO and Web are at the opposite ends of the scale. This is because classes prefixed DSO acts as simple wrappers for COM objects which is most evident in SLOC = 4 and V(G) = 1, which basically means that each method only has one "real" line of code. Web, on the other hand, contains quite big methods that are somewhat complex.

The classes prefixed with Ax also stand out, as they have the highest WMC. The V(G) is not particularly high, hence each of the classes contains lots of methods. One other thing to notice is that despite the high WMC and CBO, there are almost no comments (0,1%).

Prefix/Metric	SLOC	CP	V(G)	WMC	DIT	NOC	CBO	FI	RFC	LCOM
DSO	4,00	17,99	1,00	37,73	2,00	0,00	8,33	2,71	68,78	37,68
COS	19,03	4,07	2,81	36,39	2,14	0,27	20,88	1,74	21,89	2,86
Vend	10,82	6,30	1,94	18,36	4,07	0,83	13,27	0,89	26,46	3,72
Invent	11,72	3,21	2,47	37,22	2,57	0,66	22,33	6,99	36,96	4,30
Ledger	15,35	2,81	2,94	39,39	2,38	0,59	23,81	3,51	35,04	3,37
Proj	11,57	2,58	2,56	36,55	2,53	0,71	23,00	3,21	34,42	6,60
Sys	14,40	6,75	2,68	27,22	1,81	0,77	15,70	6,97	29,97	3,54
Ax	10,13	0,10	2,33	177,76	2,08	0,13	40,76	2,33	109,11	2,68
Web	20,63	4,02	4,12	63,09	2,17	1,02	22,66	9,45	51,19	4,41

Table 8-5 Comparison of results by modules (prefix)

8.4 Comparison by team

Table 8-6 shows the average values for each of the metrics by teams. One of the teams that stand out negatively is SCM Collaboration. They have somewhat big classes (WMC) and very low comment percentage. Also their RFC value is not good. This might be somewhat concerning since Fan-In is high, which means that a lot of other classes are depending on the functionality they provide.

Team/Metric	SLOC	CP	V(G)	WMC	DIT	NOC	CBO	FI	RFC	LCOM
AID	20,94	6,65	3,33	49,88	2,29	0,93	29,52	2,95	47,23	6,61
AIF	15,31	9,59	2,73	23,71	1,13	0,00	18,92	5,58	27,87	2,26
All	10,96	7,11	2,02	10,04	2,42	0,20	8,16	1,89	14,67	2,09
Business Intel.	8,35	13,76	1,69	45,58	1,77	0,11	15,00	3,36	58,98	21,89
Circle Capital	12,83	2,85	2,51	25,10	2,07	0,26	20,74	1,67	30,81	2,59
Circon	19,03	4,07	2,81	36,39	2,14	0,27	20,88	1,74	21,89	2,86
Client and EP	17,44	3,76	3,38	43,23	2,17	0,76	21,33	5,72	41,42	3,39
FIM	14,23	3,99	2,59	31,32	2,87	0,72	22,40	3,15	33,19	3,78
Fixed Assets	15,66	6,82	2,97	33,10	2,04	0,55	20,75	2,70	30,22	3,17
GDL	12,77	6,46	2,43	34,77	2,41	2,23	23,99	9,46	38,21	4,50
MSO	13,68	4,78	2,70	46,98	2,31	0,36	28,71	2,38	46,65	4,11
Project	11,43	2,51	2,52	32,65	2,65	0,71	20,58	2,92	31,25	6,44
SCM	11,87	2,66	2,47	35,16	2,49	0,66	22,04	4,79	36,91	4,29
SCM Collab.	11,17	0,81	2,43	90,39	1,95	0,46	29,72	16,92	68,88	3,31
SCM Tech	14,80	6,31	2,82	28,97	1,84	0,67	15,56	18,97	29,56	3,01
Server & Tools	14,91	4,92	2,73	34,54	1,71	0,31	17,59	10,53	34,11	2,48
Tectura	12,40	1,69	2,42	26,46	2,11	0,68	20,40	2,52	31,38	3,03
Thy	15,16	3,66	2,95	30,13	2,38	0,13	23,76	1,90	30,50	3,04

Table 8-6 Comparison of results by teams

Chapter 9 Metric evaluation

In the beginning of this project ten metrics were selected and implemented in X++ as described previously in this report. After having used the new tool and analyzed the generated data, it has become clear that not all the ten selected metrics are equally useful, or at least that they have different target groups.

The three traditional metrics SLOC, CP and V(G) have proven quite easy to understand and once the relatively simple rules has been studied and are in the back of the developers minds, they will automatically begin to write code of a lower complexity. And should they forget the rules, the issued BP deviations will help them remember.

The WMC metric is very useful for evaluating the total complexity of a class. When a class with a high WMC is identified, a good strategy for solving the problem, is to use the new form "Metric results" to find out if a single method in the class contributes with a very high V(G) number and should be rewritten, or if the high WMC is due to a high number of small methods which should be split into separate classes.

The CBO and RFC are good for pinpointing classes with excessive coupling. If a developer gets a BP deviation for these metrics he should consider using a Facade pattern to simplify the class' communication.

The DIT metric will most likely be used by the developer to find out how many ancestors a new class will get. Most of the classes in Dynamics AX do not have a very deep hierarchy, so in most cases the depth will not become a problem.

Fan In and NOC are probably most useful for the Dev Leads, as the developers will seldom see the BP deviations issued. The reason for this is that when creating a new class, the FI and NOC metric on that class will not trigger, since it's the FI and NOC metric of the referenced or parent class that will be affected by the change. Both of these metrics do not directly help in reducing the class complexity, but can nevertheless provide information of how high effect a change in the class will have. This may be used in connection with a source control system where the most vital classes could be given a higher security level.

As section 7.3 revealed, the metric implementations do not themselves adhere to the LCOM metric, as they have LCOM = 3 or 4. The people who reviewed the classes agreed that the classes have a correct OO design and are semantically coherent. This raises the question if LCOM is able to measure the coherence in real OO systems. The current implementation of the LCOM metric has been found to be unsuitable in a number of cases:

- A method, which only operates on data defined in a parent class will be treated as a separate component and adds to the LCOM value.
- If a method does not use any instance variable at all, it will add to the LCOM. This happens frequently when a method from a parent class (e.g. `getDescription`) is overridden.

- A class that is only used for data-storage and which does not operate on the data but merely uses get/set methods for the instance variables will get a very high LCOM value, although it is perfectly alright to have such a class according to the OO principles.
- If all instance values of a class are initialized in the constructor (`new`) method, then the class will nearly always get LCOM=1, because all variables are then connected. This should indicate that it has perfect coherence, but this is not always the case.

One of the dangers of having a metric that is somewhat misleading is that developers might be tempted to write bad code that satisfies the metric rules, instead of writing correct OO code that then causes BP complexity deviations. It is therefore recommended that the current LCOM metric is excluded from a retail version of the BP complexity tool.

Chapter 10 Future improvements

After having finished the implementation some open issues still remains. These could or should be solved if the code is to be included in the retail version of Dynamics AX. The following section describes these issues and suggests possible solutions.

10.1 Open issues

- All hardcoded texts should be replaced by the use of labels, so they can be localized according to the functional specification.
- As mentioned in the previous chapter, the LCOM metric may be removed to avoid unnecessary or even incorrect re-factoring.
- All users are generating metric statistics into the same table. Although its name starts with “Tmp” it is not a real temporary table, since it exists as a physical table in the database. The reason it is not a real temporary table, is that the metric generation takes quite some time, and it would be annoying to loose all the data when the form is closed. This has the downside that if two users are generating metrics at the same time, they might overwrite each others changes. The problem could be solved by saving all metric data per user, although this would increase the storage space needed.
- When generating metric statistics the program starts with deleting all data in the `TmpCodeMetrics` table. This makes it impossible to generate statistics for a selected number of classes, since only one starting node can be selected from the AOT. This could be solved by letting the user decide if the table should be wiped before starting a new metric generation.
- Calculation of WMC and V(G) should be optimized, since the Cyclomatic complexity is actually calculated twice per method: one time as part of the WMC and one time as the “stand alone” V(G). It could be done by running the V(G) BP check as we process the individual methods in the WMC calculation. Then V(G) should somehow be excluded from the method-level checks, but only if the check was started at class level or higher and only if the method belongs to a class (and not to a Form, Table etc.).
- All code for the new tool has been developed in the “usr” layer. The code should be transferred to the “sys” layer where the rest of the build-in system functionality resides.
- Microsoft might encounter a legal issue, if the tool is included in a release version of Dynamics AX, due to the great number of Best Practice warnings that will be generated. This could make the customers/partners sue MS for bad code quality, because the specific ranges for the metrics are not met. This issue could be resolved by changing the BP warnings to info messages instead, and/or by not specifically

stating the maximum recommended value (instead write “Low is good” or “High is good”).

10.2 New ideas

- An internal presentation of the new tool was held for the entire Dynamics AX Development management. During the great discussion of the prospects for the tool there was an idea to couple the findings of the new tool to Product Studio (internal Microsoft bug tracking tool) to see/verify if there is a connection between the number of bugs and the complexity of a module. There might be some technical challenges to get this feature to work, but it would be of great value to management.

Chapter 11 Conclusion

Process

Although I have tried to use Test Driven Development as much as possible, I have not completely adhered to this (for me) new development method. I have however found it very useful to have unit tests for most of the methods, since there has been a lot of refactoring during the process. These tests really help ones peace of mind with ensuring that a change does not break existing functionality.

In the beginning of this project, I set up a time schedule, where important milestones were highlighted. Every day I wrote in a Project Diary about what was accomplished. This has really helped track the progress and been an important tool in assuring that I was not behind schedule. In fact, for the first time I have been ahead of schedule, thereby having more time to test and document the product.

Product

The new tool can assess the complexity of classes and individual methods. Three traditional (method level) metrics and seven object-oriented (class level) metrics have been implemented. Although more than 200 different metrics have been identified in literature, the chosen ten metrics are all more or less accepted as being valid measures of complexity. For some of the metrics there was an issue with how to handle specific X++ syntax, since the language contains some constructs (e.g. embedded SQL) that the original authors did not take into account. This was solved by finding out what the original intent of the metric was, and then deriving a reasonable solution from this.

On each level (class and method), a new check has been added to the existing Best Practice tool. This check computes the metric values and, if the values are not within an acceptable range, a BP warning or info message is asserted. This information can then directly help the developer to find out what areas can be improved to decrease complexity and thereby increase the quality of the code.

In connection with an intermediate presentation of this project, I found a need to extract statistics for the metrics, which would also include methods/classes that did not create BP warnings. The new form "Metric results" can generate and show raw metric data that are computed from a specific starting point (node) in the AOT. The form also contains functionality to group the metric values based on the object's prefix. This allows the creation of team/module based statistical values.

Findings

To get an overview of how the measures perform in general for the X++ code, a generation of values was started for the entire AOT. Several interesting observations were made. The main points are

- More than 100.000 methods did not have the required amount of comments.
- Although the average Cyclomatic complexity was low, some methods had extremely high numbers (more than 10 times the accepted limit)
- Depth of Inheritance Tree was the only metric where no classes violated the constraint.
- A comparison of the modules and teams revealed big differences.

After having worked with the new tool and analyzed the data of the metrics, I have found that not all the chosen metrics are equally useful in practice. The values of the LCOM metric may be so misleading that I recommend that LCOM is not included in a retail version of the complexity tool.

Chapter 12 Bibliography

[Chidamber91]

Title: Towards A metrics suite for Object Oriented Design
Author: Shyam R.Chidamber, Chris F. Kemerer
Published: ACM Sigplan Notices 26(11), P.197-211, 1991

[Chidamber94]

Title: A metrics suite for Object Oriented Design
Author: Shyam R.Chidamber, Chris F. Kemerer
Published: IEEE Transactions on Software Engineering, P.476-493, June 1994

[Cormen01]

Title: Introduction to Algorithms, Second Edition
Author: Thomas H. Cormen, Charles E. Leis
Published: The MIT Press, 2001

[Encarta]

Page name: MSN Encarta Dictionary
URL: <http://encarta.msn.com/encnet/features/dictionary/dictionaryhome.aspx>

[Etzkorn97]

Title: A Statistical Comparison of Various Definitions of the LCOM Metric
Author: Letha Etzkorn, Carl Davis, Wei Li
Published: University of Alabama in Huntsville, Computer Science Dept., 1997

[Hitz95]

Title: Measuring Coupling and Cohesion In Object-Oriented Systems.
Author: Martin Hitz, Behzad Montazeri
Published: Proc. Int. Symposium on Applied Corporate Computing, Oct. 25-27 1995

[Hudli94]

Title: Software Metrics for Object-Oriented Designs
Author: Hudli, R.V.; Hoskins, C.L.; Hudli, A.V.
Published: IEEE Comput. Soc. Press, P.492-495, 1994

[ISO03]

Title: Software engineering – product quality: Part 3: Internal Metrics
Author: British Standards Institution
Published: ISO/IEC 9126-3:2003

[McCabe96]

Title: Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric
Author: Arthur H. Watson, Thomas J. McCabe

Published: Computer Systems Laboratory, National Institute of Standards and Technology, 1996

[MSDN06]

Page name: X++ grammar

URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Axapta/Appendix_about_EBNF/LANG_X++grammar.asp

[McConnell04]

Title: Code Complete, Second edition

Author: Steve McConnell

Published: Microsoft Press, 2004

[Newkirk04]

Title: Test-Driven Development in Microsoft.Net

Author: James W. Newkirk; Alexei A. Vorontsov

Published: Microsoft Press, 2004

[Rosenberg97]

Title: Software Quality Metrics for Object-Oriented Environments

Author: Dr. Linda H. Rosenberg, Lawrence E. Hyatt

Published: NASA Software Assurance Technology Center, 1997

[Sellers96]

Title: Object-Oriented Metrics – Measures Of Complexity

Author: Brian Henderson-Sellers

Published: Prentice Hall, 1996

Appendix A: Project diary

The following project diary has been continuously updated during the project period, so I have been able to track the progress.

Week 18

- Mon 1/5 Start of project. Installation of Ax on my laptop. Report template created. Project schedule determined. Theory section started.
- Tue 2/5 Work on theory section. Meeting with MFP where we discussed how the solution could be integrated with the BP tool. Also decided that I will use Test-Driven Development when that time comes. Got the MS templates for writing the functional specification.
- Wed 3/5 Researching. Added to theory section. Meeting with Smed discussing report contents and general questions.
- Thu 4/5 Theory work
- Fri 5/5 Theory work

Week 19

- Mon 8/5 Theory work + beginning functional specification.
- Tue 9/5 More work on the functional specification. Installed the unit test framework and read about how to use it.
- Wed 10/5 Discussed the functional specification with MFP and got some great inputs.
- Thu 11/5 Functional specification and beginning overall solution design
- Fri 12/5 Holiday: Store bededag

Week 20

- Mon 15/5 Functional specification.
- Tue 16/5 Created dummy classes for test. Design of class structure.
- Wed 17/5 Deciding what to do about embedded methods and how to handle Forms. Status meeting with Smed.
- Thu 18/5 Got a Virus so formatted and reinstalled the laptop. Merged the Functional Specification into main report.
- Fri 19/5 Exam of ITU project

Week 21

- Mon 22/5 Job interview at MS. Added "Select statements" to the functional requirements.
- Tue 23/5 Started on implementing SLOC
- Wed 24/5 Finished SLOC
- Thu 25/5 Holiday: Kr. Himmelfartsdag
- Fri 26/5 Started on implementing V(G)

Week 22

- Mon 29/5 Implementing V(G). Implemented the framework for methods.
- Tue 30/5 Finished implementing V(G). Found and corrected errors in implementation of

SLOC. Started on implementing CP.
 Wed 31/5 Finished implemented CP. **All traditional metrics complete!!!!** Started on implementing metrics calculation outside of the BP framework, for easier access to creating statistics.
 Thu 1/6 Ran calculation of V(G), SLOC & CP metrics on 75.000 methods (!) and analyzed the data.
 Fri 2/6 Created powerpoint presentation for Mid-way presentation

Week 23

Mon 5/6 Holiday: 2. Pinsedag
 Tue 6/6 Mid-way presentation for Smed, Fruergaard and Ola. Discussed some issues regarding SQL, classDeclaration, break and continue. Changed the program so classDeclaration methods are not used in calculating method metrics.
 Wed 7/6 Changed calculation of V(G) to reflect yesterdays discussion.
 Thu 8/6 Implemented DIT. Started on WMC
 Fri 9/6 Finished WMC. Started on CBO.

Week 24

Mon 12/6 Finished CBO. Implemented RFC.
 Tue 13/6 Started on LCOM.
 Wed 14/6 Implemented a support class for doing DFS on graphs.
 Thu 15/6 Changed class level metrics so they use temporary xRef. Continued LCOM implementation.
 Fri 16/6 Finished LCOM. Implemented NOC.

Week 25

Mon 19/6 Started on implementing FI
 Tue 20/6 **All implementation complete!!!!**
 Wed 21/6 Implemented creation of statistics on team/prefix level
 Thu 22/6 Data analysis
 Fri 23/6 Prepared powerpoint presentation of results for Sunday
 Sat 24/6 -
 Sun 25/6 Presentation of results for all Dynamics Ax developer-leads

Week 26

Mon 26/6 Documentation: Analysis of results
 Tue 27/6 Documentation: Analysis of results, Future improvements
 Wed 28/6 Cleaning up source code. Documentation: Adherence to own rules
 Thu 29/6 Documentation: Design
 Fri 30/6 Documentation: Implementing base classes and BP integration

Week 27

Mon 3/7 Documentation: Integration with BP, SLOC, CP
 Tue 4/7 Documentation: V(G), WMC, DIT, NOC
 Wed 5/7 Documentation: CBO, RFC, LCOM, FI
 Thu 6/7 Documentation: Statistics implementation
 Fri 7/7 Documentation: Unit tests, functional test, Conclusion
 Sat 8/7 Documentation: Summary. **Report version 1 is ready for review!**

Week 28

- Mon 10/7 Proof-reading report.
- Tue 11/7 Review of report with Michael Fruergaard, Ola Mortensen & Morten Gersborg-Hansen. Report updated with spelling and phrasing.
- Wed 12/7 Method names of SourceCodeChunker changed. retrieveAndInsert removed from sysBPCheckBase and similar functionality implemented in XXX. Report, figures and appendixes updated to reflect this change.
- Thu 13/7 New Metric evaluation chapter added and reviewed
- Fri 14/7 Printing and final check.

Week 29

- Mon 17/7 **Report hand-in!**

Appendix B: Source code

Modified	3
Class: SysBPCheckMemberFunction	3
Class: SysBPCheckClassNode	5
Macro: SysBPCheck	7
Form: SysBPSetup	7
Metric Framework.....	12
Class: CodeMetricBase	12
Class: CodeClassMetric.....	12
Class: CodeMethodMetric	13
Enumeration: BPSeverity.....	14
Metric Implementations	15
Class: CodeMetricCPMethod	15
Class: CodeMetricVGMMethod	16
Class: CodeMetricSLOCMethod.....	20
Class: CodeMetricFI	22
Class: CodeMetricNOC.....	23
Class: CodeMetricLCOM	24
Class: CodeMetricRFC	26
Class: CodeMetricCBO.....	28
Class: CodeMetricWMC.....	29
Class: CodeMetricDIT	30
Other	32
Class: GraphUndirected.....	32
Class: GraphEdge.....	35
Class: GraphNode	36
Class: ClassInstanciator	37
Class: StringUtil	38
Class: SourceCodeChunker	38
Statistics	43
Form: CodeMetricsResults	43
Class: CodeMetricGenerator	46
Class: CodeMetricStatItem	48
Class: CodeMetricTeamStatGenerator.....	49
Unit tests	53
Class: CodeMetricTeamStatGeneratorTest.....	53
Class: CodeMetricGeneratorTest	53
Class: CodeMetricStatItemTest	54
Class: CodeMetricFITest	55
Class: CodeMetricNOCTest	56
Class: GraphEdgeTest.....	57
Class: GraphUndirectedTest	57
Class: GraphNodeTest	60

Class: CodeMetricLCOMTest	60
Class: CodeMetricRFCSTest	61
Class: CodeMetricCBOTest.....	62
Class: CodeMetricWMCTest	64
Class: CodeMetricDITTest.....	65
Class: ClassInstanciatorTest	66
Class: StringUtilTest	66
Class: SourceCodeChunkerTest	67
Class: CodeMetricCPMethodTest	69
Class: CodeMetricVGMethodTest	70
Class: CodeMetricSLOCMethodTest.....	71
Class: SysBPCheckMemberFunctionTest.....	73
Class: SysBPCheckClassNodeTest	74
Test classes	76
Class: CodeMetricDummy1	76
Class: CodeMetricDummy2	79
Class: CodeMetricDummy3	79
Class: CodeMetricDummy4	79
Unit test helper classes	80
Class: SysBPCheckComplexityEnabler.....	80

Modified

This section contains any existing items that have been modified to make the new tool. **Please note that only the methods that have been modified/added will appear in this document! Existing code is marked with grey color and new code is black.**

Class: SysBPCheckMemberFunction

```
class SysBPCheckMemberFunction extends SysBPCheckBase
{
    SysMethodInfo          sysMethodInfo;
    SysScannerClass       scanner;
    xRefTmpReferences     tmpxRefReferences;    // the source, as the xRef sees it
    MemberFunction        memberFunction;
    boolean               xRefIsInited;
    UtilElementType       parentType;
    identifiername        parentName;
    boolean               allowHardcodedTexts;

    //
    //Do not dispose these maps as their content is static
    //
    Map                   CASServerMapInstance;
    Map                   CASServerMapStatic;
    Map                   CASAllMapInstance;
    Map                   CASAllMapStatic;

    //List with instances of classes that inherits from CodeMethodMetric
    List                  codeMethodMetricList;

    #define.del('DEL_')
}

public void check()
{
    super();

    if (sysMethodInfo.compiledOk() && memberFunction.AOTgetSource())
    {
        this.checkSource();
        this.checkUseLocalObjects();
        this.checkIndentation();
        this.checkConstants();

        if (parameters.CheckTwC)
        {
            this.checkUseOfDangerousClasses();
            this.checkUseOfDangerousFunctions();
            this.checkUseOfCASProtectedAPIs();
        }

        if (parameters.CheckEmptyMethods)
        {
            this.checkEmptyMethod();
        }

        if (parameters.CheckDate)
        {

```

```

        this.checkDate();
    }

    if (parameters.CheckAOS)
    {
        this.checkUseOfFieldLists();
    }

    if (parameters.CheckPrivacy)
    {
        this.checkAccessSpecifier();
    }

    if (parameters.CheckDiscontinuation)
    {
        this.checkDiscontinuation();
    }

    if (parameters.CheckFutureReservedWords)
    {
        this.checkFutureReservedWord();
    }

    if (parameters.CheckVariables)
    {
        this.checkVariables();
    }

    if (parameters.CheckSourcePrintAndPause)
    {
        this.checkUseOfPrintAndPause();
    }

    if (parameters.CheckComplexity)
    {
        this.checkComplexity();
    }
}

}

void checkComplexity()
{
    CodeMethodMetric codeMethodMetric;
    ListEnumerator enum;
    str errMessage;
    ;

    if (sysBPCheck.treeNode().treeNodeName() != 'classDeclaration')
    {

        //Loop through all the metric classes that are available
        enum = codeMethodMetricList.getEnumerator();
        while(enum.moveNext())
        {
            //Cast as CodeMethodMetric
            codeMethodMetric = enum.current();

            //Pass the tree node of the method to check
            codeMethodMetric.setElement(sysBPCheck.treeNode());
        }
    }
}

```



```

//Pass the scanner already created
codeMethodMetric.setScanner(scanner);

//Perform the check
errMessage = codeMethodMetric.getBPStr();

//If the errMessage is not empty then add a new BP message
if (errMessage != '')
{
    //Find out what to do with the message
    switch(codeMethodMetric.getBPSeverity())
    {
        case BPSeverity::Info:
sysBPCheck.addInfo(codeMethodMetric.getErrorCode(),0,0,errMessage);
                break;
        case BPSeverity::Warning:
this.addSuppressableWarning(codeMethodMetric.getErrorCode(),0,0,errMessage);
                break;
        case BPSeverity::Error:
this.addSuppressableError(codeMethodMetric.getErrorCode(),0,0,errMessage);
                break;
    }
}
}
}

protected void new()
{
    ;
    super();

    //Create a list that will hold instances of the metric classes
    codeMethodMetricList =
ClassInstanciator::createSubClassInstances(classNum(CodeMethodMetric));
}

```

Class: SysBPCheckClassNode

```

class SysBPCheckClassNode extends SysBPCheckBase
{
    SysDictClass sysDictClass;

    //List with instances of classes that inherits from CodeMethodMetric
    List codeClassMetricList;
}

public void check()
{
    super();

    this.checkRunMode();
    this.checkNamingConventions();
}

```

```

if (parameters.CheckRunBaseImplementation)
{
    this.checkRunBaseImplementation();
    this.checkPackable();
}

if (parameters.CheckMissingMember)
{
    this.checkMissingMember();
}

if (parameters.CheckClassAbstract)
{
    this.checkAbstract();
}

if (parameters.CheckConstructors)
{
    this.checkConstructors();
}
if (parameters.CheckTableAxBCParmFields)
{
    this.checkAxBCParmFields();
}

if (parameters.CheckComplexity)
{
    this.checkComplexity();
}
}

void checkComplexity()
{
    CodeClassMetric codeMetric;
    ListEnumerator enum;
    str errMessage;
    xRefUpdateTmpReferences tmpUpdate;
    xRefTmpReferences tmpxRefReferences;
    ;

    //Create tmp references for the entire class (for optimization)
    tmpUpdate = new xRefUpdateTmpReferences();
    tmpUpdate.fillTmpxRefReferences(sysBPCheck.treeNode());
    tmpxRefReferences = tmpUpdate.allTmpxRefReferences();

    //Loop through all the metric classes that are available
    enum = codeClassMetricList.getEnumerator();
    while(enum.moveNext())
    {
        //Cast as CodeClassMetric
        codeMetric = enum.current();

        //Pass the tree node of the method to check
        codeMetric.setElement(sysBPCheck.treeNode());

        //Pass the tmp references already generated
        codeMetric.setXRefTmpReferences(tmpxRefReferences);

        //Perform the check
        errMessage = codeMetric.getBPStr();
    }
}

```

```

//If the errMsg is not empty then add a new BP message
if (errMsg != '')
{
    //Find out what to do with the message
    switch(codeMetric.getBPSeverity())
    {
        case BPSeverity::Info:
            sysBPCheck.addInfo(codeMetric.getErrorCode(),0,0,errMessage);
            break;
        case BPSeverity::Warning:
            sysBPCheck.addWarning(codeMetric.getErrorCode(),0,0,errMessage);
            break;
        case BPSeverity::Error:
            sysBPCheck.addError(codeMetric.getErrorCode(),0,0,errMessage);
            break;
    }
}
}
}

protected void new()
{
    super();

    //Create a list that will hold instances of the metric classes
    codeClassMetricList =
    ClassInstanciator::createSubClassInstances(classNum(CodeClassMetric));
}

```

Macro: SysBPCheck

Note: Only the added lines are shown here

```

// Complexity metrics
#define.BPErrorCodeMetric(880)
#define.BPErrorCodeMetricSLOCMMethod(881)
#define.BPErrorCodeMetricVGMMethod(882)
#define.BPErrorCodeMetricCPMethod(883)
#define.BPErrorCodeMetricDIT(884)
#define.BPErrorCodeMetricWMC(885)
#define.BPErrorCodeMetricNOC(886)
#define.BPErrorCodeMetricCBO(887)
#define.BPErrorCodeMetricRFC(888)
#define.BPErrorCodeMetricLCOM(890)
#define.BPErrorCodeMetricFI(891)

```

Form: SysBPSetup

```

private void buildSelectionTree()
{
    int      nodeRoot;
    int      nodeGeneral;
    int      nodeSpecific;
    int      tmpNode;
    ;
}

```

```

nodeRoot = element.addNode(selectionTree.getRoot(), 0, #disabled, #gotChilds,
"@SYS70918");

// General Checks
nodeGeneral = element.addNode(selectionTree.getRoot(), 0, #disabled, #gotChilds,
"@SYS72390");

element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckProperties),
parameter.CheckProperties);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckAOTPathUnique),
parameter.CheckAOTPathUnique);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckObjectId),
parameter.CheckObjectId);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckAOS),
parameter.CheckAOS);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckTwC),
parameter.CheckTwC);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckUsed),
parameter.CheckUsed);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckReferences),
parameter.CheckReferences);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckDiscontinuation),
parameter.CheckDiscontinuation);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters,
CheckTableAndReclIdReferences), parameter.CheckTableAndReclIdReferences);

// Keys
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckConfigurationKeys),
parameter.CheckConfigurationKeys);
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckSecurityKeys),
parameter.CheckSecurityKeys);

// Labels
tmpNode = element.addNode(nodeGeneral, 0, #disabled, #gotChilds, "@SYS13322");
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckLabelUse),
parameter.CheckLabelUse);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckHelpUse),
parameter.CheckHelpUse);

// Analysis Visibility
element.addNode(nodeGeneral, fieldnum(SysBPPParameters, CheckAnalysisVisibility),
parameter.CheckAnalysisVisibility);

// Specific Checks
nodeSpecific = element.addNode(selectionTree.getRoot(), 0, #disabled, #gotChilds,
"@SYS72391");

```

```

// Tables
tmpNode = element.addNode(nodeSpecific, 0, #disabled, #GotChilds, "@SYS9678");
element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableFieldPnameUniqueness), parameter.CheckTableFieldPnameUniqueness);

    element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckTableIndexUse),
parameter.CheckTableIndexUse);
    element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckTableDeleteActions),
parameter.CheckTableDeleteActions);
    element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckTableTitleFields),
parameter.CheckTableTitleFields);
    element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckTableFormRef),
parameter.CheckTableFormRef);
    element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableAxBCParmFields), parameter.CheckTableAxBCParmFields);

// Table Fields
element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableFieldsFieldGroupMember), parameter.CheckTableFieldsFieldGroupMember);
    element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableFieldHasSameNameAsMethod),
parameter.CheckTableFieldHasSameNameAsMethod);

// Table Fields Group
element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableFieldGroupNumberOfFields),
parameter.CheckTableFieldGroupNumberOfFields);

// Analysis Behavior, Totaling, CurrencyCodeFields and CurrencyDateFields
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckTableAnalysisBehavior
), parameter.CheckTableAnalysisBehavior );
    element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableCurrencyCodeFields), parameter.CheckTableCurrencyCodeFields);
    element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableCurrencyDateFields), parameter.CheckTableCurrencyDateFields);

// Table Relations
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckTableRelations),
parameter.CheckTableRelations);

// Table Collections
tmpNode = element.addNode(nodeSpecific, 0, #disabled, #GotChilds, "@SYS25433");
element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckTableCollectionRelation), parameter.CheckTableCollectionRelation);
// Maps

// Views

```

```

// Extended Data Types

// Classes
tmpNode = element.addNode(nodeSpecific, 0, #disabled, #GotChilds, "@SYS60851");
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckClassAbstract),
parameter.CheckClassAbstract);
element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckRunBaseImplementation), parameter.CheckRunBaseImplementation);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckMissingMember),
parameter.CheckMissingMember);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckConstructors),
parameter.CheckConstructors);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckComplexity),
parameter.CheckComplexity);

// Methods (Member Functions)
tmpNode = element.addNode(nodeSpecific, 0, #disabled, #GotChilds, "@SYS25613");
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckEmptyMethods),
parameter.CheckEmptyMethods);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckDate),
parameter.CheckDate);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckPrivacy),
parameter.CheckPrivacy);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckSourcePrintAndPause),
parameter.CheckSourcePrintAndPause);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckVariables),
parameter.CheckVariables);
element.addNode(tmpNode, fieldnum(SysBPPParameters,
CheckFutureReservedWords), parameter.CheckFutureReservedWords);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckTextInSingleQuotes),
parameter.CheckTextInSingleQuotes);

// Forms
tmpNode = element.addNode(nodeSpecific, 0, #disabled, #GotChilds, "@SYS98083");
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckFormSize),
parameter.CheckFormSize);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckDisablingTechnique),
parameter.CheckDisablingTechnique); // CheckFormControlDisablingTechnique
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckFormControlNames),
parameter.CheckFormControlNames);
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckFormTabPage),
parameter.CheckFormTabPage);

// Labels
tmpNode = element.addNode(nodeSpecific, 0, #disabled, #gotChilds, "@SYS83850");
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckSpelling),
parameter.CheckSpelling);

```

```
// Perspectives
tmpNode = element.addNode(nodeSpecific, 0, #disabled, #gotChilds, "@SYS94647");
element.addNode(tmpNode, fieldnum(SysBPPParameters, CheckPerspectives),
parameter.CheckPerspectives);

SysFormTreeControl::setTreeStateImage_CheckBox(selectionTree, nodeRoot);

selectionTree.expand(nodeRoot);
selectionTree.expand(nodeGeneral);
selectionTree.expand(nodeSpecific);
}
```

Metric Framework

Class: CodeMetricBase

```
public abstract class CodeMetricBase
{
    //Import macro SysBPCheck
    #SysBPCheck

    //The node to run the metric calculations on
    TreeNode node;
}

//Return the BP info/warning string if the value violates the limits
public abstract str getBPStr()
{
}

//The calculated metric value should be returned
public abstract int getValue()
{
}

void setElement(TreeNode _node)
{
    //Saves the node for later use
    node = _node;
}

public BPSeverity getBPSeverity()
{
    //Return info as default severity level
    return BPSeverity::Info;
}

public str getDescription()
{
    //Blank as default value.
    return '';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetric;
}
```

Class: CodeClassMetric

```
public abstract class CodeClassMetric extends CodeMetricBase
{
    xRefTmpReferences tmpxRefReferences;
    boolean xRefIsInited;
}

protected void initTmpXRef()
```



```

{
    xRefUpdateTmpReferences tmpUpdate;

    if (!xRefIsInited)
    {
        //Create tmp references for the entire class
        tmpUpdate = new xRefUpdateTmpReferences();
        tmpUpdate.fillTmpxRefReferences(node);
        tmpxRefReferences = tmpUpdate.allTmpxRefReferences();

        //Set the flag to true
        xRefIsInited = true;
    }
}

void setElement(TreeNode _node)
{
    super(_node);

    //Reset the flag for generation of tmpXref
    xRefIsInited = false;
}

public void setXRefTmpReferences(xRefTmpReferences _ref)
{
    //Save the tmp ref in class level variable and set flag to true
    tmpxRefReferences = _ref;
    xRefIsInited = true;
}

```

Class: CodeMethodMetric

```

public abstract class CodeMethodMetric extends CodeMetricBase
{
    //Scanner class used for reading symbols of the method source code
    SysScannerClass scanner;
}

protected SysScannerClass getScanner()
{
    //If no scanner class have been provided, then generate a new scanner class
    based on the TreeNode
    if (scanner == null)
    {
        scanner = new SysScannerClass(node);
    }

    //Returns the scanner with information of method symbols
    return scanner;
}

void setElement(TreeNode _node)
{
    super(_node);

    //Reset the scanner
    scanner = null;
}

```

```

public void setScanner(SysScannerClass _scanner)
{
    //Save to local variable
    scanner = _scanner;
}

```

Enumeration: BPSeverity

```

ENUMTYPE #BPSeverity
PROPERTIES
    Name                #BPSeverity
    UseEnumValue        #Yes
ENDPROPERTIES

TYPEELEMENTS
#None
PROPERTIES
    Name                #None
    Label               #None
    EnumValue           #0
ENDPROPERTIES

#Info
PROPERTIES
    Name                #Info
    Label               #Info
    EnumValue           #1
ENDPROPERTIES

#Warning
PROPERTIES
    Name                #Warning
    Label               #Warning
    EnumValue           #2
ENDPROPERTIES

#Error
PROPERTIES
    Name                #Error
    Label               #Error
    EnumValue           #3
ENDPROPERTIES

ENDTYPEELEMENTS
ENDENUMTYPE

```

Metric Implementations

Class: CodeMetricCPMethod

```
class CodeMetricCPMethod extends CodeMethodMetric
{
    //The minimum allowed Comment Percentage
    #define.MinCPValue(10)
}

public BPSeverity getBPSeverity()
{
    //Warning
    return BPSeverity::Warning;
}

public str getBPStr()
{
    str ret;
    int cpVal;

    //Get the value for CP for the source code
    cpVal = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (cpVal < #MinCPValue)
        ret = sprintf('The Comment Percentage (CP) of method %1 is %2 (Min.
recommended %3)',node.treeNodeName(),int2str(cpVal),int2str(#MinCPValue));

    return ret;
}

public str getDescription()
{
    //Comment Percentage
    return 'CP';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetricCPMethod;
}

int getValue()
{
    //Return the value for CP for the source code
    return CodeMetricCPMethod::calcCP(node.AOTgetSource());
}

public static int calcCP(str sourceCode)
{
    int cp;
    int newlinesInComment;
    int linesWithComments;
    int blankLines;
    int lastCommentLine;
```

```

str tmp;

SourceCodeChunker chunker = new SourceCodeChunker(sourceCode);
;

//Loop through code/comment chunks
while(chunker.moveToNext())
{
    if (chunker.currentCommentChunk() != '')
    {
        newlinesInComment =
StringUtil::CountOccurences(chunker.currentCommentChunk(),'\n');

        if(chunker.commentStartLine() > lastCommentLine)
            linesWithComments += newlinesInComment + 1;
        else
            linesWithComments += newlinesInComment;

        lastCommentLine = chunker.commentStartLine() + newlinesInComment;
    }

    //Remove spaces from the source code chunk
    tmp = strrem(chunker.currentCodeChunk(),' ');

    //Add the number of blank lines in the chunk
    blankLines += StringUtil::CountOccurences(tmp,'\n\n');
}

//Calculate CP
if((chunker.lineCount() - blankLines) > 0)
    cp = (linesWithComments / (chunker.lineCount() - blankLines))*100;

return cp;
}

```

Class: CodeMetricVGMethod

```

class CodeMetricVGMethod extends CodeMethodMetric
{
    //Explanations of the symbol values
    #TokenTypes

    //The maximum allowed value for the Cyclomatic Complexity
    #define.MaxCCValue(10)
}

public BPSeverity getBPSeverity()
{
    //Warning
    return BPSeverity::Warning;
}

public str getBPStr()
{
    str ret;
    int ccVal;

    //Get the value for V(G) for the source code
    ccVal = this.getValue();
}

```

```

//If the value exceeds the threshold limit, return an error string
if (ccVal > #MaxCCValue)
    ret = sprintf('The Cyclomatic Complexity of method %1 is %2 (Max.
recommended %3)',node.treeNodeName(),int2str(ccVal),int2str(#MaxCCValue));

    return ret;
}

public str getDescription()
{
    //Return the description
    return 'V(G)';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrCodeMetricVGMethod;
}

int getValue()
{
    //Return the value for V(G) for the source code
    return CodeMetricVGMethod::calcVG(this.getScanner());
}

/*-----
    Need to look for:

        statementbeginend typename + identifier + "("    -> Definition of embedded
method

        "&&"                                             -> Conditional operator, must not
be within SQL

        "||"                                             -> Conditional operator, must not
be within SQL

        "?"                                             -> Inline If branch

        statementbeginend + "if"                       -> If Branch
        statementbeginend + "while"                    -> Either normal or SQL while Loop
        statementbeginend + "for"                      -> For Loop
        statementbeginend + "case"                    -> Non-fallthrough label in switch
        statementbeginend + "default"                 -> Non-fallthrough default in
switch

        statementbeginend + "try"                     -> Try/catch

        "join" + name                                  -> SQL Branch (must be followed by
a class or variable name)

    The statementbeginend is one of the following symbols: "{" "}" ";"
    The not( indicates that it must not be followed by "("
    -----
*/

public static int calcVG(SysScannerClass scanner)
{
    int cc, symbol, symbolHist_4, symbolHist_3, symbolHist_2, symbolHist_1;
    str strHist_4, strHist_3, strHist_2, strHist_1;
    boolean isSQL = false;

```

```

//Initialize cc to 1
cc = 1;

//Loop through all symbols in the source
symbol = scanner.firstSymbol();
while (symbol)
{
    //Add the new symbol to the symbol and string history
    symbolHist_4 = symbolHist_3;
    symbolHist_3 = symbolHist_2;
    symbolHist_2 = symbolHist_1;
    symbolHist_1 = symbol;

    strHist_4 = strHist_3;
    strHist_3 = strHist_2;
    strHist_2 = strHist_1;
    strHist_1 = scanner.strValue();

    //Find embedded method definitions
    if (CodeMetricVGMethod::isEmbMethodDef(symbolHist_1, symbolHist_2,
symbolHist_3, symbolHist_4, strHist_3))
        cc++;

    //Find out if we are in a SQL statement
    isSQL = CodeMetricVGMethod::isSQLStatement(isSQL, symbolHist_1,
symbolHist_2);

    //Find single symbols that will count towards cc, if they are not with a
SQL statement
    if (isSQL == false)
        switch(symbolHist_1)
        {
            case #QUEST_SYM: case #AND_SYM: case #OR_SYM:
                cc++;
        }

    //Find cases where the first symbol of the statement matches our list
    if (CodeMetricVGMethod::isStatementBeginEnd(symbolHist_2))
        switch(symbolHist_1)
        {
            case #IF_SYM: case #WHILE_SYM: case #FOR_SYM: case #CASE_SYM: case
#DEFAULT_SYM: case #CATCH_SYM:
                cc++;
        }

    //Find else if
    if (CodeMetricVGMethod::isElseIf(symbolHist_1, symbolHist_2))
        cc++;

    //Find SQL "join" constructs
    if (CodeMetricVGMethod::isSQLJoin(symbolHist_1, symbolHist_2,
symbolHist_3))
        cc++;

    //Get the next symbol from the scanner
    symbol = scanner.nextSymbol();
}

return cc;

```

```

}

public static boolean isDataType(int symbol)
{
    boolean ret = false;
    ;

    //Return true if the symbol is a simple datatype
    switch(symbol)
    {
        case #VOID_TYPE_SYM: //void
        case #INT_TYPE_SYM: //int
        case #INT64_TYPE_SYM: //int64
        case #DBL_TYPE_SYM: //real
        case #DATE_TYPE_SYM: //date
        case #STR_TYPE_SYM: //str
        case #GUID_TYPE_SYM: //guid
            ret = true;
    }

    return ret;
}

public static boolean isElseIf(int symbol_1, int symbol_2)
{
    //Return true if the symbols are "else if"
    return symbol_2 == #ELSE_SYM && symbol_1 == #IF_SYM;
}

public static boolean isEmbMethodDef(int symbol_1, int symbol_2, int symbol_3, int
symbol_4, str strVal_3)
{
    boolean ret=false;
    ;

    //Must end with "("
    if (symbol_1 == #LEFT_PAR_SYM)
    {
        //2nd last must be a identifier like "Method1"
        if (symbol_2 == #STD_ID)
        {
            //Before the method definition starts an end of the last statement must
be present
            if (CodeMetricVGMethod::isStatementBeginEnd(symbol_4))
            {
                //Check that there is a valid return type present
                if (TreeNode::isValidObjectName(strVal_3) ||
CodeMetricVGMethod::isDataType(symbol_3))
                {
                    //We have an embedded method definition!!!
                    ret = true;
                }
            }
        }
    }

    return ret;
}

public static boolean isSQLJoin(int symbol_1, int symbol_2, int symbol_3)
{

```

```

;

//[exitsts|notexists] join name
return (symbol_1 == #STD_ID && symbol_2 == #JOIN_SYM && symbol_3 != #EXISTS_SYM
&& symbol_3 != #NOTEXISTS_SYM);
}

public static boolean isSQLStatement(boolean isSQL, int symbol_1, int symbol_2)
{
    boolean ret = isSQL;
    ;

    if (isSQL && (symbol_1 == #LEFTBR_SYM || symbol_1 == #SEMICOLON_SYM))
    {
        //The SQL statement has ended
        ret = false;
    }
    else if(!isSQL)
    {
        //Its the first word of an expression
        if(CodeMetricVGMethod::isStatementBeginEnd(symbol_2))
        {
            //Its a SQL symbol
            switch(symbol_1)
            {
                case #SEARCH_SYM: //select
                case #DELETE_SYM: //delete
                case #UPDATE_SYM: //update_recordset
                case #INSERT_SYM: //insert_recordset
                    ret = true;
            }
        }
        //while select
        else if(symbol_2 == #WHILE_SYM && symbol_1 == #SEARCH_SYM)
            ret = true;
    }

    return ret;
}

public static boolean isStatementBeginEnd(int symbol)
{
    ;
    //Return true, if the symbol is "{", "}" or ";"
    return (symbol == #LEFTBR_SYM || symbol == #RIGHTBR_SYM || symbol ==
#SEMICOLON_SYM);
}

```

Class: CodeMetricSLOCMethod

```

class CodeMetricSLOCMethod extends CodeMethodMetric
{
    //The maximal allowed val for SLOC
    #define.MaxSLOCValue(40)
}

public BPSeverity getBPSeverity()
{
    //Warning
    return BPSeverity::Warning;
}

```



```

public str getBPStr()
{
    str ret;
    int slocVal;

    //Get the value for SLOC
    slocVal = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (slocVal > #MaxSLOCValue)
        ret = sprintf('The number of Source lines (SLOC) of method %1 is %2 (Max.
recommended %3)',node.treeNodeName(),int2str(slocVal),int2str(#MaxSLOCValue));

    return ret;
}

public str getDescription()
{
    //Source Lines of Code
    return 'SLOC';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetricSLOCMethod;
}

public int getValue()
{
    //Get the value for SLOC for the source code
    return CodeMetricSLOCMethod::calcSLOC(node.AOTgetSource());
}

public static int calcSLOC(str sourcecode)
{
    int sloc;
    TextBuffer textBuffer;
    str cfcode;
    str line;
    ;

    //Create TextBuffer and fill with comment-free source code
    cfcode = CodeMetricSLOCMethod::removeComments(sourcecode);
    textBuffer = new TextBuffer();
    textBuffer.setText(cfcode);

    //Get first line
    line = textBuffer.nextToken(false,'\n');

    //Loop through lines
    while(line)
    {
        //If the line is not blank then increase SLOC
        if(strrtrim(strltrim(line)) != '')
            sloc++;

        //Read next line
        line = textBuffer.nextToken(false,'\n');
    }
}

```

```

    return sloc;
}

public static str removeComments(str sourceCode)
{
    str cfcode = ''; //Comment-free code
    SourceCodeChunker chunker = new SourceCodeChunker(sourceCode);
    ;

    //Get all code chunks
    while(chunker.moveNext())
        cfcode += chunker.currentCodeChunk();

    //Return the comment-free code
    return cfcode;
}

```

Class: CodeMetricFI

```

class CodeMetricFI extends CodeClassMetric
{
    //The maximum allowed value for Fan In
    #define.MaxFIValue(50)
}

public str getBPStr()
{
    str ret;
    int val;

    //Get the value for FI for the class
    val = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (val > #MaxFIValue)
        ret = sprintf('Fan In (FI) of class %1 is %2 (Max. recommended
%3)',node.treeNodeName(),int2str(val),int2str(#MaxFIValue));

    return ret;
}

public str getDescription()
{
    //Fan In
    return 'FI';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetricFI;
}

int getValue()
{
    xRefReferences    xReferences;
    xRefPaths        xPaths;
    xRefPaths        xFromPaths;
}

```

```

xRefPath      toLikePath;

str           typeName;
Map           map;
;

//Create a map for holding the type names
map = new Map(Types::String,Types::String);

//Add \* in the end of the path for node to find, and double the amount of \
//This is needed to make the "like" work correctly
toLikePath = strReplace(node.treeNodePath() + '\\*', '\\\\', '\\\\\\');

/* Since Fan-In is a system-level measure, we need to use x-ref from the normal
tables,
and not from the temporary xref
*/
while select xFromPaths
join xReferences where xFromPaths.RecId == xReferences.xRefPathRecId &&
                    (xReferences.Reference == xRefReference::Declaration ||
                     xReferences.Reference == xRefReference::Call)
join xPaths where xPaths.RecId == xReferences.referencePathRecId &&
                (xPaths.Path == node.treeNodePath() ||
                 xPaths.Path like toLikePath
                )
{
    //Get the name of the class/form/table
    typename = SysTreeNode::applObjectPath(xFromPaths.Path);

    //Insert the found type(class) name into the map if it's not already there
    //and if it is not the class itself
    if (!map.exists(typeName) && typeName != node.treeNodePath())
        map.insert(typeName,typeName);
}

//FI = number of other types having a reference to this class
return map.elements();
}

```

Class: CodeMetricNOC

```

class CodeMetricNOC extends CodeClassMetric
{
    //The maximum allowed value for the Number Of Children
    #define.MaxNOCValue(10)
}

public str getBPStr()
{
    str ret;
    int val;

    //Get the value for NOC for the class
    val = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (val > #MaxNOCValue)
        ret = sprintf('The Number Of Children (NOC) of class %1 is %2 (Max.
recommended %3)',node.treeNodeName(),int2str(val),int2str(#MaxNOCValue));
}

```

```

    return ret;
}

public str getDescription()
{
    //Number of children
    return 'NOC';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrCodeMetricNOC;
}

int getValue()
{
    DictClass dict;
    DictClass subDict;
    Enumerator enum;
    int noc = 0;
    ;

    //Create a new dict class
    dict = new DictClass(node.applObjectId());

    //Get an enumerator containing all subclasses
    enum = dict.extendedBy().getEnumerator();

    //Loop through all subclasses
    while(enum.moveNext())
    {
        subDict = new DictClass(enum.current());

        //If the class in an immediate child then increase the count
        if (subDict.extend() == node.applObjectId())
            noc++;
    }

    return noc;
}

```

Class: CodeMetricLCOM

```

class CodeMetricLCOM extends CodeClassMetric
{
    #define.LCOMValue(1)
}

public str getBPStr()
{
    str ret;
    int val;

    //Get the value for LCOM for the node
    val = this.getValue();

    //If the value is greater then #LCOMValue, return an error string
    //We will also allow value of zero, since this might indicate a collection of
    static methods
    if (val > #LCOMValue)

```

```

        ret = sprintf('The Lack of Cohesion Of Methods (LCOM) of class %1 is %2
(Recommended %3)',node.treeNodeName(),int2str(val),int2str(#LCOMValue));

    return ret;
}

public str getDescription()
{
    //Lack of Cohesion Of Methods
    return 'LCOM';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorcodeMetricLCOM;
}

int getValue()
{
    GraphUndirected graph = new GraphUndirected();
    xRefTmpReferences thisReferences;

    str graphNodeVal;
    GraphNode fromGraphNode;
    GraphNode toGraphNode;

    //Make sure xRef is updated for the class
    this.initTmpXRef();

    thisReferences.setTmpData(tmpxRefReferences);
    while select thisReferences order by Reference
    {

        //Declaration of class level variables so add node
        if(this.isClassLevelVar(thisReferences))
        {
            graphNodeVal = thisReferences.name;
            graph.addNode(graphNodeVal);
        }
        //Definition of class method so add node
        else if(this.isMethodDef(thisReferences))
        {
            graphNodeVal = thisReferences.Path;
            graph.addNode(graphNodeVal);
        }
        //Call to class method so add edge
        else if(this.isInternalMethodCall(thisReferences))
        {
            fromGraphNode = graph.findNodeOnData(thisReferences.Path);
            toGraphNode = graph.findNodeOnData(node.treeNodePath() + '\\\' +
thisReferences.name);

            //If toGraphNode is null then it is a call to an inherited method, else
it is a regular internal method call
            graph.addEdge(fromGraphNode,toGraphNode);
        }
        //Read or write of variable
        else if(thisReferences.Reference == xRefReference::Read ||
thisReferences.Reference == xRefReference::Write)
        {

```

```

        //If the variable can be found as a node, then it must be a class-level
variable
        toGraphNode = graph.findNodeOnData(thisReferences.name);
        fromGraphNode = graph.findNodeOnData(thisReferences.Path);

        graph.addEdge(fromGraphNode,toGraphNode);
    }
}

//Start a Depth First Search on the graph
graph.runDFS();

//LCOM = the number of connected components = the number of sub-graphs
return graph.nodesWithoutParent();
}

private boolean isClassLevelVar(xRefTmpReferences thisReferences)
{
    ;
    //Declaration of class level variables
    return (thisReferences.Reference == xRefReference::Declaration &&
thisReferences.Path == node.treeNodePath() + '\\classDeclaration');
}

private boolean isInternalMethodCall(xRefTmpReferences thisReferences)
{
    ;
    //Call to class method
    return (thisReferences.Reference == xRefReference::Call &&
thisReferences.ParentName == node.treeNodeName());
}

private boolean isMethodDef(xRefTmpReferences thisReferences)
{
    boolean ret = false;
    SysMethodInfo sysMethodInfo;
    ;
    //Definition of class method
    if (thisReferences.Reference == xRefReference::Definition &&
thisReferences.Kind == xRefKind::ClassInstanceMethod)
    {
        //Get method info to find out if the method is abstract
        sysMethodInfo = new
SysMethodInfo(UtilElementType::ClassInstanceMethod,0,'');
        sysMethodInfo.setMethod(TreeNode::findNode(thisReferences.Path));

        if (!sysMethodInfo.isAbstract())
            ret = true; //It must be a "normal" method
    }

    return ret;
}
}

```

Class: CodeMetricRFC

```

class CodeMetricRFC extends CodeClassMetric
{
    #define.MaxRFCValue(50)
}

public BPSeverity getBPSeverity()

```

```

{
    //Warning
    return BPSeverity::Warning;
}

public str getBPStr()
{
    str ret;
    int val;

    //Get the value for RFC for the source code
    val = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (val > #MaxRFCValue)
        ret = sprintf('The Response For Class (RFC) of class %1 is %2 (Max.
recommended %3)',node.treeNodeName(),int2str(val),int2str(#MaxRFCValue));

    return ret;
}

public str getDescription()
{
    //Response For Class
    return 'RFC';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetricRFC;
}

int getValue()
{
    xRefTmpReferences thisReferences;

    DictClass dict;
    int methodNo;

    Map map;
    str methodName;
    ;

    //Make sure xRef is updated for this class
    this.initTmpXRef();

    //Create a map for holding the method names
    map = new Map(Types::String,Types::String);

    //Add all the methods of the class to the list
    dict = new DictClass(node.applObjectId());
    for(methodNo = 1;methodNo <= dict.objectMethodCnt();methodNo++)
    {
        //The classDeclaration should not be included
        if (dict.objectMethod(methodNo) != 'classDeclaration')
        {
            methodName = node.treeNodeName() + '\\\\' + dict.objectMethod(methodNo);
            map.insert(methodName, methodName);
        }
    }
}

```

```

//Get the paths of the objects used
thisReferences.setTmpData(tmpxRefReferences);
while select thisReferences where thisReferences.Reference ==
xRefReference::Call
{
    //Get the method name (path)
    methodName = thisReferences.ParentName + '\\\' + thisReferences.name;

    //If the type does not already exists in the map then insert it
    if (!map.exists(methodName))
        map.insert(methodName,methodName);

}

//RFC = number of distinct possible method calls
return map.elements();
}

```

Class: CodeMetricCBO

```

class CodeMetricCBO extends CodeClassMetric
{
    #define.MaxCBOValue(20)
}

public BPSeverity getBPSeverity()
{
    //Warning
    return BPSeverity::Warning;
}

public str getBPStr()
{
    str ret;
    int val;

    //Get the value for CBO for the source code
    val = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (val > #MaxCBOValue)
        ret = sprintf('The Coupling Between Objects (CBO) of class %1 is %2 (Max.
recommended %3)',node.treeNodeName(),int2str(val),int2str(#MaxCBOValue));

    return ret;
}

public str getDescription()
{
    //Coupling Between Objects
    return 'CBO';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetricCBO;
}

```



```

int getValue()
{
    xRefTmpReferences thisReferences;

    Map map;
    str typeName;
    ;

    //Make sure xRef is updated for this class
    this.initTmpXRef();

    //Create a map for holding the type names
    map = new Map(Types::String,Types::String);

    //Get the paths of the objects used
    thisReferences.setTmpData(tmpxRefReferences);
    while select thisReferences where thisReferences.Reference ==
xRefReference::Read
    {
        //Get the type name (path)
        if (thisReferences.ParentName == '')
            typeName = thisReferences.name;
        else
            typeName = thisReferences.ParentName;

        //If the type does not already exists in the map then insert it
        if (!map.exists(typeName))
            map.insert(typeName,typeName);
    }

    //CBO = number of distinct types
    return map.elements();
}

```

Class: CodeMetricWMC

```

class CodeMetricWMC extends CodeClassMetric
{
    #define.MaxWMCValue(50)
}

public BPSeverity getBPSeverity()
{
    //Warning
    return BPSeverity::Warning;
}

public str getBPStr()
{
    str ret;
    int val;

    //Get the value for WMC for the class
    val = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (val > #MaxWMCValue)

```

```

        ret = sprintf('The Weighted Methods per Class (WMC) of class %1 is %2 (Max.
recommended %3)',node.treeNodeName(),int2str(val),int2str(#MaxWMCValue));

    return ret;
}

public str getDescription()
{
    //Weighted Methods per Class
    return 'WMC';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorCodeMetricWMC;
}

int getValue()
{
    CodeMetricVGMethod vgMetric = new CodeMetricVGMethod();
    int sumVG = 0;
    TreeNode child;
    ;

    //Loop through all child methods
    child = node.AOTfirstChild();
    while(child)
    {
        if (child.treeNodeName() != 'classDeclaration')
        {
            //Pass the method to CodeMetricCCMethod
            vgMetric.setElement(child);

            //Get the value
            sumVG += vgMetric.getValue();
        }

        //Get next child method
        child = child.AOTnextSibling();
    }

    //Return sum of complexities
    return sumVG;
}

```

Class: CodeMetricDIT

```

class CodeMetricDIT extends CodeClassMetric
{
    //The maximum allowed value for the Depth of Inheritance Tree
    #define.MaxDITValue(8)
}

public BPSeverity getBPSeverity()
{
    //Warning
    return BPSeverity::Warning;
}

public str getBPStr()

```

```

{
    str ret;
    int ditVal;

    //Get the value for DIT for the class
    ditVal = this.getValue();

    //If the value exceeds the threshold limit, return an error string
    if (ditVal > #MaxDITValue)
        ret = sprintf('The Depth of Inheritance Tree (DIT) of class %1 is %2 (Max.
recommended %3)',node.nodeName(),int2str(ditVal),int2str(#MaxDITValue));

    return ret;
}

public str getDescription()
{
    //Depth of Inheritance Tree
    return 'DIT';
}

public int getErrorCode()
{
    //Errorcode defined in macro SysBPCheck
    return #BPErrorcodeMetricDIT;
}

public int getValue()
{
    DictClass dict = new DictClass(node.applObjectId());
    int depth = 1; //All classes inherit from Object
    ;

    //Repeat as long as we can go up in the hierarchy
    while(dict.extend())
    {
        //Increase depth if its not object
        if (dict.extend() != classNum(object))
        {
            depth++;
        }

        //Create a DictClass for the parent
        dict = new DictClass(dict.extend());
    }

    return depth;
}

```

Other

Class: GraphUndirected

```
class GraphUndirected
{
    List nodes; //List of nodes
    List edges; //List of edges

    int dfsTime; //For time-keeping in DFS

    #define.white(0)
    #define.grey(1)
    #define.black(2)
}

public GraphEdge addEdge(GraphNode node1, GraphNode node2)
{
    GraphEdge edge = null;
    ;

    //If there is no edge with that that already, then create a new
    if (node1 != null && node2 != null)
    {
        if (!this.findEdge(node1, node2))
        {
            edge = new GraphEdge();
            edge.setNode1(node1);
            edge.setNode2(node2);

            //Add the edge to the list
            edges.addEnd(edge);
        }
    }

    return edge;
}

public GraphNode addNode(anytype data)
{
    GraphNode newNode = null;

    //Try to find an existing node with the same data
    newNode = this.findNodeOnData(data);

    //If the data is not present in a node, then create a new node
    if (newNode == null)
    {
        newNode = new GraphNode();
        newNode.setData(data);

        //Add to the list
        nodes.addEnd(newNode);
    }

    return newNode;
}
```

```

private void DFS(GraphNode node)
{
    List neighbours;
    GraphNode neighbourNode;
    ListEnumerator enum;
    ;

    //Set the start time and change color to grey
    dfsTime++;
    node.setTimeDiscovered(dfsTime);
    node.setColor(#grey);

    //Get the list of neighbours to this node
    neighbours = this.getListOfNeighbours(node);
    if (neighbours.elements() > 0)
    {
        enum = neighbours.getEnumerator();

        //Loop through all the neighbours
        while(enum.moveNext())
        {
            neighbourNode = enum.current();

            //If the neighbour has not been discovered then perform DFS recursively
            if (neighbourNode.getColor() == #white)
            {
                neighbourNode.setParent(node);
                this.DFS(neighbourNode);
            }
        }
    }

    //Node completed so set finish time and change color to black
    dfsTime++;
    node.setTimeFinished(dfsTime);
    node.setColor(#black);
}

public GraphEdge findEdge(GraphNode node1, GraphNode node2)
{
    ListEnumerator enum = edges.getEnumerator();
    GraphEdge edge;
    int edgeNum;

    //Loop through all the edges
    while(enum.moveNext())
    {
        edge = enum.current();

        //If the edge contains the two nodes then we're done
        if ((edge.getNode1() == node1 && edge.getNode2() == node2) ||
            (edge.getNode1() == node2 && edge.getNode2() == node1))
            return edge;
    }

    return null;
}

```

```

public GraphNode findNodeOnData(anytype findData)
{
    ListEnumerator enum = nodes.getEnumerator();
    GraphNode node;
    int nodeNum;

    //Loop through all the nodes
    while(enum.moveNext())
    {
        node = enum.current();

        //If the node contains the data that we're done
        if (node.getData() == findData)
            return node;
    }

    return null;
}

public List getListOfNeighbours(GraphNode node)
{
    ListEnumerator enum = edges.getEnumerator();
    GraphEdge edge;
    int edgeNum;

    //Create new list to hold the found nodes
    List neighbours = new List(Types::Class);

    //Loop through the edges
    while(enum.moveNext())
    {
        edge = enum.current();

        //If node1 or node2 equals the node, then add the other end of the edge to
the list
        if (edge.getNode1() == node)
            neighbours.addEnd(edge.getNode2());
        else if (edge.getNode2() == node)
            neighbours.addEnd(edge.getNode1());
    }

    return neighbours;
}

public List getNodes()
{
    //Return the list of nodes
    return nodes;
}

void new()
{
    ;

    //Initialize the lists of nodes and edges
    nodes = new List(Types::Class);
    edges = new List(Types::Class);
}

public int nodesWithoutParent()

```

```

{
    ListEnumerator enum = nodes.getEnumerator();
    GraphNode node;
    int nodeNum;
    int noParent = 0;

    //Loop through all the nodes
    while(enum.moveNext())
    {
        node = enum.current();

        //Increase the count if the parent is null
        if (node.getParent() == null)
            noParent++;
    }

    return noParent;
}

public void runDFS()
{
    ListEnumerator enum = nodes.getEnumerator();
    GraphNode node;
    int nodeNum;

    //Loop through all the nodes and initialize
    while(enum.moveNext())
    {
        node = enum.current();
        node.setColor(#white);
        node.setParent(null);
    }

    //Reset time
    dfsTime = 0;

    //Loop through the nodes again and perform DFS if the color is white
    enum = nodes.getEnumerator();
    while(enum.moveNext())
    {
        node = enum.current();
        if (node.getColor() == #white)
            this.DFS(node);
    }
}
}

```

Class: GraphEdge

```

class GraphEdge
{
    GraphNode node1;
    GraphNode node2;
}

public GraphNode getNode1()
{
    //Return the first node
    return node1;
}

```

```

}

public GraphNode getNode2()
{
    //Return the second node
    return node2;
}

public void setNode1(GraphNode _node1)
{
    //Save in class variable
    node1 = _node1;
}

public void setNode2(GraphNode _node2)
{
    //Save in class variable
    node2 = _node2;
}

```

Class: GraphNode

```

class GraphNode
{
    anytype data;          //Payload

    int color;            //For graph traversal
    int timeDiscovered;  //For graph traversal
    int timeFinished;    //For graph traversal
    GraphNode parent;    //For graph traversal
}

public int getColor()
{
    //Return color for graph traversal
    return color;
}

public anytype getData()
{
    //Return the payload
    return data;
}

public GraphNode getParent()
{
    //Return the parent
    return parent;
}

public int getTimeDiscovered()
{
    //Return the timeDiscovered
    return timeDiscovered;
}

public int getTimeFinished()
{
    //Return the timeFinished
    return timeFinished;
}

```



```

public void setColor(int _color)
{
    ;
    //Save color in class variable
    color = _color;
}

public void setData(anytype _data)
{
    //Save in class variable
    data = _data;
}

public void setParent(GraphNode _parent)
{
    //Save in class variable
    parent = _parent;
}

public void setTimeDiscovered(int _timeDiscovered)
{
    //Save in class variable
    timeDiscovered = _timeDiscovered;
}

public void setTimeFinished(int _timeFinished)
{
    //Save in class variable
    timeFinished = _timeFinished;
}

```

Class: ClassInstanciator

```

class ClassInstanciator
{
}

static List createSubClassInstances(classId superClassId)
{
    List instanceList;

    Set set;
    SetEnumerator enumerator;
    SysDictClass dictClass;
    ;

    //Create a list that will hold instances of the classes
    instanceList = new List(Types::Class);

    //Get a Set containing the ids for classes that implements the superclass
    set = SysDictClass::getImplements(superClassId);
    enumerator = set.getEnumerator();
    while(enumerator.moveNext())
    {
        //Create a new SysDictClass for the classid
        dictClass = new SysDictClass(enumerator.current());

        if (dictClass.id() != superClassId)
        {

```

```

        //Add a new instance of the implementing class to the
codeMethodMetricList
        instanceList.addEnd(dictClass.makeObject());
    }
}

return instanceList;
}

```

Class: StringUtil

```

class StringUtil
{
    //No class level variables, since this class is used for grouping of related
string function
}

public static int CountOccurences(str sourcetxt, str findtxt)
{
    int cnt = 0;
    int scanpos = 1;
    ;

    //Find first occurence
    scanpos = strstr(sourcetxt,findtxt,scanpos,strlen(sourcetxt) - scanpos + 1);
    while(scanpos > 0)
    {
        //Add one to the count
        cnt++;

        //Rescan
        scanpos = strstr(sourcetxt,findtxt,scanpos+1,strlen(sourcetxt) - scanpos);
    }

    return cnt;
}

```

Class: SourceCodeChunker

```

class SourceCodeChunker
{
    str source;           //Source code to work on
    int sourcelen;       //Length of the source code, so we don't need to use
strlen(sourcecode) all the time

    int fromPos;         //The current position in the source code
    int linecount;       //Number of lines (newline characters) read

    str currentCode;     //Last created code chunk
    str currentComment;  //Last created comment chunk

    int startLineCode;   //The line number where the code starts
    int startLineComment; //The line number where the comment starts

    #define.commentLength(2) //For use with the getNext function
}

public int codeStartLine()
{
    //Return the line number where the code chunk starts
}

```

```

    return startLineCode;
}

public int commentStartLine()
{
    //Return the line number where the code chunk starts
    return startLineComment;
}

public str currentCodeChunk()
{
    //Returns the last created code chunk
    return currentCode;
}

public str currentCommentChunk()
{
    //Returns the last created comment chunk
    return currentComment;
}

//Finds the minimum value, where value <> 0
private int findMinPos(container vals)
{
    int minval = 0;
    int val;
    int i;
    ;

    //Loop for all value in the container
    for(i=1;i<=conlen(vals);i++)
    {
        //Get value from container
        val = conpeek(vals,i);

        //Check if the val is a new minimum
        if (val < minval && val != 0 || minval == 0)
            minval = val;
    }

    return minval;
}

//Finds the end of a quoted or double-quoted string
private int findStrEnd(str sourceCode, int startPos, str quote, str presymbol)
{
    #define.escapedWidth(2)
    int endPos=startPos;
    ;

    while(endPos <= strlen(sourceCode))
    {
        //Done when we find the end quote
        if (substr(sourceCode,endPos,1) == quote)
            break;

        //Verbose strings has no escape characters
        if (presymbol != '@')
        {
            //If \ or ' is escaped then ignore the next character
            switch(substr(sourceCode,endPos,#escapedWidth))

```

```

        {
            case '\\\\':
            case '\\' + quote:
                endPos++;
        }
    }

    //Next character
    endPos++;
}

return endPos;
}

int lineCount()
{
    //Number of lines (newline characters) read
    return lineCount;
}

public boolean moveNext()
{
    int scanPos;

    //Reset the output variables
    this.resetOutput();

    if(fromPos < sourceLen)
    {
        //Scan for comments and strings
        scanPos = this.scanForCommentsAndQuotes();

        //Repeat until we have found a comment
        while(scanPos > 0 && currentComment == '')
        {
            switch(substr(source,scanPos,#commentLength))
            {
                case '/*':
                    //Start of multi line comment found, so insert the text and
search for comment end
                    currentCode += substr(source,fromPos,scanPos-frompos);
                    fromPos = strstr(source,'*',scanPos,sourceLen -
scanPos)+#commentLength;
                    currentComment = substr(source,scanPos,frompos-scanPos);
                    break;

                case '///':
                    //Start of multi line comment found, so insert the text and
search for line end
                    currentCode += substr(source,fromPos,scanPos-frompos);
                    fromPos = strstr(source,'\n',scanPos,sourceLen - scanPos) > 0
? strstr(source,'\n',scanPos,sourceLen - scanPos) : sourceLen +1;
                    currentComment = substr(source,scanPos,frompos-scanPos);
                    break;

                default:
                    //All text until the next quote pos will be included, regarding
if it is a comment
                    scanPos = this.findStrEnd(source,
scanPos+1,substr(source,scanPos,1),substr(source,scanPos-1,1));
                    currentCode += substr(source,fromPos,scanPos-fromPos+1);
            }
        }
    }
}

```

```

        fromPos = scanPos + 1;
    }

    //Rescan
    scanPos = this.scanForCommentsAndQuotes();
}

if (currentComment == '')
{
    //No comments was found, so we must copy the last part of the
sourcecode to the currentCode
    currentCode += substr(source,fromPos,sourcelen-frompos+1);
    fromPos = sourceLen;
}

//Add to the linecount
lineCount += StringUtil::CountOccurences(currentCode,'\n');
startLineComment = lineCount;
lineCount += StringUtil::CountOccurences(currentComment,'\n');

return true;
}

return false;
}

void new(str sourceCode)
{
    ;

    //Set source and calculate sourcelen
    source = sourceCode;
    sourcelen = strlen(source);

    //Initialize the counters
    fromPos = 1;
    linecount = 1;

    //Reset all the output variables
    this.resetOutput();
}

private void resetOutput()
{
    //Clear all the output variables
    currentCode = '';
    currentComment = '';
    startLineComment = lineCount;
    startLineCode = lineCount;
}

private int scanForCommentsAndQuotes()
{
    int singlePos, multiPos, quotePos, doubleQuotePos;
    ;

    //Find the next positions of comments and quotes
    multiPos = strstr(source, '/*', fromPos, sourcelen);
    singlePos = strstr(source, '//', fromPos, sourcelen);
    quotePos = strstr(source, '\'', fromPos, sourcelen);
    doubleQuotePos = strstr(source, '"', fromPos, sourcelen);
}

```

```
//Return the first position that is not zero  
return this.findMinPos([multiPos, singlePos, quotePos, doubleQuotePos]);  
}
```

Statistics

Form: CodeMetricsResults

```
public class FormRun extends ObjectRun
{
}

void startGenerateTeamStats(str filename)
{
    Map statMap;
    MapIterator metricIterator;
    MapIterator itemIterator;
    str metric;

    CodeMetricStatItem item;
    TmpCodeMetricsTeamStat stat;

    if (filename != '')
    {
        //Start calculation
        statMap = CodeMetricTeamStatGenerator::statByTeam(filename);

        //Clear the team statistics table
        delete_from stat;

        metricIterator = new MapIterator(statMap);
        while(metricIterator.more())
        {
            metric = metricIterator.key();

            itemIterator = new MapIterator(metricIterator.value());
            while(itemIterator.more())
            {
                item = itemIterator.value();

                //Clear record
                stat.clear();

                //Fill with values
                stat.Metric = metric;
                stat.Team = item.getName();
                stat.Prefix = item.getPrefixName();
                stat.Occurences = item.getItemCount();
                stat.ValueSum = item.getSum();
                stat.AverageValue = item.getAvg();
                stat.MaxValue = item.getMax();

                if (item.getItemCount() == 0)
                    stat.MinValue = 0;
                else
                    stat.MinValue = item.getMin();

                //Insert into table
                stat.insert();

                //Get next item
                itemIterator.next();
            }
        }
    }
}
```

```

        }

        metricIterator.next();
    }

    //Refresh grid datasource
    tmpCodeMetricsTeamStat_ds.research();
}

container fileNameLookupFilter()
{
    #File
    Filename    filepath;
    Filename    filename;
    Filename    fileExtention;

    //Extract path, filename and extension from any existing filename
    [filepath, fileName, fileExtention] =
Global::fileNameSplit(teamFileName.text());

    //Set default file extension to .txt
    if (!fileExtention)
    {
        fileExtention = #txt;
    }

    return [WinAPI::fileType(fileExtention),#AllFileName+fileExtention,
#AllFilesExt, #AllFilesType];
}

// AOSRunMode::client
str fileNameLookupInitialPath()
{
    #WinAPI

    Filename    filepath;
    Filename    filename;
    Filename    fileType;

    [filepath, fileName, fileType] = Global::fileNameSplit(teamFileName.text());

    // Default path
    if (!filePath)
    {
        filePath = WinAPI::getFolderPath(#CSIDL_Personal);
    }

    return filePath;
}

// AOSRunMode::client
str fileNameLookupTitle()
{
    return teamFileName.label();
}

str fileNameLookupFilename()
{
    Filename    filepath;
    Filename    filename;

```



```

    Filename    fileType;

    //Split name into the tree parts
    [filepath, fileName, fileType] = fileNameSplit(teamFileName.text());

    return fileName + fileType;
}

void startGeneration()
{
    TreeNode startNode;
    ;

    startNode = TreeNode::findNode(treePath.text());
    if (startNode)
    {
        //Clear the data first
        ttsbegin;
        delete_from TmpCodeMetrics;
        ttscommit;

        //Start generating
        CodeMetricGenerator::generateMetrics(startNode);

        //Update the grid
        tmpCodeMetrics_ds.research();
        grid.update();
    }
    else
    {
        error('Invalid path to TreeNode');
    }
}

public int mouseDbClick(int _x, int _y, int _button, boolean _ctrl, boolean
_shift)
{
    int ret;
    TreeNode node;

    ret = super(_x, _y, _button, _ctrl, _shift);

    //Find the treenode that corresponds to line that was clicked
    node = TreeNode::findNode(tmpCodeMetrics.TreeNodePath);

    //Edit the node
    node.AOTedit();

    return ret;
}

void clicked()
{
    super();

    //Start the generation
    element.startGeneration();
}

void clicked()

```

```

{
    super();

    //Call method to start generation of statistics per team/prefix
    element.startGenerateTeamStats(teamFileName.text());
}

```

Class: CodeMetricGenerator

```

class CodeMetricGenerator
{
    //No instance variables, since all methods are static
}

public static void doClassMetric(TreeNode treeNode, List codeMetricList)
{
    CodeClassMetric codeMetric;
    ListEnumerator metricEnum;

    xRefUpdateTmpReferences tmpUpdate;
    xRefTmpReferences tmpxRefReferences;
    ;

    //Create tmp cross references for the entire class (for optimization)
    tmpUpdate = new xRefUpdateTmpReferences();
    tmpUpdate.fillTmpxRefReferences(treeNode);
    tmpxRefReferences = tmpUpdate.allTmpxRefReferences();

    //Loop through all the metric classes that are available
    metricEnum = codeMetricList.getEnumerator();
    while(metricEnum.moveNext())
    {
        //Cast as CodeClassMetric
        codeMetric = SysDictClass::as(metricEnum.current(),
classNum(CodeClassMetric));

        //Pass the tree node of the class to check
        codeMetric.setElement(treeNode);

        //Pass the cross references to the metric class
        codeMetric.setXRefTmpReferences(tmpxRefReferences);

        //Get the value and insert into database
        codeMetricGenerator::saveInDB(codeMetric, treeNode.treeNodePath());
    }
}

public static void doMethodMetric(TreeNode treeNode, List codeMetricList)
{
    CodeMethodMetric codeMetric;
    ListEnumerator metricEnum;

    SysScannerClass scanner;
    ;

    //Create scanner
    scanner = new SysScannerClass(treeNode);

    //Loop through all the metric classes that are available
    metricEnum = codeMetricList.getEnumerator();
    while(metricEnum.moveNext())

```

```

    {
        //Cast as CodeMethodMetric
        codeMetric = SysDictClass::as(metricEnum.current(),
classNum(CodeMethodMetric));

        //Pass the tree node of the class to check
        codeMetric.setElement(treeNode);

        //Pass the scanner for optimization
        codeMetric.setScanner(scanner);

        //Get the value and insert into database
        codeMetricGenerator::saveInDB(codeMetric, treeNode.treeNodePath());
    }
}

public static void generateMetrics(TreeNode startnode)
{
    //Create lists with instances of CodeMethodMetric/CodeClassMetric classes
    List codeMethodMetricList =
ClassInstanciator::createSubClassInstances(classNum(CodeMethodMetric));
    List codeClassMetricList =
ClassInstanciator::createSubClassInstances(classNum(CodeClassMetric));

    TreeNode treeNode;
    TreeNodeTraverser treeNodeTraverser;

    #avifiles
    SysOperationProgress simpleProgress;
    ;

    //Create a progress indicator
    simpleProgress = SysOperationProgress::newGeneral(#aviUpdate, 'Metrics',
startnode.AOTchildNodeCount());

    //Traverse the startnode
    treeNodeTraverser = new TreeNodeTraverser(startnode);
    while (treeNodeTraverser.next())
    {
        //Get the current node
        treeNode = treeNodeTraverser.currentNode();

        //Increment and set text on progress
        simpleProgress.incCount();
        simpleProgress.setText(treeNode.treeNodePath());

        //Perform different actions depending on the type of TreeNode
        switch (treeNode.handle())
        {
            case classnum(MemberFunction):
                if (treeNode.treeNodeName() != 'classDeclaration')
                    CodeMetricGenerator::doMethodMetric(treeNode,
codeMethodMetricList);
                break;
            case classnum(ClassNode):
                CodeMetricGenerator::doClassMetric(treeNode, codeClassMetricList);
                break;
        }
    }

    //Done!!
}

```

```

}

public static void saveInDB(CodeMetricBase codeMetric, TreeNodePath path)
{
    TmpCodeMetrics tmpCodeMetrics;
    ;

    //Perform the check
    tmpCodeMetrics.Value = codeMetric.getValue();

    //Add standard info and insert into the table
    tmpCodeMetrics.Metric = codeMetric.getDescription();
    tmpCodeMetrics.TreeNodePath = path;
    tmpCodeMetrics.insert();
}

```

Class: CodeMetricStatItem

```

class CodeMetricStatItem
{
    str prefixName;
    str groupName;
    int itemCount;
    int minValue;
    int maxValue;
    int valueSum;

    #define.infinity(9999999)
}

public void addValue(int value)
{
    //Increase count
    itemCount++;

    //Add value to sum
    valueSum += value;

    //Set min and max
    if(value < minValue)
        minValue = value;
    if(value > maxValue)
        maxValue = value;
}

public real getAvg()
{
    real avgvalue = 0;

    //If the itemCount is greater than zero, then calculate the average value
    if (itemCount > 0)
        avgValue = valueSum / itemCount;

    return avgvalue;
}

int getItemCount()
{
    //Return the number of items
    return itemCount;
}

```

```

int getMax()
{
    //Return the maximum value recorded
    return maxValue;
}

int getMin()
{
    //Return the minimum value recorded
    return minValue;
}

str getName()
{
    //Return the name of the team
    return groupName;
}

public str getPrefixName()
{
    //Return the name of the prefix
    return prefixName;
}

int getSum()
{
    //Return the sum of recorded values
    return valueSum;
}

void new(str _groupName, str _prefixName)
{
    ;

    //Initialize values
    prefixName = _prefixName;
    groupName = _groupName;
    maxValue = 0;
    valueSum = 0;
    itemCount = 0;

    //Set the minvalue to a high number so we can track the actual min value
    minValue = #infinity;
}

```

Class: CodeMetricTeamStatGenerator

```

class CodeMetricTeamStatGenerator
{
}

static str findPrefix(str path, Map teamPrefixMap)
{
    #define.firstPostFixPos(2)

    str prefix = '';
    str okPrefix = '';
    boolean ok;
    int pos;
}

```

```

//Get the object name from the path
str objName = SysTreeNode::ap1ObjectName(path);

//Loop through all prefixes in the map
MapIterator prefixIterator = new MapIterator(teamPrefixMap);
while(prefixIterator.more())
{
    prefix = prefixIterator.key();
    ok = false;

    if (strscan(prefix, '*', 1, 1) == 1)
    {
        //Is really a postfix
        pos = strscan(objname, substr(prefix, #firstPostFixPos, strlen(prefix)-
1), 1, strlen(objName));
        if (pos > 0 && pos == (strlen(objName) - strlen(prefix) +
#firstPostFixPos) )
            ok = true;
    }
    else if (strscan(objName, prefix, 1, strlen(objName)) == 1)
        ok = true;

    //If match found and its longer than the previous one, and a prefix is not
overriding a postfix
    if (ok == true && strlen(prefix) >= strlen(okPrefix) && !(
(strscan(okPrefix, '*', 1, 1) == 1 && strscan(prefix, '*', 1, 1) == 0)) )
        okPrefix = prefix;

    //Read the nex prefix
    prefixIterator.next();
}

//Return the found prefix
return okPrefix;
}

public static Map initStatMap(Map teamPrefixMap)
{
    CodeMetricStatItem newItem;

    //Create new map for holding maps of CodeMetricStatItems per metric
    Map metricStatMap = new Map(Types::String, Types::Class);

    Map statMap;           //Map for holding CodeMetricStatItems per team
    MapIterator iterator; //Iterator for looping through prefix/team names

    SetEnumerator metricEnumerator;
    DictClass dict;
    str metric;

    //Loop through available metrics
    metricEnumerator =
SysDictClass::getImplements(classNum(CodeMetricBase)).getEnumerator();
    while(metricEnumerator.moveNext())
    {
        //Get metric name
        dict = new DictClass(metricEnumerator.current());
        metric = dict.callStatic('getDescription');

        //New map for this metric

```

```

    statMap = new Map(Types::String,Types::Class);

    //Loop through team names
    iterator = new MapIterator(teamPrefixMap);
    while(iterator.more())
    {
        //Create new CodeMetricStatItem and insert int map
        newItem = new CodeMetricStatItem(iterator.value(),iterator.key());
        statMap.insert(newItem.getPrefixName(),newItem);

        iterator.next();
    }

    //Add statMap to metricStatMap
    metricStatMap.insert(metric, statMap);
}

return metricStatMap;
}

public static Map loadPrefixMap(str _fileName)
{
    #define.prefixrecordLen(2)
    #file

    Map map = new Map(Types::String,Types::String);
    Io file;
    container data;

    //Check that the file exists
    if (WinAPI::fileExists(_fileName))
    {
        file = new TextIo(_fileName, #io_read);

        //Each record is on a single line, and field are delimited by ';'
        file.inRecordDelimiter('\r\n');
        file.inFieldDelimiter(';');

        //Loop through all lines in the file
        while(file.status() == IO_Status::Ok)
        {
            data = file.read();
            if (conlen(data) == #prefixrecordLen)
            {
                map.insert(conpeek(data,#prefixrecordLen),conpeek(data,1));
            }
        }

        return map;
    }

}

public static Map statByTeam(str _teamFileName)
{
    //Load map with prefix/team pairs from file
    Map teamPrefixMap =
CodeMetricTeamStatGenerator::loadPrefixMap(_teamFileName);

    //Get map to hold maps of CodeMetricStatItems per team per metric
    Map statMap = CodeMetricTeamStatGenerator::initStatMap(teamPrefixMap);
    Map metricMap;
}

```

```

CodeMetricStatItem statItem;

str path = '';
str team;
str prefix;

TmpCodeMetrics result;

#avifiles
SysOperationProgress simpleProgress;
;

//Create a progress indicator
select count(value) from result;
simpleProgress = SysOperationProgress::newGeneral(#aviUpdate, 'Statistics',
result.Value);

//Loop through all records in tmpCodeMetrics to decide which prefix/metric map
they should be added to
while select result order by TreeNodePath, Metric
{
    if (result.TreeNodePath != path)
    {
        //Save the path
        path = result.TreeNodePath;

        //Find the team name from prefix map
        prefix = CodeMetricTeamStatGenerator::findPrefix(path, teamPrefixMap);
    }

    //Increment and set text on progress
    simpleProgress.incCount();
    simpleProgress.setText(path);

    //Get the map for the metric (ie. SLOC)
    metricMap = statMap.lookup(result.Metric);

    //Get statItem from prefix
    statItem = metricMap.lookup(prefix);

    if (statItem != null)
    {
        //Update item
        statItem.addValue(result.Value);
    }
}

return statMap;
}

```


Unit tests

Class: CodeMetricTeamStatGeneratorTest

```
class CodeMetricTeamStatGeneratorTest extends XUnitDevTest
{
    #define.TeamPrefixFile('c:\\teamlist.txt')
}

void testFindPrefix()
{
    Map prefixMap = CodeMetricTeamStatGenerator::loadPrefixMap('c:\\teamlist.txt');

    //Test of prefixes
    this.assertEquals('',
CodeMetricTeamStatGenerator::findPrefix('\\Classes\\kjhadkjhasdkjashd',
prefixMap),'No prefix should be found');
    this.assertEquals('SysSign',
CodeMetricTeamStatGenerator::findPrefix('\\Classes\\SysSignDialogForm',
prefixMap),'SysSign should be found');
    this.assertEquals('Sys',
CodeMetricTeamStatGenerator::findPrefix('\\Classes\\SysShell', prefixMap),'Sys
should be found');

    //Test of postfixes
    this.assertEquals('*FI',
CodeMetricTeamStatGenerator::findPrefix('\\Classes\\PaymMoneyTransferSlip_FI',
prefixMap),'*FI should be found');
}

void testInitStatMap()
{
    Map prefixMap = CodeMetricTeamStatGenerator::loadPrefixMap(#TeamPrefixFile);

    //Check that a map with the statistics is actually created
    Map statMap = CodeMetricTeamStatGenerator::initStatMap(prefixMap);
    this.assertNotEqual(0,statMap.elements(),'Map with statistics should not be
empty');
}

void testLoadPrefixMap()
{
    //Load the prefix map and check that it is not empty
    Map result = CodeMetricTeamStatGenerator::loadPrefixMap(#TeamPrefixFile);
    this.assertNotEqual(0,result.elements(),'Map with team/prefixes should not be
empty');
}
```

Class: CodeMetricGeneratorTest

```
class CodeMetricGeneratorTest extends XUnitDevTest
{
}

void testGenerateMetrics()
```

```

{
    TmpCodeMetrics tmp;

    //Make sure to clean up the TmpCodeMetrics before we start
    delete_from tmp;

    //Start a generation of metrics

CodeMetricGenerator::generateMetrics(TreeNode::findNode('@\Classes\CodeMetricDummy1
'));

    //Check that 22 metrics (7 class level + 5*3 method level) has been generated
    select count(Value) from tmp;
    this.assertEqual(22,tmp.Value,"Incorrect number of metrics generated");
}

```

Class: CodeMetricStatItemTest

```

class CodeMetricStatItemTest extends XUnitDevTest
{
}

void testNew()
{
    CodeMetricStatItem statItem;

    //Create new item
    statItem = new CodeMetricStatItem('group','prefix');

    //Check that the correct group and prefix are saved/returned correct
    this.assertEqual('group',statItem.getName(),"Group name not saved/fetched
correct");
    this.assertEqual('prefix',statItem.getPrefixName(),"Prefix name not
saved/fetched correct");
}

void testAddValue()
{
    //Create new item
    CodeMetricStatItem statItem = new CodeMetricStatItem('group','prefix');

    //Check that no values are added, and that the initialize values are correct
    this.assertEqual(0,statItem.getItemCount(),"Zero items should be added");
    this.assertEqual(0,statItem.getAvg(),"Average should be 0");
    this.assertEqual(0,statItem.getMax(),"Max value should be 0");
    this.assertNotEqual(0,statItem.getMin(),"Min value should not be 0");
    this.assertEqual(0,statItem.getSum(),"Sum should be 0");

    //Add the first value
    statItem.addValue(100);

    //Check that the correct values are computed
    this.assertEqual(1,statItem.getItemCount(),"One item should be added");
    this.assertEqual(100.00,statItem.getAvg(),"Average should be 100");
    this.assertEqual(100,statItem.getMax(),"Max value should be 100");
    this.assertEqual(100,statItem.getMin(),"Min value should be 100");
    this.assertEqual(100,statItem.getSum(),"Sum should be 100");

    //Add another value

```

```

statItem.addValue(200);

//Check again
this.assertEquals(2,statItem.getItemCount(),"Two items should be added");
this.assertEquals(150.00,statItem.getAvg(),"Average should be 150");
this.assertEquals(200,statItem.getMax(),"Max value should be 200");
this.assertEquals(100,statItem.getMin(),"Min value should be 100");
this.assertEquals(300,statItem.getSum(),"Sum should be 300");
}

```

Class: CodeMetricFITest

```

class CodeMetricFITest extends XUnitDevTest
{
    #SysBPCheck

    CodeMetricFI fiClass;
}

void testGetBPStr()
{
    CodeMetricFI fi;
    str bp;
    ;

    //FI for the class CodeMetricFI = 1 should not result in BP warning
    fi = new CodeMetricFI();
    fi.setElement(TreeNode::findNode('@'\Classes\CodeMetricFI'));
    bp = fi.getBPStr();
    this.assertEquals('',bp,"FI for CodeMetricFI should not result in BP warning");

    //FI for the class BOX = 411 should result in BP warning
    fi.setElement(TreeNode::findNode('@'\Classes\Box'));
    bp = fi.getBPStr();
    this.assertNotEqual('',bp,"FI for Box should result in BP warning");

}

void testGetDescription()
{
    CodeMetricFI fi = new CodeMetricFI();

    ;
    //Call instance method to get description
    this.assertEquals('FI', fi.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricFI fi = new CodeMetricFI();

    ;
    //Call instance method to get errorcode
    this.assertEquals(#BPErrorCodeMetricFI, fi.getErrorCode(), 'Wrong errorcode');
}

void testGetValue()

```

```

{
    CodeMetricFI fi;
    int val;
    ;

    //FI for CodeMetricFITest should be 0
    fi = new CodeMetricFI();
    fi.setElement(TreeNode::findNode('@'\Classes\CodeMetricFITest'));
    val = fi.getValue();
    this.assertEquals(0,val,"FI for CodeMetricFITest should be 0");

    //FI for CodeMetricFI should be 1, since CodeMetricFITest has a dependency on
it
    fi.setElement(TreeNode::findNode('@'\Classes\CodeMetricFI'));
    val = fi.getValue();
    this.assertEquals(1,val,"FI for CodeMetricFI should be 1");
}

```

Class: CodeMetricNOCTest

```

class CodeMetricNOCTest extends XUnitDevTest
{
    #SysBPCheck
}

void testGetBPStr()
{
    CodeMetricNOC noc;
    str bp;
    ;

    //NOC for the class CodeMetricBase = 2 should not result in BP warning
    noc = new CodeMetricNOC();
    noc.setElement(TreeNode::findNode('@'\Classes\CodeMetricBase'));
    bp = noc.getBPStr();
    this.assertEquals('',bp,"NOC for CodeMetricBase should not result in BP
warning");

    //NOC for the class AxInternalBase = 64 should result in BP warning
    noc.setElement(TreeNode::findNode('@'\Classes\AxInternalBase'));
    bp = noc.getBPStr();
    this.assertNotEqual('',bp,"NOC for AxInternalBase should result in BP
warning");
}

void testGetDescription()
{
    CodeMetricNOC noc = new CodeMetricNOC();

    ;
    //Call instance method to get description
    this.assertEquals('NOC', noc.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricNOC noc = new CodeMetricNOC();

```

```

    ;
    //Call instance method to get errorcode
    this.assertEquals(#BPErrorCodeMetricNOC, noc.getErrorCode(), 'Wrong
errorcode');
}

void testGetValue()
{
    CodeMetricNOC noc;
    int val;
    ;

    //NOC for \Classes\CodeMetricDummy4 should be 0
    noc = new CodeMetricNOC();
    noc.setElement(TreeNode::findNode('@\Classes\CodeMetricDummy4'));
    val = noc.getValue();
    this.assertEquals(0,val,"NOC for CodeMetricDummy4 should be 0");

    //NOC for \Classes\CodeMetricBase should be 2
    noc.setElement(TreeNode::findNode('@\Classes\CodeMetricBase'));
    val = noc.getValue();
    this.assertEquals(2,val,"NOC for CodeMetricBase should be 2");
}

```

Class: GraphEdgeTest

```

class GraphEdgeTest extends XUnitDevTest
{
}

void testSetNode1()
{
    GraphNode node1 = new GraphNode();
    GraphNode node2 = new GraphNode();
    GraphEdge edge = new GraphEdge();

    //Set the two nodes in the edge
    edge.setNode1(node1);
    edge.setNode2(node2);

    //Check that we can retrieve them again
    this.assertEquals(node1, edge.getNode1(), 'getNode1 did not return the correct
value');
    this.assertEquals(node2, edge.getNode2(), 'getNode2 did not return the correct
value');
}

```

Class: GraphUndirectedTest

```

class GraphUndirectedTest extends XUnitDevTest
{
}

void testAddEdge()
{
    GraphUndirected graph = new GraphUndirected();
    GraphNode node1;
    GraphNode node2;
}

```

```

GraphEdge edge;

//Add two nodes
node1 = graph.addNode('hello');
node2 = graph.addNode('world');

//Add edge between nodes
edge = graph.addEdge(node1, node2);

this.assertEquals(1, edge != null, 'new edge should be added');
this.assertEquals(node1, edge.getNode1(), 'node1 not added correct');
this.assertEquals(node2, edge.getNode2(), 'node2 not added correct');

//Try to add the same nodes as an edge again
edge = graph.addEdge(node1, node2);
this.assertEquals(1, edge == null, 'new edge should not be added');
}

void testAddNode()
{
    GraphUndirected graph = new GraphUndirected();
    GraphNode node;
    str testStr = 'hello world';

    //Test for empty when initialized
    this.assertEquals(0, graph.getNodes().elements(), 'graph should not contain any
elements');

    //Add an element and test that it was added
    node = graph.addNode(testStr);
    this.assertEquals(1, node != null, 'node should be added');
    this.assertEquals(1, graph.getNodes().elements(), 'graph should contain one
element');

    //Find the node containing 'hello world'
    node = graph.findNodeOnData(testStr);
    this.assertEquals(1, node != null, 'node should be found');
}

void testGetListOfNeighbours()
{
    GraphUndirected graph = new GraphUndirected();
    GraphNode node1;
    GraphNode node2;
    GraphNode node3;

    GraphEdge edge13;
    GraphEdge edge32;

    List neighbours;

    //Add nodes and edges
    node1 = graph.addNode('hello');
    node2 = graph.addNode('world');
    node3 = graph.addNode('today');
    edge13 = graph.addEdge(node1, node3);
    edge32 = graph.addEdge(node3, node2);

    //Get the neighbours of node1

```

```

    neighbours = graph.getListOfNeighbours(node1);
    this.assertEquals(1, neighbours.elements(), 'node1 should have 1 neighbour');

    //Get the neighbours of node3
    neighbours = graph.getListOfNeighbours(node3);
    this.assertEquals(2, neighbours.elements(), 'node3 should have 2 neighbours');

    //Get the neighbours of node2
    neighbours = graph.getListOfNeighbours(node2);
    this.assertEquals(1, neighbours.elements(), 'node2 should have 1 neighbour');

}

void testRunDFS()
{
    ListEnumerator enum;
    GraphNode node;
    int nodesWithoutParent;

    GraphUndirected graph = new GraphUndirected();

    /*The test graph consists of two disconnected parts:

        A - B - C
        | /
        E      D - F

    */

    GraphNode nodeA = graph.addNode('A');
    GraphNode nodeB = graph.addNode('B');
    GraphNode nodeC = graph.addNode('C');
    GraphNode nodeD = graph.addNode('D');
    GraphNode nodeE = graph.addNode('E');
    GraphNode nodeF = graph.addNode('F');

    //Add the edges
    graph.addEdge(nodeA, nodeE);
    graph.addEdge(nodeA, nodeB);
    graph.addEdge(nodeB, nodeC);
    graph.addEdge(nodeD, nodeF);

    //Before we run the DFS none of the nodes should have a parent
    this.assertEquals(6, graph.nodesWithoutParent(), 'nodesWithoutParent should
    equal the number of nodes before runDFS');

    //Do the DFS
    graph.runDFS();

    //Check that only two nodes does not have a parent
    this.assertEquals(2, graph.nodesWithoutParent(), 'nodesWithoutParent did not
    return the correct value, so runDFS must have failed');

}

```

Class: GraphNodeTest

```
class GraphNodeTest extends XUnitDevTest
{
}

void testSetAndGet()
{
    GraphNode gnode = new GraphNode();
    GraphNode gnodeParent = new GraphNode();

    //Add some data and retrieve it again
    gnode.setData(gnodeParent);
    this.assertEquals(gnodeParent, gnode.getData(), 'getData did not return the
correct value');

    //Add the color and retrieve it again
    gnode.setColor(2);
    this.assertEquals(2, gnode.getColor(), 'getColor did not return the correct
value');

    //Set timediscovered and retrieve it again
    gnode.setTimeDiscovered(3);
    this.assertEquals(3, gnode.getTimeDiscovered(), 'getTimeDiscovered did not
return the correct value');

    //Set time finished and retrieve it again
    gnode.setTimeFinished(4);
    this.assertEquals(4, gnode.getTimeFinished(), 'getTimeFinished did not return
the correct value');

    //Set parent and retrieve it again
    gnode.setParent(gnodeParent);
    this.assertEquals(gnodeParent, gnode.getParent(), 'getParent did not return the
correct value');
}
}
```

Class: CodeMetricLCOMTest

```
class CodeMetricLCOMTest extends XUnitDevTest
{
    #SysBPCheck
}

void testGetBPStr()
{
    CodeMetricLCOM lcom;
    str bp;
    ;

    //LCOM for the class CodeMetricDummy4 = 2 should result in BP warning
    lcom = new CodeMetricLCOM();
    lcom.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy4'));
    bp = lcom.getBPStr();
    this.assertNotEqual('', bp, "LCOM for CodeMetricDummy4 should result in BP
warning");

    //LCOM for the class SourceCodeChunker = 1 should not result in BP warning
}
```



```

    lcom = new CodeMetricLCOM();
    lcom.setElement(TreeNode::findNode(@"\Classes\SourceCodeChunker"));
    bp = lcom.getBPStr();
    this.assertEqual('',bp,"LCOM for SourceCodeChunker should not result in BP
warning");
}

void testGetDescription()
{
    CodeMetricLCOM lcom = new CodeMetricLCOM();

    ;
    //Call instance method to get description
    this.assertEqual('LCOM', lcom.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricLCOM lcom = new CodeMetricLCOM();

    ;
    //Call instance method to get errorcode
    this.assertEqual(#BPErrorCodeMetricLCOM, lcom.getErrorCode(), 'Wrong
errorcode');
}

void testGetValue()
{
    CodeMetricLCOM lcom;
    int val;
    ;

    /* LCOM for CodeMetricDummy4 = 2:
       a         b         c
       |         |         |
       f() - g() - h()   x()
       |_____|
    */

    lcom = new CodeMetricLCOM();
    lcom.setElement(TreeNode::findNode(@"\Classes\CodeMetricDummy4"));
    val = lcom.getValue();
    this.assertEqual(2,val,"LCOM for CodeMetricDummy4 should be 2");
}

```

Class: CodeMetricRFCTest

```

class CodeMetricRFCTest extends XUnitDevTest
{
    #SysBPCheck
}

void testGetBPStr()
{
    CodeMetricRFC rfc;
    str bp;
    ;

    //RFC for the class CodeMetricDummy3 = 6 should not result in BP warning
    rfc = new CodeMetricRFC();
}

```

```

    rfc.setElement(TreeNode::findNode(@"\Classes\CodeMetricDummy3"));
    bp = rfc.getBPStr();
    this.assertEqual('',bp,"RFC for CodeMetricDummy3 should not result in BP
warning");

    //RFC for the class SysStartupCmdCheckBestPractices should result in BP
warning
    rfc = new CodeMetricRFC();

rfc.setElement(TreeNode::findNode(@"\Classes\SysStartupCmdCheckBestPractices"));
    bp = rfc.getBPStr();
    this.assertNotEqual('',bp,"RFC for SysStartupCmdCheckBestPractices should
result in BP warning");
}

void testGetDescription()
{
    CodeMetricRFC rfc = new CodeMetricRFC();

    ;
    //Call instance method to get description
    this.assertEqual('RFC', rfc.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricRFC rfc = new CodeMetricRFC();

    ;
    //Call instance method to get errorcode
    this.assertEqual(#BPErrorCodeMetricRFC, rfc.getErrorCode(), 'Wrong
errorcode');
}

void testGetValue()
{
    CodeMetricRFC rfc;
    int val;
    ;

    /*RFC for the class CodeMetricDummy3 = 6:
    \Classes\CodeMetricDummy3\methodX

    \Classes\DictClass\new
    \Classes\ClassInstanciator\createSubClassInstances
    \Classes\List\elements
    \Classes\List\addEnd
    \Classes\CodeMethodMetric\new
    */

    rfc = new CodeMetricRFC();
    rfc.setElement(TreeNode::findNode(@"\Classes\CodeMetricDummy3"));
    val = rfc.getValue();
    this.assertEqual(6,val,"RFC for CodeMetricDummy3 should be 6");
}

```

Class: CodeMetricCBOTest

```

class CodeMetricCBOTest extends XUnitDevTest
{
    #SysBPCheck

```

```

}

void testGetBPStr()
{
    CodeMetricCBO cbo;
    str bp;
    ;

    //CBO for the class CodeMetricDummy3 = 9 should not result in BP warning
    cbo = new CodeMetricCBO();
    cbo.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy3'));
    bp = cbo.getBPStr();
    this.assertEquals('',bp,"CBO for CodeMetricDummy3 should not result in BP
warning");

    //CBO for the class SysStartupCmdCheckBestPractices should result in BP
warning
    cbo = new CodeMetricCBO();

    cbo.setElement(TreeNode::findNode('@'\Classes\SysStartupCmdCheckBestPractices'));
    bp = cbo.getBPStr();
    this.assertNotEqual('',bp,"CBO for SysStartupCmdCheckBestPractices should
result in BP warning");

}

void testGetDescription()
{
    CodeMetricCBO cbo = new CodeMetricCBO();

    ;
    //Call instance method to get description
    this.assertEquals('CBO', cbo.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricCBO cbo = new CodeMetricCBO();

    ;
    //Call instance method to get errorcode
    this.assertEquals(#BPErrorCodeMetricCBO, cbo.getErrorCode(), 'Wrong
errorcode');
}

void testGetValue()
{
    CodeMetricCBO cbo;
    int val;
    ;

    /*CBO for the class CodeMetricDummy3 = 10:
    \Classes\Address
    \Classes\AddressWizard
    \Classes\ClassInstanciator
    \Classes\CodeMethodMetric
    \Classes\CodeMetricDummy2
    \Classes\StringUtil
    \Data Dictionary\Tables\Address
    \Data Dictionary\Tables\CustTable

```

```

        \System Documentation\Classes\DictClass
        \System Documentation\Classes\List
    */

    cbo = new CodeMetricCBO();
    cbo.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy3'));
    val = cbo.getValue();
    this.assertEquals(10,val,"CBO for CodeMetricDummy3 should be 10");
}

```

Class: CodeMetricWMCTest

```

class CodeMetricWMCTest extends XUnitDevTest
{
    #SysBPCheck
}

void testGetBPStr()
{
    CodeMetricWMC wmc;
    str bp;
    ;

    //WMC for the class CodeMetricDummy1 = 33 should not result in BP warning
    wmc = new CodeMetricWMC();
    wmc.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy1'));
    bp = wmc.getBPStr();
    this.assertEquals('',bp,"WMC for CodeMetricDummy1 should not result in BP
warning");

    //WMC for the class SysStartupCmdCheckBestPractices should result in BP
warning
    wmc = new CodeMetricWMC();

    wmc.setElement(TreeNode::findNode('@'\Classes\SysStartupCmdCheckBestPractices'));
    bp = wmc.getBPStr();
    this.assertNotEqual('',bp,"WMC for SysStartupCmdCheckBestPractices should
result in BP warning");
}

void testGetDescription()
{
    CodeMetricWMC wmc = new CodeMetricWMC();

    ;
    //Call instance method to get description
    this.assertEquals('WMC', wmc.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricWMC wmc = new CodeMetricWMC();

    ;
    //Call instance method to get errorcode
    this.assertEquals(#BPErrorCodeMetricWMC, wmc.getErrorCode(), 'Wrong
errorcode');
}

```

```

void testGetValue()
{
    CodeMetricWMC wmc;
    int val;
    ;

    //WMC for the class CodeMetricDummy1 = 9 + 5 + 2 + 16 + 1 = 33
    wmc = new CodeMetricWMC();
    wmc.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy1'));
    val = wmc.getValue();

    //Test that we get the correct value
    this.assertEquals(33,val,"WMC for CodeMetricDummy1 should be 33");
}

```

Class: CodeMetricDITTest

```

class CodeMetricDITTest extends XUnitDevTest
{
    #SysBPCheck
}

void testGetBPStr()
{
    CodeMetricDIT dit;
    str val;
    ;

    //DIT.GetBpStr for the class CodeMetricDIT = ''
    dit = new CodeMetricDIT();
    dit.setElement(TreeNode::findNode('@'\Classes\CodeMetricDIT'));
    val = dit.getBPStr();
    this.assertEquals('',val,'getBPStr for CodeMetricDIT should be blank');

    //DIT.GetBpStr for the class CodeMetricDummy2 != ''
    dit.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy2'));
    val = dit.getBPStr();
    this.assertNotEqual('',val,'getBPStr for CodeMetricDummy2 should not be
blank');
}

void testGetDescription()
{
    CodeMetricDIT dit = new CodeMetricDIT();
    ;

    //Call instance method to get description
    this.assertEquals('DIT', dit.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricDIT dit = new CodeMetricDIT();
    ;
    //Call instance method to get errorcode
    this.assertEquals(#BPErrorCodeMetricDIT, dit.getErrorCode(), 'Wrong
errorcode');
}

```

```

}

void testGetValue()
{
    CodeMetricDIT dit;
    int val;
    ;

    //DIT for the class CodeMetricDummy1 = 1
    dit = new CodeMetricDIT();
    dit.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy1'));
    val = dit.getValue();
    this.assertEquals(1,val,"getValue for CodeMetricDummy1 should be 1");

    //DIT for the class CodeMetricDIT = 3
    dit = new CodeMetricDIT();
    dit.setElement(TreeNode::findNode('@'\Classes\CodeMetricDIT'));
    val = dit.getValue();
    this.assertEquals(3,val,"getValue for CodeMetricDIT should be 3");

    //DIT for the class CodeMetricDummy2 = 9
    dit = new CodeMetricDIT();
    dit.setElement(TreeNode::findNode('@'\Classes\CodeMetricDummy2'));
    val = dit.getValue();
    this.assertEquals(9,val,"getValue for CodeMetricDummy2 should be 9");
}

```

Class: ClassInstanciatorTest

```

class ClassInstanciatorTest extends xUnitDevTest
{
}

void testCreateSubClassInstances()
{
    List list;
    ;

    //Check that it will return an empty list if no subclasses exists
    list =
ClassInstanciator::createSubClassInstances(classnum(ClassInstanciatorTest));
    this.assertEquals(0, list.elements(), 'List should be empty');

    //Check that the list is not empty, when called with the CodeMethodMetric id
    list = ClassInstanciator::createSubClassInstances(classnum(CodeMethodMetric));
    this.assertNotEqual(0, list.elements(), 'List should contain elements');
}

```

Class: StringUtilTest

```

class StringUtilTest extends xUnitDevTest
{
}

void testCountOccurences()
{

```

```

str orgtext = '\n\n\n \n ';
;

//Test of finding single character
this.assertEquals(4, StringUtil::CountOccurrences(orgtext, '\n'),
'StringUtil::CountOccurrences failed on finding single character');

//Test of finding multiple character sequence
this.assertEquals(2, StringUtil::CountOccurrences(orgtext, '\n\n'),
'StringUtil::CountOccurrences failed on finding multiple character sequence');

//Test of finding character sequence that does not exist
this.assertEquals(0, StringUtil::CountOccurrences(orgtext, 'hello'),
'StringUtil::CountOccurrences failed on finding character sequence that does not exist');
}

```

Class: SourceCodeChunkerTest

```

class SourceCodeChunkerTest extends xUnitDevTest
{
    str orgCode;
}

public void setUp()
{
    super();

    /*Create dummy code for test
    Is in setup since it is shared by various tests
    */
    orgCode = '/*Starting comment\n'
+ '  Comment line 2*/\n'
+ 'int MyMethod()\n'
+ '{\n'
+ '    int a;\n'
+ '    int b; //Comment here\n'
+ '    str s=/* hello */ \\\\ \\\ // \\\\' \';\n'
+ '    /*comment*/ int c; //Line ends with comment\n'
+ '\n'
+ '    s=@\'hello \\\\' ;\n'
+ '    ;\n'
+ '    if (a==b)\n'
+ '        this.doSomething();\n'
+ '\n'
+ '    //Only comment line\n'
+ '}'\n';

}

void testGetNext()
{
    str newCode;
    SourceCodeChunker chunker = new SourceCodeChunker(orgCode); //Use the code from
variable orgCode
    ;

    //Get the first chunk

```

```

    this.assertEquals(true, chunker.moveToNext(), 'SourceCodechunker.moveToNext failed
on call 1');
    this.assertEquals('', chunker.currentCodeChunk(),
'SourceCodechunker.currentCodeChunk failed on call 1');
    this.assertEquals('/*Starting comment\n Comment line 2*/',
chunker.currentCommentChunk(), 'SourceCodechunker.currentCommentChunk failed on
call 1');
    this.assertEquals(1, chunker.codeStartLine(), 'SourceCodechunker.codeStartLine
failed on call 1');
    this.assertEquals(1, chunker.commentStartLine(),
'SourceCodechunker.commentStartLine failed on call 1');

    //Get and test subsequent chunks
    this.assertEquals(true, chunker.moveToNext(), 'SourceCodechunker.moveToNext failed
on call 2');
    this.assertEquals('\nint MyMethod()\n{\n    int a;\n    int b; ',
chunker.currentCodeChunk(), 'SourceCodechunker.currentCodeChunk failed on call 2');
    this.assertEquals('//Comment here', chunker.currentCommentChunk(),
'SourceCodechunker.currentCommentChunk failed on call 2');
    this.assertEquals(2, chunker.codeStartLine(), 'SourceCodechunker.codeStartLine
failed on call 2');
    this.assertEquals(6, chunker.commentStartLine(),
'SourceCodechunker.commentStartLine failed on call 2');

    this.assertEquals(true, chunker.moveToNext(), 'SourceCodechunker.moveToNext failed
on call 3');
    this.assertEquals('\n    str s=\'/* hello */ \\\\' // \\\\' \';\n    ',
chunker.currentCodeChunk(), 'SourceCodechunker.currentCodeChunk failed on call 3');
    this.assertEquals('/*comment*/', chunker.currentCommentChunk(),
'SourceCodechunker.currentCommentChunk failed on call 3');
    this.assertEquals(6, chunker.codeStartLine(), 'SourceCodechunker.codeStartLine
failed on call 3');
    this.assertEquals(8, chunker.commentStartLine(),
'SourceCodechunker.commentStartLine failed on call 3');

    this.assertEquals(true, chunker.moveToNext(), 'SourceCodechunker.moveToNext failed
on call 4');
    this.assertEquals(' int c; ', chunker.currentCodeChunk(),
'SourceCodechunker.currentCodeChunk failed on call 4');
    this.assertEquals('//Line ends with comment', chunker.currentCommentChunk(),
'SourceCodechunker.currentCommentChunk failed on call 4');
    this.assertEquals(8, chunker.codeStartLine(), 'SourceCodechunker.codeStartLine
failed on call 4');
    this.assertEquals(8, chunker.commentStartLine(),
'SourceCodechunker.commentStartLine failed on call 4');

    this.assertEquals(true, chunker.moveToNext(), 'SourceCodechunker.moveToNext failed
on call 5');
    this.assertEquals('\n\n    s=@\'hello \\\\';\n    ;\n    if (a==b)\n
this.doSomething();\n\n    ', chunker.currentCodeChunk(),
'SourceCodechunker.currentCodeChunk failed on call 5');
    this.assertEquals('//Only comment line', chunker.currentCommentChunk(),
'SourceCodechunker.currentCommentChunk failed on call 5');
    this.assertEquals(8, chunker.codeStartLine(), 'SourceCodechunker.codeStartLine
failed on call 5');
    this.assertEquals(15, chunker.commentStartLine(),
'SourceCodechunker.commentStartLine failed on call 5');

    this.assertEquals(true, chunker.moveToNext(), 'SourceCodechunker.moveToNext failed
on call 6');

```



```

    this.assertEquals('\n}\n', chunker.currentCodeChunk(),
'SourceCodechunker.currentCodeChunk failed on call 6');
    this.assertEquals('', chunker.currentCommentChunk(),
'SourceCodechunker.currentCommentChunk failed on call 6');
    this.assertEquals(15, chunker.codeStartLine(), 'SourceCodechunker.codeStartLine
failed on call 6');
    this.assertEquals(17, chunker.commentStartLine(),
'SourceCodechunker.commentStartLine failed on call 6');

    //We should now have reached the end of the source code, so any subsequent
calls to getNext should return false!
    this.assertEquals(false, chunker.moveToNext(), 'SourceCodechunker.moveToNext failed
on call 7');
}

```

Class: CodeMetricCPMethodTest

```

class CodeMetricCPMethodTest extends xUnitDevTest
{
    #SysBPCheck
}

void testCalcCP()
{
    TreeNode node;
    int value;
    ;

    /* Calculation for: \Classes\CodeMetricDummy1\method1

        Lines with comments = 16
        Total lines = 34
        Blank lines = 6

        CP=(16/(34-6))*100 = 57%
    */
    node = TreeNode::findNode(@'\Classes\CodeMetricDummy1\method1');
    value = CodeMetricCPMethod::calcCP(node.AOTgetSource());
    this.assertEquals(57, value, 'CP of CodeMetricDummy1.method1 not correct');

}

void testGetBPStr()
{
    CodeMetricCPMethod cp;
    str val;
    ;

    //No best practice message should occur
    cp = new CodeMetricCPMethod();
    cp.setElement(TreeNode::findNode(@'\Classes\CodeMetricDummy1\method1'));
    val = cp.getBPStr();
    this.assertEquals(val, "", "getBPStr for CodeMetricDummy1\method1 should be
blank");

    //A best practice warning should occur
    cp.setElement(TreeNode::findNode(@'\Classes\CodeMetricDummy1\noComments'));
}

```

```

        val = cp.getBPStr();
        this.assertNotEqual(val, "", @"getBPStr for CodeMetricDummy1\noComments should
result in a BP warning");
    }

void testGetDescription()
{
    CodeMetricCPMethod cp = new CodeMetricCPMethod();

    ;
    //Call instance method to get description
    this.assertEqual('CP', cp.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricCPMethod cp = new CodeMetricCPMethod();
    ;
    //Call static method to get errorcode
    this.assertEqual(#BPErrorCodeMetricCPMethod, cp.getErrorCode(), 'Wrong
errorcode');
}

```

Class: CodeMetricVGMethodTest

```

class CodeMetricVGMethodTest extends XUnitDevTest
{
    #SysBPCheck
}

void testCalcVG()
{
    TreeNode node;
    SysScannerClass scanner;
    int value;
    ;

    //V(G) with two embedded methods, and all other code-constructs (besides SQL)
that will add to the CC
    node = TreeNode::findNode(@"\Classes\CodeMetricDummy1\method2");
    scanner = new SysScannerClass(node);
    value = CodeMetricVGMethod::calcVG(scanner);
    this.assertEqual(16, value, 'V(G) of CodeMetricDummy1.method2 not correct');

    //V(G) with just one embedded method.
    node = TreeNode::findNode(@"\Classes\CodeMetricDummy1\method1");
    scanner = new SysScannerClass(node);
    value = CodeMetricVGMethod::calcVG(scanner);
    this.assertEqual(2, value, 'V(G) of CodeMetricDummy1.method1 not correct');

    //Test of V(G) in SQL
    node = TreeNode::findNode(@"\Classes\CodeMetricDummy1\if");
    scanner = new SysScannerClass(node);
    value = CodeMetricVGMethod::calcVG(scanner);
    this.assertEqual(5, value, 'V(G) of CodeMetricDummy1.if not correct');

    //Test of method definitions
    node = TreeNode::findNode(@"\Classes\CodeMetricDummy1\abc");
    scanner = new SysScannerClass(node);
    value = CodeMetricVGMethod::calcVG(scanner);
}

```

```

        this.assertEquals(9, value, 'V(G) of CodeMetricDummy1.abc not correct');
    }

void testGetBPStr()
{
    CodeMetricVGMethod vg;
    str val;
    ;

    //No best practice message should occur, since the CC value=2
    vg = new CodeMetricVGMethod();
    vg.setElement(TreeNode::findNode(@"\Classes\CodeMetricDummy1\method1"));
    val = vg.getBPStr();
    this.assertEquals(val, "", "getBPStr for CodeMetricDummy1\method1 should be
blank");

    //A best practice message should occur, since the CC value=16
    vg.setElement(TreeNode::findNode(@"\Classes\CodeMetricDummy1\method2"));
    val = vg.getBPStr();
    this.assertNotEqual(val, "", "getBPStr for CodeMetricDummy1\method2 should not be
blank");
}

void testGetDescription()
{
    CodeMetricVGMethod vg = new CodeMetricVGMethod();

    //Call instance method to get description
    this.assertEquals('V(G)', vg.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricVGMethod vg = new CodeMetricVGMethod();
    ;
    //Call instance method to get errorcode
    this.assertEquals(#BPErrorCodeMetricVGMethod, vg.getErrorCode(), 'Wrong
errorcode');
}

```

Class: CodeMetricSLOCMethodTest

```

class CodeMetricSLOCMethodTest extends XUnitDevTest
{
    #SysBPCheck
}

public void testCalcSLOC()
{
    TreeNode node;
    int value;
    ;

    //Load treenode and call static method to calculate SLOC
    node = TreeNode::findNode(@"\Classes\CodeMetricDummy1\method1");
    value = CodeMetricSLOCMethod::calcSLOC(node.AOTgetSource());

    this.assertEquals(15, value, 'SLOC of CodeMetricDummy1.method1 not correct');
}

```

```

void testGetBPStr()
{
    CodeMetricSLOCMethod sloc;
    str val;
    ;

    //No best practice message should occur
    sloc = new CodeMetricSLOCMethod();
    sloc.setElement(TreeNode::findNode(@"\Classes\CodeMetricDummy1\method1"));
    val = sloc.getBPStr();
    this.assertEqual(val, "", "getBPStr for CodeMetricDummy1\method1 should be
blank");

    //A best practice warning should occur

sloc.setElement(TreeNode::findNode(@"\Classes\SysStartupCmdCheckBestPractices\update
ExcelWorkbook"));
    val = sloc.getBPStr();
    this.assertNotEqual(val, "", @"getBPStr for
SysStartupCmdCheckBestPractices\updateExcelWorkbook should result in a BP
warning");

}

void testGetDescription()
{
    CodeMetricSLOCMethod sloc = new CodeMetricSLOCMethod();

    ;
    //Call instance method to get description
    this.assertEqual('SLOC', sloc.getDescription(), 'Wrong description');
}

void testGetErrorCode()
{
    CodeMetricSLOCMethod sloc = new CodeMetricSLOCMethod();
    ;
    //Call instance method to get errorcode
    this.assertEqual(#BPErrorCodeMetricSLOCMethod, sloc.getErrorCode(), 'Wrong
errorcode');
}

public void testRemoveComments()
{
    str orgCode;
    str expectedNewCode;
    str newCode;
    ;

    //Check of code with comments
    orgCode = "/* hello */\nprivate int something{\n int x; //comment\n /*1\n
2*/\n\n /* 123 // */x=4; /* 123 */\n}";
    expectedNewCode = "\nprivate int something{\n int x; \n \n\n x=4;\n}";
    newCode = CodeMetricSLOCMethod::removeComments(orgCode);
    this.assertEqual(expectedNewCode, newCode, 'Removal of comments failed');

    //Check of code without any comments
    orgCode = "\nprivate int something{\n int x; \n \n\n x=4;\n}";
}

```

```

        expectedNewCode = orgCode;
        newCode = CodeMetricSLOCMethod::removeComments(orgCode);
        this.assertEquals(expectedNewCode, newCode, 'Code contains no comments but has
changed anyway!');

        //Check of code with strings containing escaped comment chars
        orgCode = 'int MethodA{ \n  str a; \n \n a = \'\'\''; a = \'*/\'; a = \'\\\\\\\\\'
};
        expectedNewCode = orgCode;
        newCode = CodeMetricSLOCMethod::removeComments(orgCode);
        this.assertEquals(expectedNewCode, newCode, 'Code containing escaped comment
chars!');

        //Check where code end with sigle comment and no newlines
        orgcode = '// Start comment \n'
            + 'void method1() \n'
            + '{\n'
            + '}\n'
            + '// End comment';
        expectedNewCode = '\n'
            + 'void method1() \n'
            + '{\n'
            + '}\n';
        newCode = CodeMetricSLOCMethod::removeComments(orgCode);
        this.assertEquals(expectedNewCode, newCode, 'Code ending with single comment
and no newlins');
    }
}

```

Class: SysBPCheckMemberFunctionTest

```

class SysBPCheckMemberFunctionTest extends XUnitDevTest
{
}

void testCheckComplexity()
{
    #SysBPCheck

    TreeNode testNode =
TreeNode::findNode('@\Classes\SysStartupCmdCheckBestPractices\updateExcelWorkbook')
;

    SysCompilerOutput output;
    TmpCompilerOutput tmpout;

    int bpcount;
    ;

    //Start by enabling the complexity check
    SysBPCheckComplexityEnabler::setBPComplexity(true);

    //Clear the output
    infolog.clear(0);

    //Do the check
    SysBPCheck::checkTreeNode(testNode);

    //Get output
    output = infolog.compilerOutput();
}

```

```

tmpout = output.compilerOutput();

while select tmpout
    where tmpout.SysCompilerOutputTab == SysCompilerOutPutTab::BestPractices
        && ( tmpout.CompileErrorCode == #BPErrorCodeMetricSLOCMMethod
            || tmpout.CompileErrorCode == #BPErrorCodeMetricVGMMethod
            || tmpout.CompileErrorCode == #BPErrorCodeMetricCPMethod)
    {
        bpcount++;
    }

    //Check that three BP deviations occurred
    this.assertEqual(3,bpcount,@'3 BP complexity deviations should occur for
method \Classes\SysStartupCmdCheckBestPractices\updateExcelWorkbook');
}

```

Class: SysBPCheckClassNodeTest

```

class SysBPCheckClassNodeTest extends XUnitDevTest
{
}

void testCheckComplexity()
{
    #SysBPCheck

    TreeNode testNode =
TreeNode::findNode(@'\Classes\SysStartupCmdCheckBestPractices');
    SysCompilerOutput output;
    TmpCompilerOutput tmpout;

    int bpcount;
    ;

    //Start by enabling the complexity check
    SysBPCheckComplexityEnabler::setBPComplexity(true);

    //Clear the output
    infolog.clear(0);

    //Do the check
    SysBPCheck::checkTreeNode(testNode);

    //Get output
    output = infolog.compilerOutput();
    tmpout = output.compilerOutput();

    while select tmpout
        where tmpout.SysCompilerOutputTab == SysCompilerOutPutTab::BestPractices
            && tmpout.CompileErrorCode >= #BPErrorCodeMetricDIT
            && tmpout.CompileErrorCode <= #BPErrorCodeMetricFI
        {
            bpcount++;
        }

    //Check that only BP deviations occurred
    this.assertEqual(3,bpcount,@'3 BP complexity deviations should occur for class
\Classes\SysStartupCmdCheckBestPractices (WMC, CBO, RFC)');
}

```

}

Test classes

Class: CodeMetricDummy1

```
class CodeMetricDummy1 extends object
{
}

void abc()
{
    //Each of these embedded methods adds one to CC
    int a() { ;return 1;}
    int64 b() { ;return 2;}
    boolean c() { ;return true;}
    real d() { ;return 3.0;}
    date e() { ;return today();}
    timeofday f() { ;return timenow();}
    str g() { ;return 'hello';}
    guid h() { ;return str2guid('hello');}
    ;

    //These should not add anything to CC
    startLengthyOperation();
    endLengthyOperation(true);

    //Here CC=9
}

void if(str someval)
{
    CustTable cust;
    ContactPerson contact;
    ;

    //Adds 0 to CC
    select cust where Cust.AccountNum == "4000";

    //Adds 0 to CC
    select cust where Cust.AccountNum == "4000" && cust.Name == "Hello" ||
cust.BankAccount == "123456";

    //Adds 1 to CC
    while select cust where Cust.AccountNum == "4000" && cust.Name == "Hello" ||
cust.BankAccount == "123456"
    {
        print cust.NameAlias, " ", cust.Phone, '\n';
    }

    //Adds 1 to CC
    select cust where cust.AccountNum == "4000"
    join contact where contact.Address == cust.Address;

    //Adds 2 to CC
    while select cust where Cust.AccountNum == "4000" && cust.Name == "Hello"
    join contact where contact.Address == cust.Address && contact.AssistantName ==
"World"
    {

```



```

        print cust.NameAlias, " ", cust.Phone, '\n';
    }

    //Adds 0 to CC
    delete_from cust where cust.AccountNum == "4000" && cust.Name == "HelloWorld";

    //Adds 0 to CC
    update_recordset cust setting Name = 'NewName' where cust.AccountNum == "4000"
    && cust.Name == "HelloWorld";

    //Adds 0 to CC
    insert_recordset cust (Name, Address) select Name, Address from contact;

    //Here CC=5
}

/*Comment before method name
  Comment line 2* //
*/
public void method1()
{
    str thisname; //in-line comment followed by 3 blanks
    int x;

    //Comment before method in method
    int plus(int a, int b)
    {
        /*Comment indside method in method

        */
        int z;
        z = b+a;

        return z;
    }
    //Comment right after method in method

    ;
    thisname = methodStr(CodeMetricDummy1, method1);
    /* comment before code*/x = plus(plus(1,2),3);/*comment after code*/

    Box::info('x in the method ' + thisname + ' equals ' + int2str(x)); /*
multiline comment start here

    and ends
    }
    // after here
    */

}

public int method2()
{
    //CC=1
    int a=4;
    int i;

    //This adds 2 to CC
    int embedMethod(int x)
    {

```

```

    if (x==1)
        return x*x;
    else
        return x;
}

//This adds 1 to CC
SysBPCheckBase getBase()
{
    return SysBPCheckBase::construct();
}
;

//This adds nothing to CC
this.if('hello world');
i = embedMethod(a);
getBase();

//This adds 1 to CC
for (i=0;i<1;i++)
{
    a++;
}

//This adds 2 to CC
while(a==2 || a==3)
{
    a--;
}

//This adds 1 to CC
do
{
    a++;
}while(a==0);

//This adds 4 to CC
switch(a)
{
    case 1:
        a=1;
    case 2:
        {
            a=2;
        }
    case 3:
    case 4:
        a=2+3;
    default:
        a=99;
}

//The try/catch adds 1 to CC
try
{
    //This adds 3 to CC
    if(a==1 && a==2 || a==3)
        a=0;
    else
        a=1;
}

```

```

    catch(Exception::Error)
    {
        print 'error';
    }

    //Here CC=16

    return a;
}

int noComments(int x)
{
    ;

    return x+x;
}

```

Class: CodeMetricDummy2

```

class CodeMetricDummy2 extends ProdJournalCheckPostRouteJob
{
}

```

Class: CodeMetricDummy3

```

class CodeMetricDummy3
{
    CodeMetricDummy2 d2;
    Address add;

    #SysBPCheck
}

public AddressWizard methodX(int a, StringUtil b)
{
    DictClass dict;
    CustTable cust;
    List list;
    int xx;
    ;

    dict = new DictClass(classNum(CodeMetricDummy2));

    select cust where Cust.AccountNum == "4000";

    list = ClassInstanciator::createSubClassInstances(classNum(CodeMethodMetric));
    if (list.elements() == 0)
        list.addEnd(new CodeMetricSLOCMethod());

    return null;
}

```

Class: CodeMetricDummy4

```

class CodeMetricDummy4
{
    int a,b,c;
}

```

```

void f()
{
    a = a + 1;
    this.g();
}

void g()
{
    int localvar;
    localvar = 2;
}

void h()
{
    b = b + 2;
    this.f();
    this.g();
}

void x()
{
    c = c + 4;
}

```

Unit test helper classes

Class: SysBPCheckComplexityEnabler

```

class SysBPCheckComplexityEnabler
{
}

static void setBPComplexity(boolean complexity_enabled=false)
{
    SysBPPParameters parameters;
;
    ttsbegin;
    parameters = SysBPPParameters::find(curuserid(), true);

    //Enable all best practice checks
    parameters.initValue();

    //Enable/disable the complexity check
    parameters.CheckComplexity = complexity_enabled;

    //Report all
    parameters.WarningLevel = SysBPWarningLevel::All;

    //Save the new settings
    parameters.update();

    //Enable best practice check in compiler
    xUserInfo::compilerWarningLevel(CompilerWarningLevel::Level4);

    ttscommit;

    //Let the compiler output get the new parameters
    SysCompilerOutput::updateParm();
}

```


Appendix C: Setup instructions

The following document gives step-by-step instructions of how to install the BP complexity tool.

Prerequisites

- Dynamics AX 4.0 Client must be installed on the machine and be connected to an AX Object Server.
- Developer license must be installed
- The SysTest (previously named XUnit) framework must be imported
- Make sure that any source control in Dynamics AX has been disabled

Import XPO file




1. Open the Dynamics AX Client.
2. Open the Application Object Tree (AOT), by pressing <CTRL+D>.
3. Click the Import button  in the AOT.
4. In the Import form, enter the path and filename of the complexity xpo file (e.g. PrivateProject_Complexity final version.xpo).
5. Press the Import button to start the import.
6. The result of the import can be seen in the Infolog.

Enable complexity check

1. Select the menu item Tools -> Options ...
2. In the Options form, press the Best Practices button
3. In the Best Practice parameters form, set the warning level to "All"
4. In the treeview on the Best Practice parameters form, check the node Best Practice checks -> Specific checks -> Classes -> Complexity
5. Press the OK button to save the parameters.
6. Restart the Dynamics AX Client for the changes to take effect

Appendix D: CD

Contents of CD

-  Appendix B – SourceCode.doc
-  MBS Functional Specification - Complexity.doc
-  PrivateProject_Complexity final version.xpo



CD