

Examination #2

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 all the code for a thread-safe RelationshipTable used to store the type relationships found in Project #2.

Answer:

```
class RelInfo
{
    public string instance { get; set; }
    public string relationship { get; set; }
}
class RelationshipTable
{
    private Dictionary<string, List<RelInfo>> relationships;
    private object locker_ = new object();

    RelationshipTable()
    {
        relationships = new Dictionary<string, List<RelInfo>>();
    }
    T lockit<T>(Func<T> f)
    {
        lock (locker_) { return f.Invoke(); }
    }

    public void add(string src, RelInfo tgtRif)
    {
        lockit<bool>(() =>
        {
            if (relationships.Keys.Contains(src))
                relationships[src].Add(tgtRif);
            else
            {
                List<RelInfo> list = new List<RelInfo>();
                list.Add(tgtRif);
                relationships[src] = list;
            }
            return true; // not used
        });
    }
    public bool remove(string src)
    {
        return lockit<bool>(() => relationships.Remove(src));
    }
    public string[] typeKeys()
    {
        return lockit<string[]>(() => relationships.Keys.ToArray<string>());
    }
    public List<RelInfo> getRelationships(string type)
    {
        // returning List<RelInfo> is thread safe since
        // contents are strings and so immutable
        return lockit<List<RelInfo>>(() => relationships[type]);
    }
}
```

2. Project #4 requires you to implement a message-passing communication system based on WCF. When a client starts up it may send a request to a specified server for a list of all its projects. Create a message type for that request that has two public properties for send and receive addresses. You may use additional properties if you think they are needed. Now create a message type for the server's reply. Finally write a service contract that accepts either one of those messages.

Answer:

```
[DataContract(Namespace="MT2Q2")] // works for request and reply
public class SvcMsg
{
    public enum Command { Projects, ProjectList }; // Needs bigger set of enums
    [DataMember]
    public Command cmd { get; set; }
    [DataMember]
    public Uri src { get; set; }
    [DataMember]
    public Uri dst { get; set; }
    [DataMember]
    public string body { get; set; } // Used to send XML for structured data

    public void ShowMessage()
    {
        Console.WriteLine("\n Received Message:");
        Console.WriteLine("\n    src = {0}\n    dst = {1}", src.ToString(), dst.ToString());
        Console.WriteLine("\n    cmd = {0}", cmd.ToString());
        Console.WriteLine("\n    body:\n{0}", body);
    }
}

[ServiceContract(Namespace="MT2Q2")]
public interface IMessageService
{
    [OperationContract(IsOneWay=true)]
    void PostMessage(SvcMsg msg);
}

[ServiceBehavior(Namespace="MT2Q2")]
public class MessageService : IMessageService
{
    public void PostMessage(SvcMsg msg)
    {
        msg.ShowMessage();
    }
}
```

3. Write XML that represents a few of the types from the Parser project. Include the type's namespace and package. Now write one or more LINQ queries that extract all that information.

Answer:

```

/*
 * <types>
 *   <type cat="class">
 *     <name>Parser</name>
 *     <namespace>CodeAnalysis</namespace>
 *     <filename>Parser.cs</filename>
 *   </type>
 *   <type cat="class">
 *     <name>CSemiExp</name>
 *     <namespace>CSsemi</namespace>
 *     <filename>Semi.cs</filename>
 *   </type>
 * </types>
 */

static void Main(string[] args)
{
    "MT2Q3 - Type Queries from XML".Title();
    XDocument doc = MakeQ3Xml(); // see MTF14-Code -> MT2Q3 for details
    Console.WriteLine("\n{0}", doc.ToString());

    "\nResults of Query for type name values".Title('-', false);
    var q = from x in doc.Descendants("type")
            .Elements("name")
            select x.Value;
    ShowQuery(q);

    "\nResults of Query for namespace values".Title('-', false);
    q = from x in doc.Descendants("type")
        .Elements("namespace")
        select x.Value;
    ShowQuery(q);

    "\nResults of Query for filename values".Title('-', false);
    q = from x in doc.Descendants("type")
        .Elements("filename")
        select x.Value;
    ShowQuery(q);

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

```

See MTF14-Code -> MT2Q3.cs for more details

4. Pick two of the users groups for Project #2 and write the uses part of the corresponding OCD. You should be able to do this in less space than provided on this page.

Answer:

Your answers should provide specifics and discuss impact of the needs of each user group on the design of the Analyzer Tool. Part of your grade depends on how clearly you describe the user's needs and how the tool helps. Your grade does not depend on the quality of your English grammar.

- Developers:
Need to analyze their own code for quality factors and analyze code on which their code depends for content, e.g., types defined and their relationships.
- Impact on Design:
Design needs to support:
 - Finding code easily using GUI browse facility that can interrogate remote machines as well as local machine (implemented in Project #4).
 - Provide options for selecting specific packages, showing code metrics (see next item) and type relationships.
 - Provide options for display so the developer can focus on some specific aspect of their design.
- Quality Assurance:
Need to analyze quality metrics: function line count, function size, and type fan-out, e.g., the number of types on which a given type depends. If too large that suggests that the parent type may be trying to do too much. It also makes reuse more difficult, e.g., the type's package may depend on many other packages, and it is harder to understand the specifics of what this type does.
- Impact on Design:
QA may need to analyze hundreds of packages, so they need a batch mode that can run unattended, requiring no user input, and which writes its output in some persistent form, e.g., XML file(s).

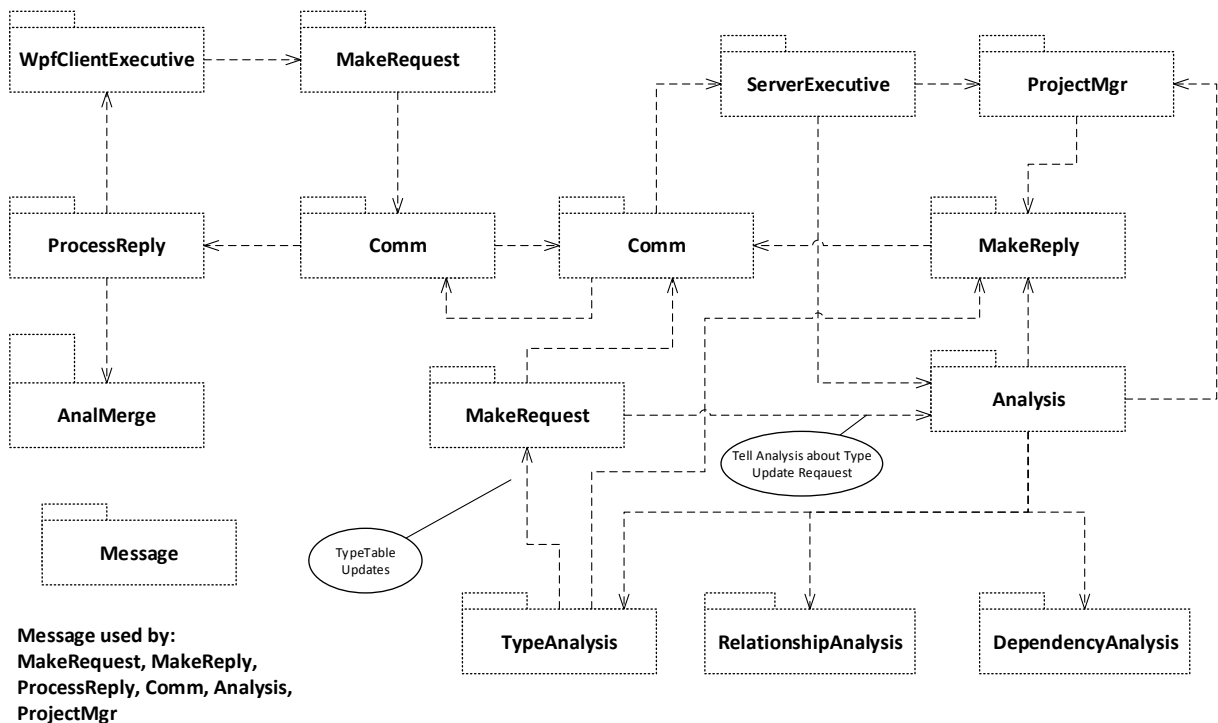
Other interesting User Groups are Testers and Team Leads. Managers aren't likely to use a code analysis tool, at least in their role as Manager. A Software Architect is likely to use the Analyzer from Project #4 to inspect the package dependency structure, but not so likely to use the tool from Project #2. This tool focuses on software design issues.

5. Draw a package diagram for the system of Project #4 and give a brief description of the responsibilities of two of the most important packages.

Answer:

For clarity lower level packages like Parser, ActionsAndRules, and XMLProc are not shown.

- WpfClientExecutive
Responds to user inputs by creating request messages using MakeRequest and Displays results provided by ProcessReply. ProcessReply may be required to merge results from several different server replies. Note that AnalMerge may display server data, using Dispatcher, progressively, as the data comes in from each server. Thus the dependency information may complete in stages.
- ServerExecutive
Responds to incoming request messages with the services of FileMgr and Analysis. FileMgr and Analysis compose responses using MakeReply. TypeAnalysis sends and receives messages concerned with TypeTable updates. Note that the Analysis package is responsible for ensuring that all available TypeTable updates have arrived.



A reasonable alternative is to use a Master Server. You need to show and discuss the type updates and results merging and which packages participate on Master Server, other Servers, and Clients. It is not a good idea to have a Master Server collect all the files to be analyzed, so merging of Analysis Servers results is necessary.

Your grade depends on how well I understand your processing based on your diagram and discussion and if the call flows seem to be accurate. Note that very brief discussions are appropriate (see above).

6. Write a lambda that accepts two doubles and adds them and returns the result. Write another lambda that accepts two doubles and multiplies them. Now show how a delegate can call them both with a single delegate method invocation. Please use either an Action or Func delegate with the correct signature (only one will work).

Answer:

```
class MT2Q6b
{
    delegate void CallType(double arg1, double arg2);

    double addResult;
    double mltResult;

    static void Main(string[] args)
    {
        "MT2Q6b - Two lambdas invoked".Title();
        MT2Q6b q6b = new MT2Q6b();

        CallType caller = new CallType(
            (double arg1, double arg2) => q6b.addResult = arg1 + arg2
        );
        caller += new CallType(
            (double arg1, double arg2) => q6b.mltResult = arg1 * arg2
        );
        caller.Invoke(2.5, 3.5);
        Console.Write(
            "\n addResult = {0}, mltResult = {1}\n\n",
            q6b.addResult, q6b.mltResult
        );
    }
}
```

7. What characteristics distinguishes a Client/Server system from other system configurations.

Answer:

Client/Server systems have these characteristics:

- The server manages data that is shared between multiple clients.
- Servers do not initiate any communication, they simply respond to possibly concurrent Client requests.
- Servers usually run continuously.
- Usually clients reside in machines separate from the server machine.

Usually Client/Server communication is synchronous. It can be made asynchronous using a duplex channel. However, if one-way asynchronous communication is used, then the server must initiate a reply when its results are ready, so that is no-longer Client/Server, but instead Peer-to-Peer communication.