CSE3403 Fall 2008

Project #1:

Due: November 19,24,26, 2008

Weight 30%

In this project you are invited to evaluate and compare the complexity of the .NET platform Library vs. the complexity of the Java platform Library.

Background

In general, when developing software (s/w), a main issue is to measure the complexity of the resulting s/w product. This is useful especially during the maintenance part of the s/w lifecycle. The maintenance part of the s/w lifecycle is the most long-lived part of every s/w product. A major s/w product can take a year or two (or more) to build, but it is expected to be used for many years (in some cases, decades). During these many years the s/w usually undergoes changes, enhancements, corrections, etc. This is what we call "the maintenance" phase of the s/w lifecycle. Obviously, if the s/w is initially easy to understand, it will be easier to maintain it during the maintenance phase. Therefore, the notion of *complexity* of s/w has been established. In short, *s/w complexity* is the degree of how difficult a program is to comprehend and work with[1]. Over the years, computer scientists have developed many ways to measure (and predict) s/w complexity. These ways are called *metrics*. A metric is basically a number that indicates how complex a s/w artifact may be. For example, if s/w artifacts S1 and S2 have metrics m1 and m2 and m1 > m2 then S1 is more complex than S2. Similar to program complexity, people have devised metrics for structure complexity of XML schemas (reference [1]) and UML models (reference [2]). This type of metrics is typically referred to as *structure metrics*. For this project you will devise *Library complexity metrics*. Similar to s/w program metrics, we can define a *Library complexity metric as the degree of how difficult a Library is to comprehend and work with*.

[1] Note, this notion of complexity is different from the one typically used in algorithms. In algorithms, the term complexity refers to how fast an algorithm is (time complexity), or how much space it takes to execute (space complexity).

Potential benefits that you will have by doing this project.

1. Become familiar with the libraries of the .NET and Java platforms.
2. Increase your understanding and appreciation of the complexity of a platform's library.
3. Become familiar with the issues and difficulties involved in evaluating s/w and specifically platform-grade s/w products.
4. (May be) discover something "surprising" – e.g., that one library is much more complex than the other library.

Discussion.

In devising your collection of metrics, you should become familiar with relevant literature of similar works. The following four references are a good starting point.

[1] Harrison, Warren, "An Entropy-Based Measure of Software Complexity", *IEEE Transactions on Software Engineering*, vol. 18, no. 11, 1992, pp. 1025-1029.

[2] John Stephen Davis and Richard J. LeBlanc, "A study of the Applicability of Complexity Measures", *IEEE Transactions on Software Engineering*, vol. 14, no. 9, 1988, pp. 1366-1372.

[3] Joost Visser, "Structure Metrics for XML Schema", available at www.di.uminho.pt/~joostvisser/publications/StructureMetricsForXMLSchema.pdf

[4] Hyoseob Kim1 and Cornelia Boldyreff, "Developing Software Metrics Applicable to UML Models", available at http://alarcos.inf-cr.uclm.es/qaoose2002/docs/QAOOSE-Kim-Bol.pdf

You can obtain references [1] and [2] from the York U. libraries (e-resources). You can obtain references [3] and [4] from the URLs listed with the references.

Several metrics are described in the above references. Especially relevant to this project are probably the metrics mentioned in [3] and [4]. Note, not all metrics described in the

references may be applicable to this project. It is your task (as part of this project) to read the references (as well as other articles as you become more familiar with the topic) and then decide what to use and not use from what you read.

One of the metrics mentioned in [1], [2], and [3] is an *Entropy*-based metric. As you will read, the use of this metric for s/w vs. XML schema is somewhat different, but the idea is the same. You should adapt and use the Entropy metric in this project (you should also devise more metrics, but the use the Entropy metric is a minimum requirement).

Below, is an idea (to get you started) of how this metric can possibly be adapted and used for comparing the .NET and Java libraries.

Both the .NET and Java libraries have the following structure.

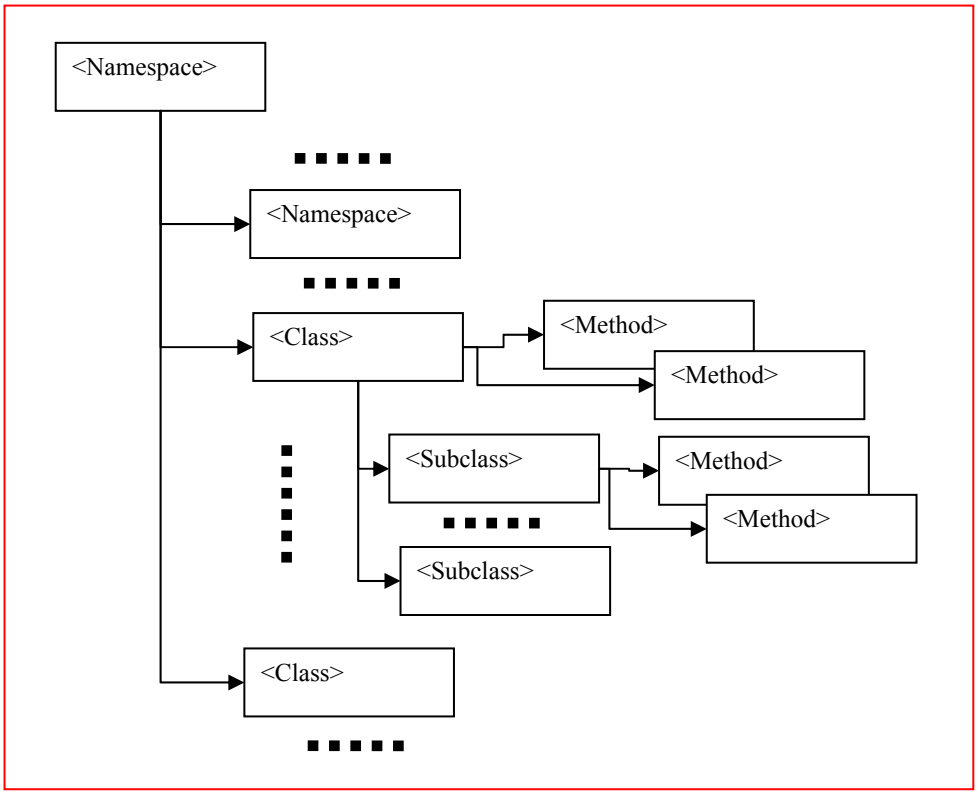

Figure 1: General Structure of .NET (and Java) library (in Java, the term "Package" is used for "Namespace" of .NET).

For using the entropy metric to judge the complexity of .NET from a graph like the one shown in Figure 1, you calculate

$$H = -\sum_{i=1}^{N} P_i \cdot \lg P_i \qquad (1)$$

where

$$P_i = P(C_i) = \frac{\text{number of nodes of class } C_i}{\text{total number of nodes in the graph}},$$

N = total number of classes C_i, i.e., total number of P_i s.

C_i: Equivalency class. This is the set of all nodes such that any two nodes N_i and N_j of the above graph have indegree(N_i) = indegree(N_j) *and* outdegree(N_i) = outdegree(N_j). (indegree(N_i) = the number of edges coming into node N_i in the graph; outdegree(N_i) = the number of edges going out of node N_i in the graph).

Lg is log base 2, in expression (1).

Example: if a class C3 has 4 elements (4 nodes) and there are 10 nodes in the graph, then P(C3) = 4/10.

In order to calculate the Entropy H of the .NET library, you should calculate the equivalency classes of the directed graph corresponding to the .NET library and then use the above expression (1). (The same applies for the Java library). If you want to calculate the H of only a *part* of .NET, then you should do similar for the graph corresponding to that part, and so on.

The example below illustrates how to calculate H for a general directed graph.

Example:
Consider the directed graph shown in figure 2.

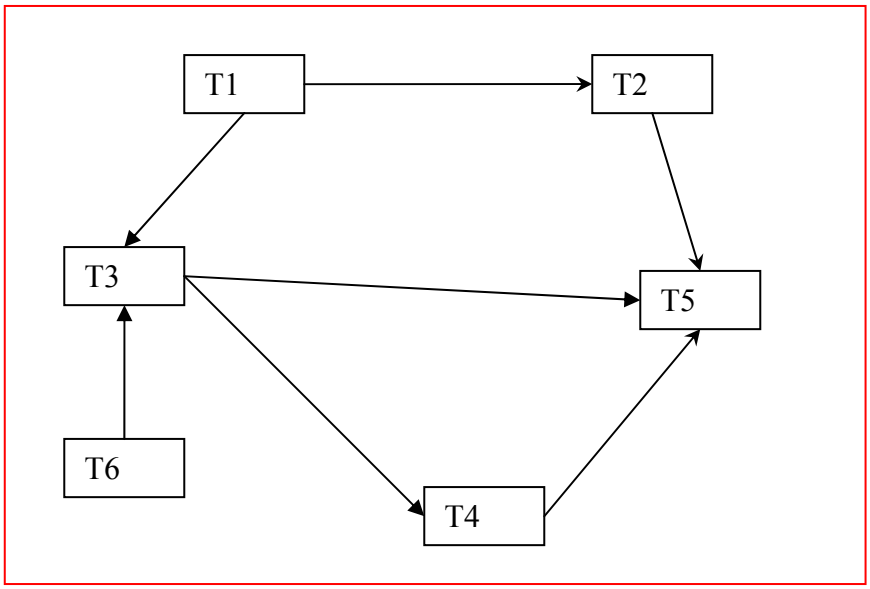

Figure 2: A sample directed graph

Node	Indegree	Outdegree	Equivalency Class
T1	-	2	C1 <0,2>
T2	1	1	C2 <1,1>
T3	2	2	C3 <2,2>
T4	1	1	C2 <1,1>
T5	3	-	C4 <3,0>
T6	-	1	C5 <0,1>

Equivalency classes (5 equivalency classes)

C_i	$P(C_i)$
C1 <0,2> = {T1}	1/6
C2 <1,1> = {T2, T4}	2/6
C3 <2,2> = {T3}	1/6
C4 <3,0> = {T5}	1/6
C5 <0,1> = {T6}	1/6

Entropy: $H = -\sum_{i=1}^{5} P(C_i) \cdot \lg\left(P(C_i)\right) = 2.252.$

Notes:

- How to generate the .NET and Java library graph? It is up to you (you can write a C# program to do it automatically by traversing the .NET documentation, or do it semi-automatically, in some other way).
- How to calculate H? Up to you. (You can write a C# program to calculate H from the generated graph, or do it semi-automatically – e.g. with Excel).

How your project is expected to look like.

- Provide your list of metrics that you devised and use in your evaluation.
- For each metric, explain why you choose the metric and what the intuition of this metric is. That is, why the metric provides a meaningful estimate of the complexity of the .NET and/or Java libraries. (for example, if your metric is "number of classes", then the intuition could be that "the more the number of classes in a library, the more difficult is to comprehend and work with this library".
- Provide a description of how the metric is calculated, applied, and/or used (for example, similar to the description for the entropy metric give above).
- Provide statistics from the results of applying your metric to the .NET and Java libraries (e.g. the number of classes, number of subclasses, H of the .NET graph, etc).
- Can you identify parts if the .NET library graph with higher complexity and/or parts with lower complexity?
- This is an open-ended project, in the sense that there is no fixed amount of detail that you have to achieve. At the minimum, you are expected to evaluate the two platforms using the Entropy metric, but you are expected to do more than that. Also, the amount of detail you can go even with the Entropy metric alone, varies. For example, do you apply metrics for classes only? … Or for methods of classes

as well? … Or for parts of the .NET as well? … in addition to the entire library? And so on.

- At the end, you are expected to hand in the following:
  - Implementation files of your work. Include some documentation in your code so it is feasible to understand what you are doing.
  - A presentation of your work (in my office, as will be scheduled toward rge end of the course. Tentative dates for presentations are November 19, 24, 26 toward the end of the course. (There will be presentations during those times, instead of class).
  - A written report that describes and presents your work. Include also a "user's manual" so it is feasible to understand how to compile and run your code. You can hand in your written report *after* your presentation, by December 1 at the latest.

How you can work.

You can do this project in groups of up to 5 (five) students per group. In case that you work in a group, you should also provide an individual report that describes – in your opinion, the amount of work done by each of the group members, including yourself. This report will also be used in assessing the work of each student individually, within your group.

Evaluation.

The evaluation of your project will be done on the basis of:

- The detail of the work you did.
- The comprehensiveness of your work.
- The quality of your written report.
- The overall quality of your work.
- As stated in class and in the course web site, evaluation will also be relative to all other projects.

Possible risks.

- If one or more members of your group happen to drop the course, the remaining members of the group are responsible for completing the project.
- Once a group is formed, members of a group cannot move to another group, unless there is agreement in doing so by *all* involved (i.e., all members of the group that will lose a member and all members of the group that will gain a member).
- Exchanging ideas between different groups is OK, but be reminded that evaluation is relative. (Your communicated ideas may give a better advantage to someone else).
- Copying code from one group to another is not allowed.
- Copying (non-copyrighted or copyrighted-but-allowed-to-use) code from the web is allowed, provided that you supply the reference (web address) from where you copied, give credit to the original author of the code, and specify clearly the parts of your code that have been copied.
- Copying chunks of text from articles/books/web and pasting it into your written report is not allowed, in general. If you need to do so, you should clearly indicate that this is copied text, by surrounding it with quotes and also by providing reference to the source from where you copied, including the exact page of the article where the copied text is found.