

Examination #1

Name: _____ Instructor's Solution _____ SUID: _____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All examinations will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be easiest.

1. Write an XML representation of the information needed to capture the quality output of Project #2, e.g., file name, each function name, cyclomatic complexity, and size. Now, using LINQ, write code to retrieve each part of that information.

Answer:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<QualOut>
```

```
  <file>
```

```
    <name>file1.cs</name>
```

```
    <function>
```

```
      <name>fun1</name>
```

```
      <cc>8</cc>
```

```
      <size>23</size>
```

```
    </function>
```

```
    <function>
```

```
      <name>fun2</name>
```

```
      <cc>5</cc>
```

```
      <size>42</size>
```

```
    </function>
```

```
  </file>
```

```
  <file>
```

```
    <name>file2.cs</name>
```

```
    <function>
```

```
      <name>fun3</name>
```

```
      <cc>9</cc>
```

```
      <size>48</size>
```

```
    </function>
```

```
  </file>
```

```
</QualOut>
```

```
class Program
```

```
{
```

```
  static void Main(string[] args)
```

```
  {
```

```
    XDocument doc = XDocument.Load(@"..\..\Qual.xml");
```

```
    Console.Write(doc.ToString());
```

```
    Console.WriteLine("\n\n");
```

```
    var files = from e in
```

```
                doc.Elements("QualOut")
```

```
                .Elements("file")
```

```
                select e;
```

```
    foreach (var file in files)
```

```
    {
```

```
      var q2 = from e in file.Elements("name") select e;
```

```
      Console.WriteLine("\n {0}", q2.ElementAt(0).Value);
```

```
      var q3 = from e in file.Elements("function").Elements() select e;
```

```
      int numFuncs = q3.Count() / 3;
```

```
      for (int i = 0; i < numFuncs; ++i)
```

```
      {
```

```
        Console.WriteLine("\n {0}", q3.ElementAt(3 * i).Value); // name
```

```
        Console.WriteLine("\n {0}", q3.ElementAt(3 * i + 1).Value); // cc
```

```
        Console.WriteLine("\n {0}", q3.ElementAt(3 * i + 2).Value); // size
```

```
      }
```

```
    }
```

```
    Console.WriteLine("\n\n");
```

```
  }
```

```
}
```

2. What is a delegate? How will using delegates support the development of reusable packages? Support your answer with an example.

Answer:

A delegate is a .Net mechanism to defer execution of a function until some event occurs. Usually the function invoker resides in library code and the executed function in application code.

Delegates are defined by the Delegate and MulticastDelegate classes. Each Delegate instance holds a reference to a function to be invoked. MulticastDelegate instances build linked lists of delegates to invoke functions for multiple targets.

Delegates need only know the function signature to be called. They don't use the Type of the target, so library code can build event notification without binding to the specific types used by an application. In this way, the library code doesn't need to change and so is reusable.

A good example of reusability is the NavigateWithDelegates demo, discussed in class. The navigator uses recursive descent to explore a directory rooted at some path and creates events for discovered files and for discovered subdirectories. Application code can subscribe for those events using the navigator without change.

3. Write a service contract for a communication channel that sends quality information, developed in Project #2¹ to a remote machine for analysis.

Answer:

The simplest way is to send the quality xml file as a string. We will need to prepend some xml elements so the receiver knows what was sent and where it came from. The service should provide an `OperationContract` method for clients to post the XML string and a method for the server to retrieve the string, which should not be an `OperationContract`. With this model, here's the service contract:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;

namespace HandCraftedService
{
    [ServiceContract(Namespace="QualAnal")]
    public interface IQualService
    {
        [OperationContract]
        void sendMessage(string msg);

        string getMessage();
    }
}
```

An alternative could use a `DataContract` with `DataMembers`: function name, size, and cc. See `WCF_MessagePassingCom` in the `CoreTechnology WCF` examples to see how this is done.

¹ See question #1

4. Write all the code for a FileManager package to support the file operations needed for Project #2².

Answer:

```
namespace MT1Q4
{
    public class FileMgr
    {
        private string patt;
        private bool recurse;
        List<string> files;

        public FileMgr(string pattern)
        {
            patt = pattern;
            files = new List<string>();
        }
        public List<string> getFiles(string path, bool recursive)
        {
            string[] localFiles = Directory.GetFiles(path, patt);
            foreach (string file in localFiles)
                files.Add(Path.GetFullPath(file));
            if (!recursive)
                return files;
            string[] directories = Directory.GetDirectories(path);
            foreach (string dir in directories)
                getFiles(dir, recursive);
            return files;
        }
    }
}
class Program
{
    static void Main(string[] args)
    {
        FileMgr fm = new FileMgr("*");
        List<string> files = fm.getFiles("../\\..", true);
        foreach (string file in files)
            Console.WriteLine("{0}", file);
        Console.WriteLine("\n\n");
    }
}
```

² You may wish to write the relevant class declarations and come back to fill in methods as time permits.

5. What is meant by the term "software architecture"?

Answer:

Software architecture describes the structure of a software system, e.g., its packages, their responsibilities and interactions, and describes those features which make the system serve the needs of its users. The architecture document (OCD) analyzes the users and how they interact with the system, the structure, and any critical issues associated with the system development.

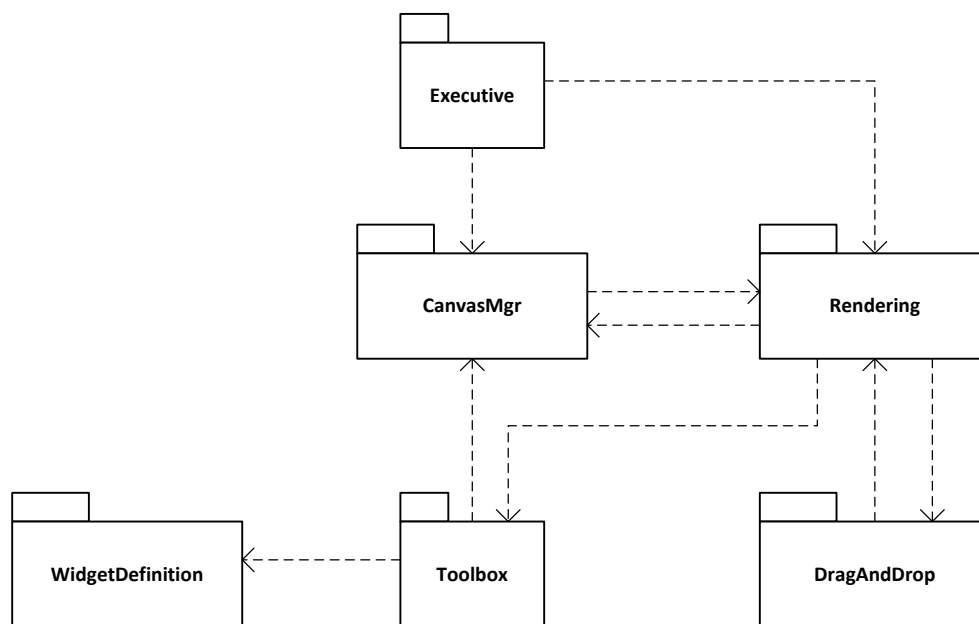
The purpose of the architectural description is to present a clear view of the system's concept unencumbered by the many implementation details needed to build a large system. The architecture describes the system concept, not its implementation.

6. Draw a package diagram for your design of Project #4. Please provide some detail for the selection of components and placement on the drawing canvas as well as other important parts.

Answer:

Package diagrams for a Project are not unique. That is, there are usually more than one correct diagram. The important properties of a package diagram is that each package is focused on a single activity and all the needed calling relationships are shown.

One reasonable diagram is shown below. The Executive is responsible for starting the application. The CanvasMgr represents the current diagram as a collection of connected shapes. Rendering is the core of a WPF application. It's responsibility is to display the CanvasMgr's current diagram state. It uses DragAndDrop to support user control of shape positions. Toolbox is a container for shapes to display, allowing selection of a user specified diagram element.



7. Write an Executive Summary for your OCD for Project #1. Remember that summarizes your project concept, not what you did while preparing the document.

Answer:

This project has two goals: to provide quality analysis of a potentially large set of files, based on function sizes and cyclomatic complexities; and to measure the distance between files based on cyclomatic complexity. Its principle users are developers and Quality Assurance personnel.

The tool should support the addition of other quality metrics and does so through the use of a rule-based parser that makes it easy to incorporate new rules without changing any of the existing code.

The distance measure is weak in that it can only tell if two files are, or are not, nearly the same. A goal for the next version is to attempt to find regions of code that are similar to support refactoring into a less redundant form. This prototype project does not do that, but will provide a stable platform for experimenting with algorithms for redundancy analysis.

The prototype will require developing packages that: collect files for analysis, provide parser rules and actions, evaluate the distance measure, and display the results. Development of all the packages will be straightforward with the exception of the display package. That will need to support both the analysis of small sets of developer's packages and the analysis of an entire baseline which may typically consist of hundreds of packages.

We do not anticipate any significant technical difficulties in developing this prototype tool.