

# SOFTWARE COLLABORATION FEDERATION

---

**INSTRUCTOR: JIM FAWCETT, Ph.D.**

---

## OPERATIONAL CONCEPT DOCUMENT

**KARTHIK YOGENDRA BANGERA**

**FALL 2016**

## Table of Contents

1	Executive Summary.....	8
2	Concept .....	10
3	Users and Uses.....	11
3.1	Developers .....	11
3.1.1	Uses of SCF .....	11
3.2	Quality Assurance (QA) Personnel .....	12
3.2.1	Uses of SCF .....	12
3.3	Administrator (Admin) .....	12
3.3.1	Uses of SCF .....	13
3.4	Technology Architect (Tech Arch).....	13
3.4.1	Uses of SCF .....	13
3.5	Project Manager.....	13
3.5.1	Uses of SCF .....	14
3.6	Clients.....	14
3.6.1	Uses of SCF .....	14
3.7	Business Analyst (BA) .....	14
3.7.1	Uses of SCF .....	15
4	Brief description of the SCF components .....	15
4.1	Repository Server .....	15
4.2	Test Harness Server.....	15
4.3	Client .....	16
4.4	Collaboration Server .....	16
4.5	Virtual Display System.....	16
5	Critical Issues.....	17
5.1	Same file name issue.....	17
5.2	Two users accessing a file at the same time .....	17
5.3	Search time for files in repository.....	17
5.4	Frequent search for same files .....	18
5.5	Large file size.....	18
5.6	Repository server overload with requests.....	18
5.7	Connectivity issues.....	18
5.8	Faulty test cases .....	19

6	Organizing Principles.....	19
6.1	Notification .....	19
6.2	Caching.....	19
6.3	WCF channel .....	19
6.4	Bug free code .....	20
6.5	Versioning .....	20
6.6	File locking.....	20
6.7	Asynchronous tasks.....	20
6.8	Dependencies of the file .....	20
7	Storage Management Subsystem .....	21
7.1	Organizing principles.....	21
7.2	Uses and Users.....	21
7.2.1	Clients.....	21
	Uses of SMS.....	21
7.2.2	Test Harness Server.....	22
	Uses of SMS.....	22
7.2.3	Collaboration Server .....	22
	Uses of SMS.....	22
7.2.4	Repository Server.....	22
	Uses of SMS.....	22
7.3	Structure .....	23
7.4	Responsibilities .....	24
7.5	Activities.....	24
7.6	Critical Issues.....	25
7.6.1	Sending invalid request.....	26
7.6.2	Connectivity issues.....	26
7.6.3	Large file size .....	26
8	Client .....	26
8.1	Uses and Users.....	27
8.1.1	Developers .....	27
8.1.2	Technology Architect (Tech Arch).....	27
8.1.3	Project Manager.....	27
8.1.4	Quality assurance team .....	27
8.2	Structure .....	28

8.3	Responsibilities .....	29
8.4	Activities.....	29
8.5	Critical Issues.....	31
8.5.1	Access control based on user role .....	31
9	Repository Server.....	31
9.1	Organizing principles.....	31
9.1.1	Versioning .....	31
9.1.2	Check-in policy .....	32
9.1.3	Message passing .....	32
9.1.4	File dependencies .....	32
9.2	Uses and Users.....	33
9.2.1	Developers .....	33
9.2.2	Project Manager.....	33
9.2.3	Quality assurance team .....	33
9.3	Structure .....	34
9.4	Interfaces: .....	35
9.5	Responsibilities .....	36
9.6	Activities.....	36
9.7	Critical Issues.....	38
9.7.1	File Search .....	38
9.7.2	File Deletion .....	38
9.7.3	File Insertion.....	39
9.7.4	Parallel Access.....	39
10	Build Server .....	39
10.1	Responsibilities .....	39
10.2	Users .....	39
10.2.1	Users of the test harness .....	39
10.3	Activities.....	40
11	Test Harness.....	41
11.1	Organizing principles.....	41
11.1.1	Building source code.....	41
11.1.2	Generate test results .....	41
11.1.3	Automated tests.....	41
11.2	Users .....	41

11.2.1	Developers .....	41
11.2.2	Quality Assurance (QA) Personnel .....	42
11.2.3	Managers .....	42
11.2.4	Clients.....	42
11.3	Uses.....	42
11.3.1	Developers will be able to write code faster with fewer bugs .....	43
11.3.2	Addition and updating of tests by the developers.....	43
11.3.3	Application under test remains stable:.....	43
11.3.4	Check if Application doesn't have any bugs accidentally introduced: .....	43
11.3.5	Tracking the bugs is so much easier: .....	43
11.4	Structure .....	44
11.5	Interfaces: .....	46
11.6	Activities.....	47
11.7	Critical Issues.....	49
11.7.1	Ease of Use .....	49
11.7.2	Performance related to test objects .....	49
11.7.3	Demonstration .....	49
11.7.4	Invalid XML file.....	49
11.7.5	Appropriate access to the path of the dynamic link libraries .....	50
11.7.6	Huge log file size .....	50
12	Collaboration Server .....	50
12.1	Organizing principles.....	50
12.1.1	Tools need to be user friendly .....	50
12.1.2	Access control .....	50
12.1.3	Project Management .....	51
12.2	Uses and Users.....	51
12.2.1	Developers .....	51
12.2.2	Technology Architect (Tech Arch).....	51
12.2.3	Project Manager.....	51
12.3	Structure .....	52
12.4	Responsibilities .....	52
12.5	Activities.....	53
12.6	Critical Issues.....	54
12.6.1	Data Security .....	54

12.6.2	Connection drop .....	54
12.6.3	Task overlap .....	55
13	Messages.....	55
13.1	Organizing principles.....	55
13.1.1	Message Structure .....	55
13.1.2	Sender and Receiver addresses .....	55
13.1.3	Metadata related information .....	55
13.2	Uses and Users .....	56
13.2.1	Comm Service .....	56
13.3	Types of messages .....	56
13.3.1	Repository-Test Harness messages.....	56
13.3.2	Client-Test Harness messages.....	56
13.3.3	Client-Repository messages .....	56
13.3.4	Client-Collaboration messages .....	57
13.4	Critical Issues.....	57
13.4.1	Incorrect format.....	57
13.4.2	Secure transmission .....	57
14	Comm Service .....	57
14.1	Organizing principles.....	58
14.1.1	Proxy list.....	58
14.1.2	Queues .....	58
14.2	Uses and Users.....	58
14.2.1	Storage Management Subsystem .....	58
14.3	Activities.....	58
14.4	Critical Issues.....	59
14.4.1	Multiple messages at the receiver's end at the same time .....	59
14.4.2	Transmission of messages securely .....	59
15	Core Services and Policies.....	60
15.1	Core Services.....	60
15.1.1	Blocking queues .....	60
15.1.2	Managing metadata.....	60
15.1.3	Notifications.....	60
15.1.4	WCF Communication .....	60
15.2	Policies .....	61

15.2.1	Ownership Transfer policy .....	61
15.2.2	Single Ownership policy .....	61
15.2.3	Group Ownership policy .....	61
15.2.4	Caching policy .....	61
15.2.5	Versioning policy .....	62
16	Error handling .....	62
17	Administration and security.....	63
18	Baseline Management .....	63
18.1	Responsibilities .....	63
19	Appendix .....	64
19.1	Message Encryption and Decryption Prototype (Critical issue prototype) .....	64
19.2	Test Harness prototype.....	66
20	Conclusion .....	79
21	References .....	80

## Table of Figures

Figure 1: SCF Architecture.....	11
Figure 2: SMS Architecture .....	23
Figure 3: Activity diagram for SMS.....	25
Figure 4: Structure of Client .....	28
Figure 5: Activity diagram for client.....	30
Figure 6: Structure of Repository.....	34
Figure 7: Activity diagram for Versioning.....	37
Figure 8: Activity diagram for check-in .....	38
Figure 9: Activity diagram for build server.....	40
Figure 10: Test Harness Structure.....	44
Figure 11: Test Driver .....	45
Figure 12: Activity Diagram for test harness.....	47
Figure 13: Structure of Collaboration server .....	52
Figure 14: Activity diagram of Collaboration server .....	54
Figure 15: Activity diagram of Comm service .....	59



## 1 Executive Summary

In the modern world of information, storage management has become a primary concern to deal with. Storage management would involve the products and services that help us in storage and retrieval of information efficiently. The storage could be either related to the data which is being used regularly by the end users or the backup data which is a necessity in scenarios like corrupt new data or loss of data. When storage management is extended to software industry where large amount of data flows throughout the organization it becomes a huge task to manage such large volume of data. With the growing size of projects, software development could become a very challenging task if appropriate means are not devised to manage and monitor the large volume of data. For this very reason, a Software Collaboration Federation (SCF) is designed to manage software development process.

The term Software Collaboration Federation (SCF) defines a collection of clients and servers that are associated with software designed to support activities of a software development team in various stages of software development lifecycle. It helps in creating and publishing plans for development, writing concept documents, creating and editing work packages, scheduling work packages, allocating resources to work packages, writing and publishing specification and design documents, preparing new source code packages, making the existing source code packages reusable, building execution images and loadable libraries, executing tests like unit tests, integration tests, regression tests, performance tests, stress tests, and acceptance tests on the source code before its deployed in the server, deploying software executables and documentation, reporting progress, key events, and failures. Hence, a SCF helps in continuous build integration of the packages and testing it before they make it to production, making sure that the integration is smooth and an error free code is deployed in the production environment.

The SCF is comprised of a federation of clients and servers:

- **Collaboration Server:** It supports project management, storing work package details and providing collaboration tools. The server is designed to help users involved in a common task to achieve their goals.
- **Repository Server:** It stores source code and documents using a configuration control system and provides exploratory tools based on metadata associated with each stored item.
- **Build Server:** It builds and caches execution images and libraries as needed for test and deployment.

- **Test Harness Server:** It loads and executes test images provided by the build server. The test server also provides deployment activities.
- **Virtual Display Server:** It provides a sophisticated interface for publishing management information and collaboration activities that involve viewing source code, documents, diagrams, sketches, and webcams to enhance personal and team interactions. This is a prime vehicle for collaboration between remote teams.
- **Development Clients:** It provides user access to the many facilities of SCF including check-in and check-out of code and documents, creation and scheduling tests, and viewing results.
- **Virtual Display Client:** A specialized client that focuses on managing displays of documents, code, web cam views, and various charts and tables that describe progress and interactions between development groups.

SCF will use a Storage Management Subsystem (SMS) that provides a core SCF service and is composed of data and event managers hosted on each of the clients and servers in the SCF Federation. It also includes a data and event coordinator that supervises the operation of all of the individual data and event managers. SMS improves storage space use, allows central control of external storage, and enables you to manage storage growth more efficiently.

The Storage Management System provides a number of services including storing and managing:

- Routing lists for communication of messages and notifications
- Layout information for VDS: where are the windows, what content do they display
- Metadata for source code and documents in the repository
- Check-in/out status for code and documents in the repository
- Test configurations and results
- Lists of services and servers available to clients
- Templates for displays and messages.

## 2 Concept

The services provided by the repository, test harness, collaboration and virtual display system help the many clients to work in collaboration; this is achieved through the Software Collaboration Federation that follows the client server architecture. We would use web services to access the centralized data from a remote client. The clients would have GUI using Windows Presentation Foundation (WPF), whereas the services would be utilised through the Window Communication Framework (WCF). The use of web services would simplify the code structure and benefit the SCF in achieving its objectives.

The SCF would consist of several distributed virtual servers, but these servers could be projected as a single instance using the virtual environments. All the servers would collaborate using the message passing communication so as to provide an effective development environment. Since the servers are distributed, they would enable the communication using the Windows Communication Foundation (WCF), whereas the clients would use the Windows Presentation Foundation (WPF) to consume the services provided by the server. With regard to the steps of communication between a client and a server, the client would have to authenticate to the server and provide the username and password so that the server can authenticate and assure that it is serving the intended client's request. After the authentication, the client would be able to access the services offered by that server.

The below SCF architecture showcases the components that offer services to the clients:

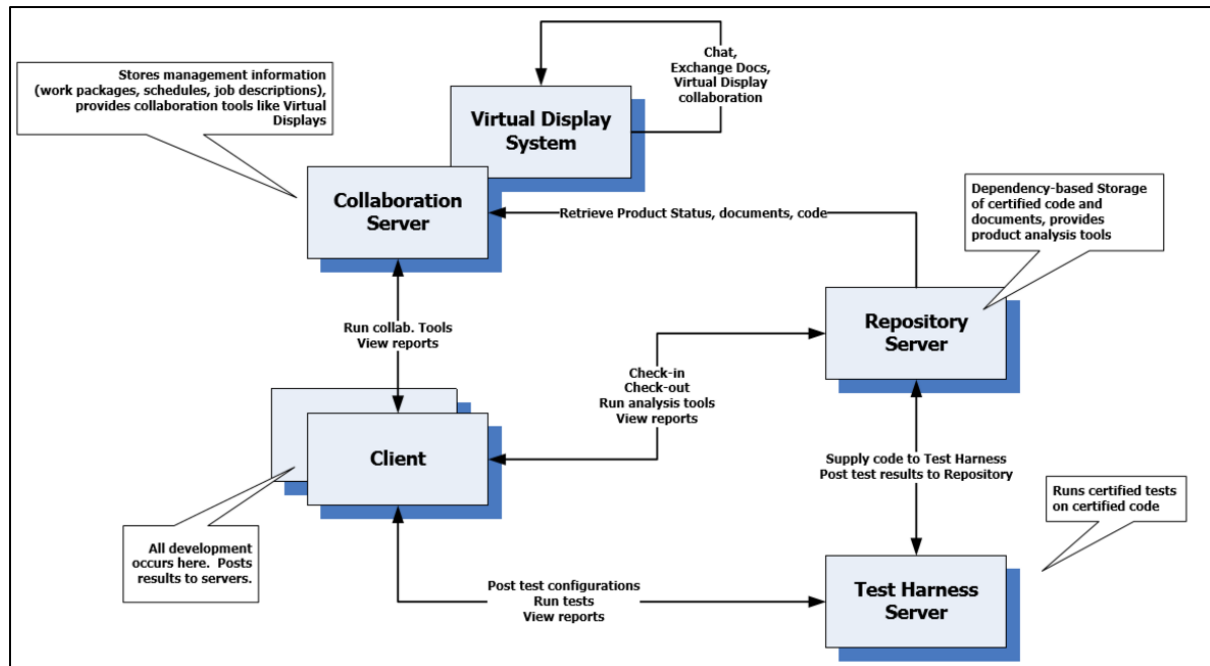


Figure 1: SCF Architecture

### 3 Users and Uses

#### 3.1 Developers

The developers would be responsible for the design, development and maintenance of the application. SCF provides the perfect environment for the developers for collaboration where they can communicate and share files with developers in other locations. Integration of the code would be the last thing on their mind as the SCF would handle all this and hence the developers can be more focussed on developing efficient code.

- They can share and present their ideas to the other developers instantaneously via collaboration.
- They would work with the business analyst to regularly check if all the functional requirements that have been transitioned into a technical document have been met.
- They would need to create code which requires the least or no rework.
- They would work with the QA team and the business analyst to build a better application.

##### 3.1.1 Uses of SCF

- Provides a faster and a more efficient mode of integration of the developer's code with the large system.
- Provides extraction and check-in of source code files from and into the repository.

- Provides a user friendly GUI and also gives the option of a roll-back to older dependent source code links when new source code links are creating an issue.
- Provides notifications in case the check-in fails/passes, also whenever there is a new source code file version available.
- Can display/ download the test results through logs and hence keep a track of all the tests that have failed or passed.

### 3.2 Quality Assurance (QA) Personnel

The QA team would have a list of test cases against which the system would be thoroughly tested and would check if all the functional requirements that were listed out by the business analyst have been met. This way they make sure that the system is free of any defects.

- Check and list out any performance issues while performing tests
- Perform a check on the coding standards and in case the defaulters list would be handed over to the author of the code (i.e. the developer), who would in return make the corrections.
- Write the test cases which would test all the functionalities of the application from both the technical and business point of view; they could also automate the tests so that testing is faster.

#### 3.2.1 Uses of SCF

- The test drivers that would be provided by the QA team would test the developer's code and provide the details to the QA team as to how it either passed or failed.
- The test results provided would give an elaborate detail on how the tests have either passed or failed and will be saved as logs that can be referenced in the future.

### 3.3 Administrator (Admin)

This role could be taken over by one of the team leads or it could be made into an entirely independent role. The admin would be responsible for data backups, access issues, permission related issues, monitoring of the server logs and the licensing of the application and software used by the application.

- Control the access of any user to all the servers.
- Monitor and fetch system logs in case of any server issues.

### 3.3.1 Uses of SCF

- A user friendly interface would be provided for monitoring, granting and revoking access to various servers of the SCF.

## 3.4 Technology Architect (Tech Arch)

The technology architect holds the complete responsibility with respect to the design of the project. The development team follow the design laid out by the technology architect. The technology architect works with the business analyst and makes the architectural design document for the team to follow.

- Present the architecture of the project to the team and decide on the technology that needs to be used for the project.
- Would be responsible for the design of the project.

### 3.4.1 Uses of SCF

- Give a complete understanding of the dependencies and the interactions among the different components in the system.
- Can check in the documents related with the design of the project and other project related presentations which could in turn be accessed by the manager or team leads whenever required. It could also send a notification to the team whenever the document is updated to a newer version.

## 3.5 Project Manager

The project manager would be the first in command of the software development project. A large development project would have quite a few team leads that are managed by the tech arch and the project manager. The project manager would ensure that the development is completed as per the schedule; the manager would regularly be requesting for the weekly status updates of the tasks from the teams (development and QA teams) and in return would forward a report to the client after every milestone is completed. The manager would also

- The manager would regularly check the progress of the project and schedule demos to satisfy and gain the client's confidence.
- The manager would keep a check on the growth of the team and the also individual development within the team.

### 3.5.1 Uses of SCF

- Gives a good idea of the actual development progress of the project.
- Provides the options to check the total tests passed and failed for particular scenarios and gets the logs for any particular tests when the project manager queries for the same in the application.
- Maintains proper versioning of the project management files in the server.

## 3.6 Clients

They are the customers who require this application.

- After the tests are executed, the clients should receive the results of the tests and only then approve the deployment in the other environments.
- Check the status of the project by scheduling meetings with the concerned project manager.
- Check for all the test results and coding standards approval provided by the QA team to the project manager or ask a brief demo on the same.

### 3.6.1 Uses of SCF

- Can check the test case results, for their own satisfaction of the application under development.
- Customer can have access to the project's documents but with a read only access, this way they can also see the progress.

## 3.7 Business Analyst (BA)

There can be scenarios when the requirements change; it could be due to either due to misunderstanding of a requirement from the customer or a technological constraint. The tech arch can discuss the same with the customers and get the requirements altered. However, after any change in the requirements is approved, it's the responsibility of the BA to change and update all the documents related to the project.

- Would be responsible for all the updated documents in the project.
- In case of any clarifications required with regard to the business requirements, it's the responsibility of the BA to convey the understanding of the same to the developer and QA teams.

### 3.7.1 Uses of SCF

- Provides the publishing of documents and also provide notifications to all the stakeholders whenever the document is updated.

## 4 Brief description of the SCF components

### 4.1 Repository Server

The repository server plays a major role in collaboration of the source code. It usually contains test drivers, build scripts, test results, test logs etc., along with different versions of the source code. Having a central repository for code helps in minimizing the communication overhead between the developers and speeds up the process of code integration. Versioning of the source code helps in rolling back the code at any instance.

- Notifies the user with regard to the check in status of the code.
- It publishes the test results after the test harness has performed the tests and sends the test outcome; it also shows the source code information and metadata to the client.
- Invokes the test harness to run the tests on the source code once it's checked in.
- It helps the project managers use the collaboration server to check the project status and reports.
- It compiles the checked in code for build errors.

### 4.2 Test Harness Server

The test harness runs several tests on the newly checked in code to ensure that the code works as per the requirement and does not break the existing code. It helps in maintaining a bug free code. It configures the test requests, generates test results and logs which would be published through a service. After the code is checked-in the repository server, the test harness would be invoked to run the tests on the recently moved source code. Once the tests are run, the results are again stored in the repository and published to the stake holders.

- It's used to perform scheduled regression tests.
- Returns test results and logs to the repository server.
- It loads test requests defined by the xml messages and it also loads test images provided by the build system.



### 4.3 Client

It can be seen as a network of computers used by developers and other team members to collaborate development work. It has access to the repository server where the files are stored and it can update the same and save the latest version in the repository server. The repository server should extract the files for the client whenever it is requested. SMS provides the communication channel for the clients to communicate with repository server, test harness and collaboration server.

- It has a user interface to use the different services offered by the SCF.
- Allows collaboration between different project team members.
- Displays the check-in and check-out status of the source to and from the repository server.
- Can query for files on the repository server.
- It displays results of the tests carried out on the source code.
- It displays the contents of the file as well as the file's metadata and other relationship information of the file extracted from the repository server.

### 4.4 Collaboration Server

It contains all the project management information. It contains information about the work packages, schedules and task descriptions which helps in task distribution among the team members. It helps to plan, organize, and manage resource pools and come up with project estimate.

- It helps in scheduling the code deployment to the server and also helps in collaboration between project teams that work remotely.
- It provides information which would be useful for the project manager to give the project estimates and billing.
- It helps the project managers use the collaboration server to check the project status and reports.

### 4.5 Virtual Display System

The VDS is a component of the collaboration server, it is backed up by the collaboration server. It provides an interface for collaboration between the different development teams. Collaboration activities such as chat, viewing source code, documents, diagrams, sketches, videos conferencing using webcams are supported by virtual display system.

- Notifies the user with regard to the check in status of the code.
- It publishes the test results after the test harness has performed the tests and sends the test outcome; it also shows the source code information and metadata to the client.

- Invokes the test harness to run the tests on the source code once it's checked in.
- It helps the project managers use the collaboration server to check the project status and reports.
- It compiles the checked in code for build errors.

## 5 Critical Issues

Below discussed is a list of critical issues related to implementing a SCF along with their proposed solutions:

### 5.1 Same file name issue

- An issue could occur when two different users upload 2 different files into the repository with the same name at the same time. This would lead to incorrect query results as there would be two files with the same name but different content.
- We could use a queue to resolve this issue, when the two users try upload two files with the same name, the Repository Server would put those two requests in a queue. So, when second file is to be uploaded, then the server sees that a file already exists with that name and it can send a notification that a file with the same name already exists, please change the file name.

### 5.2 Two users accessing a file at the same time

- There could be a scenario when two clients are trying to access the same file in repository, this could lead to a deadlock and also there could be a scenario; where one client is trying to edit the file while the other is trying to access it.
- File Locking could be a solution to this scenario, where a lock is put on the file when it's accessed by a user. This wouldn't allow the other user to access the file.

### 5.3 Search time for files in repository

- There could come a time when the numerous files in the repository and eventually the request for the file will take longer than usual to respond to.
- Indexing and caching of the files could be resolution; the most frequent queries can also be cached.

#### 5.4 Frequent search for same files

- There could be a scenario where the client is requesting to access the same set of files on the server which is an overhead for the server.
- To resolve the issue, the resources that are frequently accessed can be cached either in client or server side so that the request gets served quickly; and in case of caching on client side the call to server can be avoided. But there could be a scenario where a newer version of the file is uploaded in the server, in that case a notification needs to be sent to the user that a new version is available in the repository and the cache for that for that resource needs to be erased.

#### 5.5 Large file size

- Transferring of large files could be an issue due to client and network limitations.
- File streaming and file chunking can be used if the file is too large.

#### 5.6 Repository server overload with requests

- There could be a scenario where multiple clients post their requests to the repository server which would lead to multiple accesses to the database which would lead to instability.
- The requests get queued on the blocking queue and each request would be dequeued and served through a single thread.

#### 5.7 Connectivity issues

- Suppose a connection cuts off between a client and a server during a file transfer, the file transferred is partial and possibly corrupted. Both the client and server would be unaware of the situation.
- The client and server both should be aware of the status of the current file transfer as it happens. Messages can be passed to and fro to determine file transfer completion. When the end user uploads a file, the server should return a message like “receiving file” and once a file has been completely transferred, a message like “file received” must be sent to the end user. In case, the end user doesn’t receive the message, the user should try uploading the file once again and the server should delete the partially received file.

## 5.8 Faulty test cases

- Users may upload empty or logically flawed test cases that may ensure that the test passes but can introduce severe bugs in the system. Users can also upload multiple test cases for say testing package and dependencies which have the same test cases.
- There needs to be a tool which parses these test cases fed to the test harness and analyses the contents of the test cases. The tool could also determine whether multiple test cases match and returns only unique test cases.

## 6 Organizing Principles

Below discussed are organizing principles which are assumptions around which related objects can be built keeping it as reference:

### 6.1 Notification

- The end users can view all the notifications through the user interface; they would receive notifications for almost every event on the server. For instance when the clients check-in the code into the repository server, the server would notify the clients about the status of the check-in along with the test run results.
- Collaboration server could also provide notifications with regard to document sharing, meeting and chat.

### 6.2 Caching

- Caching minimizes the overhead on the server side and speeds up the application. In the case of client side caching, files that are rarely updated on the server should be marked for caching. So the client would not have to request for the file from the server since its available in the client's cache. Suppose the client side cached file is updated in the server, the server should publish a message stating that the concerned file has been updated and that the client would need to request for the latest copy of the file.
- Caching on the server side is done for the files that are frequently accessed, this way the files that are cached on the server side can be immediately served on request.

### 6.3 WCF channel

- The Windows Communication Foundation (WCF) message passing communication channel would be used for client server communication.

- The WCF services provided by the servers would serve the client requests.

#### 6.4 Bug free code

- After check-in of the files in the repository, files are compiled and checked for any build errors.
- After it passes the check, the test harness would run various tests on the same code to check whether it affects the existing baseline. After this check is also passed, the code is committed to the repository.

#### 6.5 Versioning

- To enable the concept of rollback, each time the source code is checked-in to the repository, it would get a version number.
- The server would save the older versions and assign a new version number each time the code is checked in. This would be really helpful when the code has to be rolled back.

#### 6.6 File locking

- To avoid the scenario of two or more clients accessing the same file at the same time, the repository server will implement the file locking system which would lock the file as soon as it is accessed by a client.
- Once the file is locked, other clients should receive a message stating it is being used by another user.

#### 6.7 Asynchronous tasks

- The client need not wait for the server to respond to its request as the requests on the server side will be handled asynchronously.
- Hence the client can carry on with some other task after making a request to the server, as the server will send the response once it's ready.

#### 6.8 Dependencies of the file

- The repository holds all the information about the file along with their dependencies
- When the files are modified, the old and the new dependency information would be stores in the server; this would be very useful in the scenario of a rollback.

## 7 Storage Management Subsystem

Storage Management Subsystem (SMS) is composed of data and event managers, this subsystem is present on each of the clients and servers in the federation system. Each of the SCF components are provided data management and communication services by the SMS. It also has data and event coordinators which control the operation of all data and event managers. Management of data in a system like SCF would be difficult, hence, the need for data and event managers; which leads to a dedicated virtual machine to handle this issue and run the management system. This way, the components of the SCF won't have the overhead of management. The SMS would be distributed and available in each component, it would have the roles of formatting events and data, as well as communicating with other components of SCF through a communication channel.

### 7.1 Organizing principles

- **Routing table:** This would Communication between the different components of the SCF is important for the collaboration to take place. The SMS maintains a routing table which will have the destination address and also the proxy of the destination which would make the communication much easier.
- **Managing events:** The subscribers of the events that are being handled by SMS are registered in the SMS itself. Hence, the SMS goes through this list each time an event gets triggered and publishes the required notification to all the subscribers in the list.

### 7.2 Uses and Users

The users of these services are the various components of the Software Collaboration Federation like Clients, Collaboration server, Repository and Test harness server.

#### 7.2.1 Clients

The clients use SMS to communicate with other components of the Software Collaboration Federation (SCF), clients utilise the data and event services provided by the SMS for the communication. The client can also send the request along with the destination details and the rest is taken care by the storage management subsystem.

#### Uses of SMS

- The SMS can communicate with the repository with regard to the check in and check out code and get the job done.

- It can fetch check in and check out status anytime in the repository as well as build the requests for running the tests on the baseline code in the repository.

### 7.2.2 Test Harness Server

The test harness server will work with the SMS to retrieve baseline code from the repository and generating test case results and publishing them. It will receive check in messages from the repository to start the testing on the baseline code. Once the code is successfully built, the code undergoes various tests and the test results are generated. This happens with the help of storage management subsystem.

#### Uses of SMS

- Send the test results and logs to the clients and repository server.
- Fetch the checked in code from the repository along with the baseline code for running tests.

### 7.2.3 Collaboration Server

Collaboration server helps in providing tools for collaboration along with handling and assigning packages. It also has a Virtual display system which handles the collaboration tools; this server receives requests with regard to collaboration tool. The SMS will have the list of tools available for collaboration; it receives the requests and helps the server launch the tool with the VDS as an interface.

#### Uses of SMS

- Get the right collaboration tool for the virtual display system.
- Helps in scheduling meeting and get status about files likes documents.

### 7.2.4 Repository Server

The repository server closely works with the SMS for the check in and check out process. The server is a storage for all the code required for testing as well as documents. It also contains the dependency information of the files; this information is handled by the SMS. Hence, in case of a request for a file, the dependency information of that file is retrieved from the SMS and all the other dependent files are fetched in response.

#### Uses of SMS

- It handles the check in and check out requests from the clients.

- Manages the file dependency information.

### 7.3 Structure

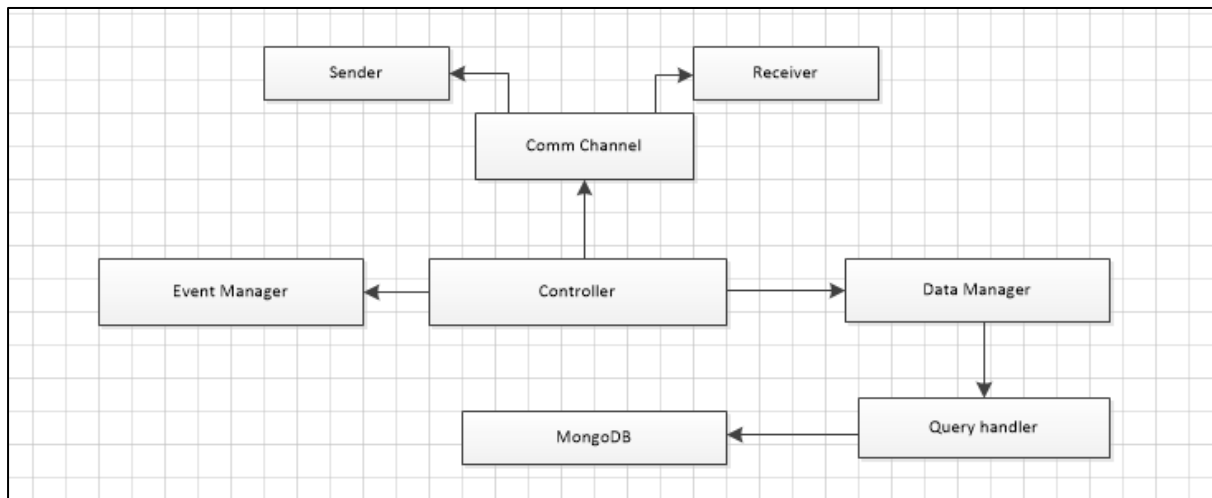


Figure 2: SMS Architecture

The above figure gives us an idea of the SMS architecture; it would contain the following components:

- **Comm channel:** This would be the communication channel for data transmission between SCF components. It would contain a sender and a receiver. The sender maintains a queue in which the messages get enqueued, it is then dequeued and sent to the destination by using a proxy list available in SMS. Similarly, the receiver also maintains a queue into which the messages get enqueued, they are dequeued one by one and processed.
- **Controller:** It controls the event manager, data manager and the comm channel packages of the Storage Management subsystem.
- **Data Manager:** It is responsible for data going to and fro from the system. The incoming data would be stored in the MongoDB.
- **Query handler:** It is responsible for communicating with the MongoDB and fetching information that the end user has requested or storing the information. The data manager will parse the data it has received to the query handler using an object.
- **MongoDB:** SMS would consist of a MongoDB, which would contain a list of the routing details and also the details of the proxy list which would help in fetching the proxy server.
- **Event Manager:** It is responsible for triggering new events as well as managing all incoming events. Based on the event, it decides what action needs to be taken.



## 7.4 Responsibilities

- Manages the file dependency or metadata information of the code and documents in the repository server.
- Manages the service of publishing check in and check out statuses of the code in repository.
- Fetches the information to be used by the clients through Virtual Display System.
- Fetches the test results and sends it over to the clients and repository respectively
- Will be responsible for all the communication between the SCF components.

## 7.5 Activities

Below mentioned are the activities performed by the Storage Management System:

- The SMS has all its services up and running. It receives a SCF component's request; the SMS can have a blocking queue so as to resolve the issue of serving multiple SCF component requests at the same time.
- Once the request is received, it would be analysed and checked if that request is valid, then it would be checked if it is related to data or an event.
- If it is related to the event manager, the manager would check as to what action needs to be taken for the particular event that has occurred and it would inform all the registered subscribers the details of the event that has occurred.
- If it is related to the data manager, the manager would check as to where the message needs to be delivered so that the request can be served, it would utilise the query handler and MongoDB for this purpose.
- SMS uses the communication channel for data transmission among different components of the SCF.

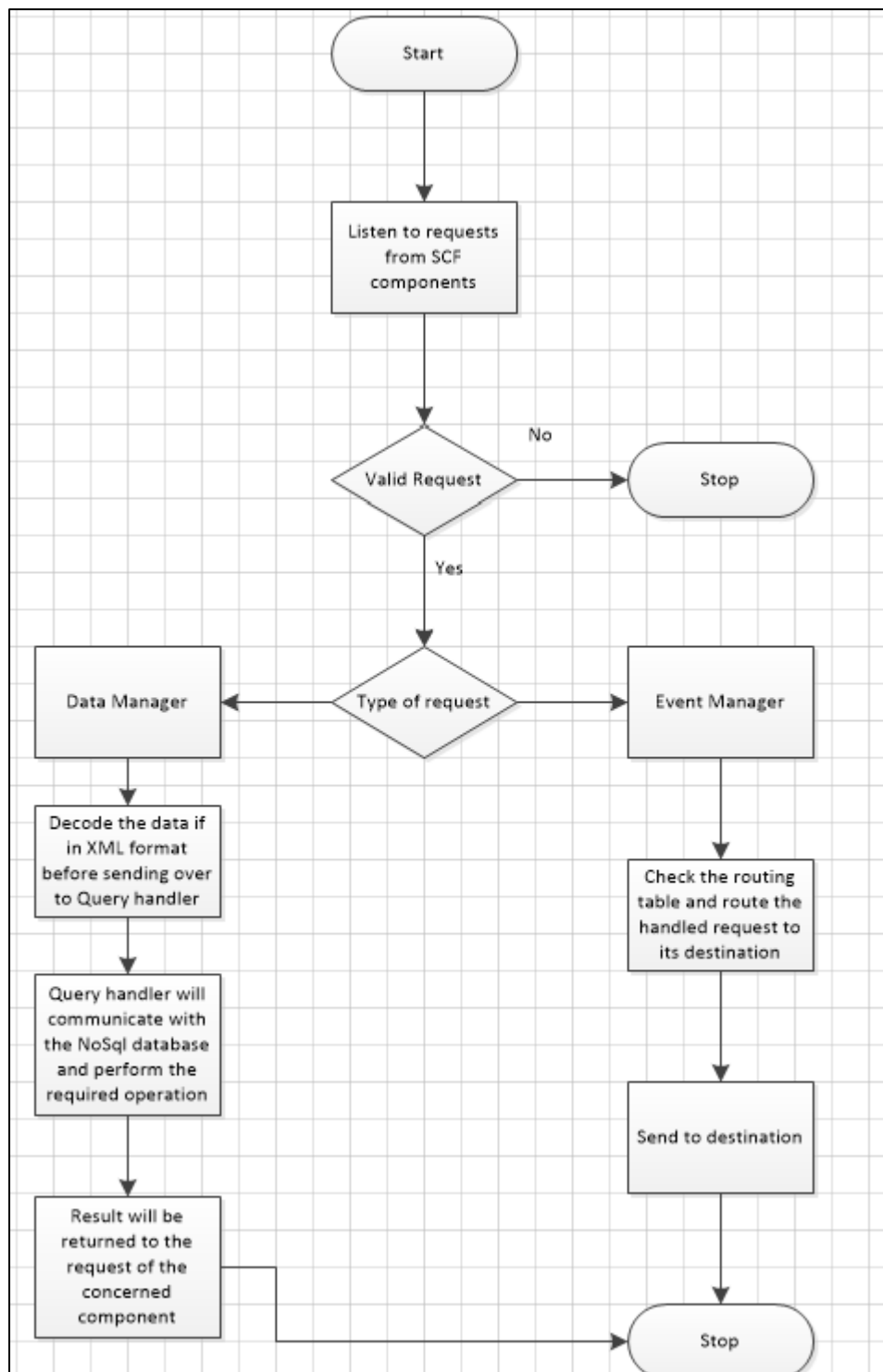


Figure 3: Activity diagram for SMS

## 7.6 Critical Issues

Below discussed is a list of critical issues related to implementing a SMS along with their proposed solutions:

### 7.6.1 Sending invalid request

- There could be a scenario where invalid requests could be sent to other SCF components, this would overload the network bandwidth and might decrease the performance of the system.
- A check should be introduced in the SMS to check the request that is being sent over, this way only valid requests will be sent.

### 7.6.2 Connectivity issues

- Suppose a connection cuts off and the SMS is in the middle of some data transfer, the sender and receiver would be unaware of the situation and junk data can enter the system.
- The SMS should handle the situation when the connection fails, it should try reconnecting and once successful; it should start a new transfer of the same file. Messages can be passed to and fro to determine file transfer completion. When the end user uploads a file, the server should return a message like “receiving file” and once a file has been completely transferred, a message like “file received” must be sent to the end user. In case, the end user doesn’t receive the message, the user should try uploading the file once again and the server should delete the partially received file.

### 7.6.3 Large file size

- Transferring of large files could be an issue due to client and network limitations.
- The storage management system (SMS) can implement the file streaming and file chunking can be used if the file is too large.

## 8 Client

The client will serve several users distributed across remote places by providing a user interface to have an access to the system. The client application will interact with the web services which provide access to the data and all the other services provided by the servers of SCF depending on the user role of the end user. The client application can improve the performance and usability by caching data. The end users are required to make a login to the client system in order to access the files and services of the repository server. Every user would be given role based access to the services. The client system will use WCF based message passing communication to communicate with the repository server.

## 8.1 Uses and Users

The client subsystem will be used by every end user who wants to interact with a component of the software collaboration federation (SCF).

### 8.1.1 Developers

- Extraction of source code files, test results and logs, checking in and checking out of the source code files.
- Can be used Run tests in test harness tool before checking in the source code in the repository.
- Can view all the dependent files related to a file in a project, which would help in better understanding of the structure of the code.
- Can view the notifications like meeting requests, new source code versions and test reports related notifications.

### 8.1.2 Technology Architect (Tech Arch)

- Uploading documents and presentations related to the project. These can be accessible to the project manager and team leads from repository.
- Schedule meeting requests; understand the dependencies among various components in the SCF.

### 8.1.3 Project Manager

- Manage the versioning of the project management files.
- Send meeting request and notification to a particular team member.
- Understand the progress of the project/teams.
- Can understand the real progress of the project team by viewing the project progress reports. Can view the logs of any particular tests that were held in the test harness

### 8.1.4 Quality assurance team

- Can build test cases by looking through the different files and documents that have been uploaded by the tech arch or developers which would help in understanding the business ends as well.
- Extraction of source code files, checking in of the test drivers and test code.
- Can be used Run tests in test harness tool before checking in the source code in the repository.

- Can view the notifications like meeting requests, new source code versions and test reports related notifications.

## 8.2 Structure

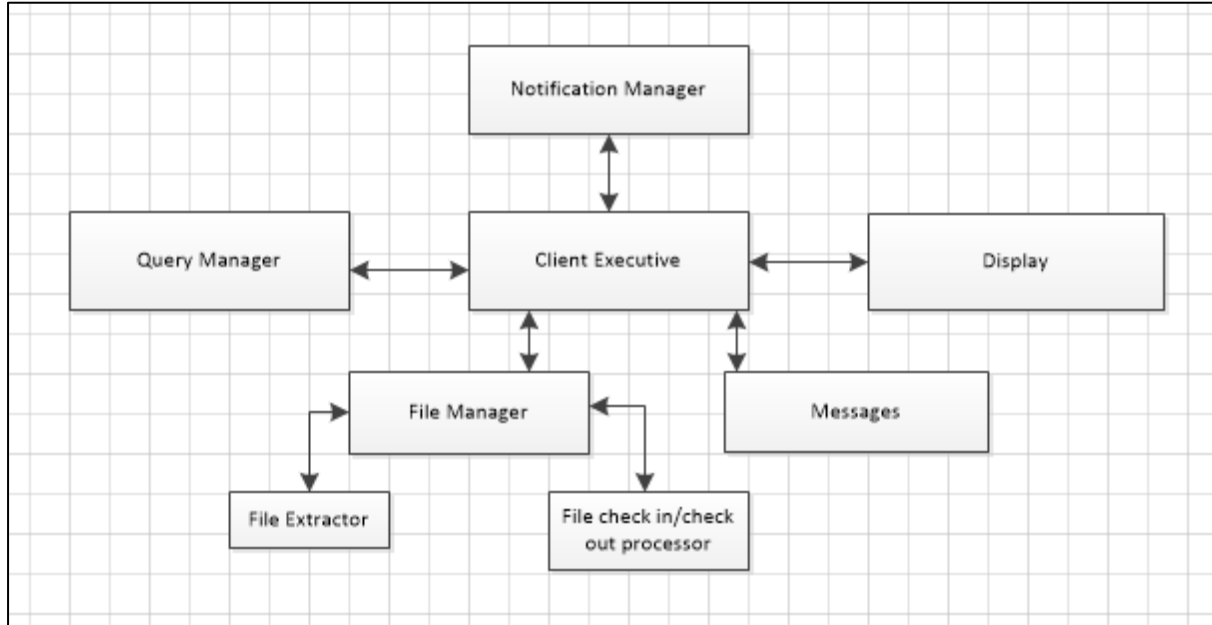


Figure 4: Structure of Client

The above figure gives us an idea of the Client architecture; it would contain the following components:

- **Client Executive:** It takes the end user request from the GUI (display) and sends it to the appropriate module. This package will wrap up all the modules of the client system.
- **Notification Manager:** It would be responsible to display all the notifications on the GUI to the end user, depending on the communication of the client with the servers of SCF, it would receive notifications regarding team schedules, meetings, work progress etc.
- **Display:** It provides a GUI interface to the end user to work with the application; this would forward the end user request to the client executive and also displays the results of the request to the end user.
- **File Manager:** The manager consists of the File extractor and file check in/check out processor. While the former would be responsible for file extraction requests to the repository server and retrieving the files along with their dependencies which would be displayed on the GUI, the latter would be used to commit source code changes or to access files from the repository.

- **Messages:** It's responsible for the message passing communication facility between the client and the server through WCF. It would be used to establish client-SCF component communication channels. It will use the blocking queue to send and receive data.
- **Query Manager:** It's responsibility is to route the query to the right query processor. For instance, all text based queries from the user should be sent to the text query processor. This could be handled through the GUI where the end user could select a text based query option and also expect the result for the same on the GUI. Similarly for a metadata based query, the metadata query processor would accept the query and will return the output to the GUI for the end user.

### 8.3 Responsibilities

- Allows the end user to access content of the selected file, the end user can also add/update the metadata of the file.
- It extracts the requested files from the repository server and also receives the test result logs from the Test Harness server.
- Provides a GUI to the end users who can also monitor the progress and view the definition of work packages.
- The end user can forward the check in requests of the packages to the repository.

### 8.4 Activities

Below mentioned are the activities performed by the Client:

- The client has to authenticate himself/herself before using the application.
- The user is asked to enter the login credentials again if not authenticated, else the user details are authenticated and the user role is validated by the application.
- Based on the user role, the user has different services offered from the different components of the SCF and the user can interact with these components depending on the service they require.
- For the services offered by the collaboration server, the client could request for the different collaboration tools on offer. For instance, chat tool for texting remote teams, or even better would be video conferencing or file sharing tools. The requested tool is then displayed on the client.
- For the services offered by the repository server, the client can check in or check out source code. The team leads and the managers can also view the progress and reports of the tests available in the repository server, they can also access the documents.

- For the services offered by the test harness, the client can send the test requests, once the testing is done, it will receive the test results logs. The test result will also be displayed on the interface. Team leads on the other hand can use the code analyser for validating the code before checking it in.

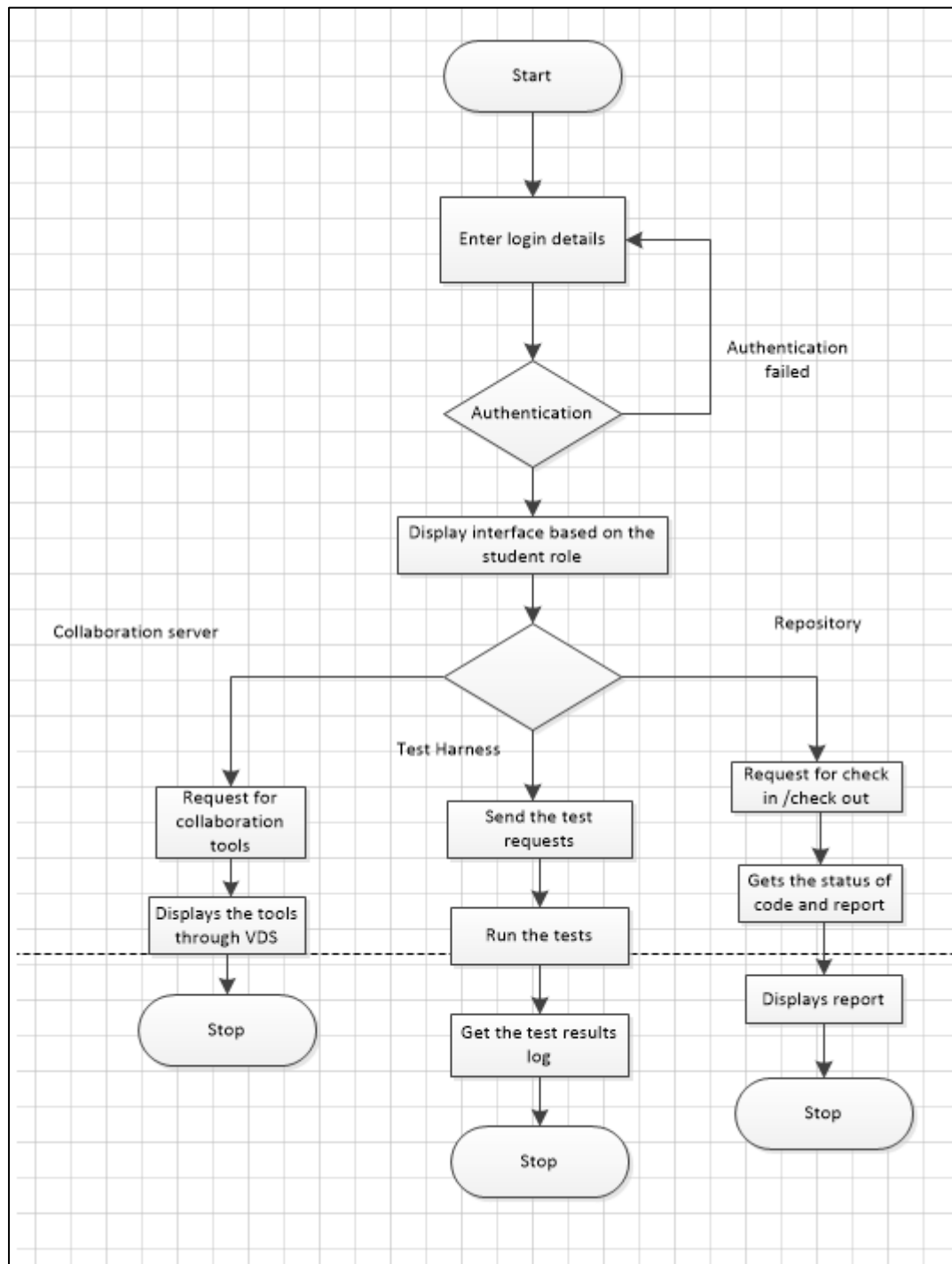


Figure 5: Activity diagram for client

## 8.5 Critical Issues

Below discussed is a list of critical issues related to implementing a Client along with their proposed solutions:

### 8.5.1 Access control based on user role

- The tools and services of the Software Collaboration Federation (SCF) shouldn't be accessible for all.
- This can be done at the time of authentication, where the user credentials can be extracted and based on their user role; they are presented with the right amount of access to the application.

## 9 Repository Server

The Repository server manages the collection of the source code representing the project's current baseline. The server manages dependency relationships between the files it stores. It provides end users the facility to extract source code files as well as insert them. The end users would first need to authenticate themselves in order to gain access to the repository server and then access the code in the repository based on the ownership policies that have been defined in the server. For inserting source code into the server, the user would need to check in and for extracting the same the user would need to check out. The check in and check out of the source code files would also be associated with the notifications that would in turn notify the owners of the current status of their files.

### 9.1 Organizing principles

#### 9.1.1 Versioning

Versioning helps the developers to roll back to the older versions of the code and to also update the user as to what is the latest version present in the server. The server should be able to convert a unversioned file into a versioned file. The repository should also accept a versioned file and change its version number at the time of check in. The versioning information can be saved in an XML file, each time a package is checked in, the file is traversed and the latest version against that file is incremented, in case the file name isn't available in the XML file, the file name along with the version number 1.0 would be inserted in the XML file.



### 9.1.2 Check-in policy

The repository server is responsible for a valid check in when a source code is checked in from the client.

- When the file which the client wants to check in is a source code file, the client is also responsible for providing all the required code that is dependent for this file for checking in.
- The repository server will check the file type being checked in, if it's a file other than a source code file, it will check if the file already exists in the repository, if yes, a successive version of the file is added in the repository; else it will create a new version and add that in the repository.
- For the source code files, the repository server will have send them to the test harness for performing the automated tests and generate the test results.
- Successful check in means that the source code file, dependent files and the test result logs have been stored in the code repository.
- Unsuccessful check in mean would send a notification to the user stating the test failed and the test results can be returned to the sender to check as to why the tests failed, or it could have also been due to network outage.
- Repository server will also comply with file versioning as well as sending notifications to developers regarding any version changes to any of the existing code versions.

### 9.1.3 Message passing

It aims to resolve computational problems by communicating with others servers through message passing.

### 9.1.4 File dependencies

The repository server will use a dependency graph to hold dependency relationships where the nodes are XML files which would identify dependent metadata files. The dependency information is embedded into these files. For traversing a graph, the metadata file reads the dependencies and fetches all the dependent filenames, after this a recursive search is performed on all dependent metadata files. Whenever a file is updated and checked into the repository server, new version of the metadata file is created. When this file is checked in, older ones are not replaced and a newer version is created.

## 9.2 Uses and Users

### 9.2.1 Developers

- Extraction of source code files, test results and logs, checking in and checking out of the source code files.
- Enables the developers to maintain a good understanding among them, in building and working on a project as a team.
- Can understand all the dependent files related to a file in a project, which would help in better understanding of the structure of the code.
- Provides facility for text and metadata search.

### 9.2.2 Project Manager

- Can check the status of the project from a remote location, the manager can also extract necessary file projects.
- Understand the progress of the project/teams irrespective of his location.

### 9.2.3 Quality assurance team

- Can build test cases by looking through the different files and documents that have been uploaded by the tech arch or developers which would help in understanding the business ends as well.
- Can review the code by accessing it through the repository and if any bugs found, the QA personnel would contact the concerned developer to fix it.
- The QA team could also suggest the changes to the developers and they would do a check on the coding standards too.

### 9.3 Structure

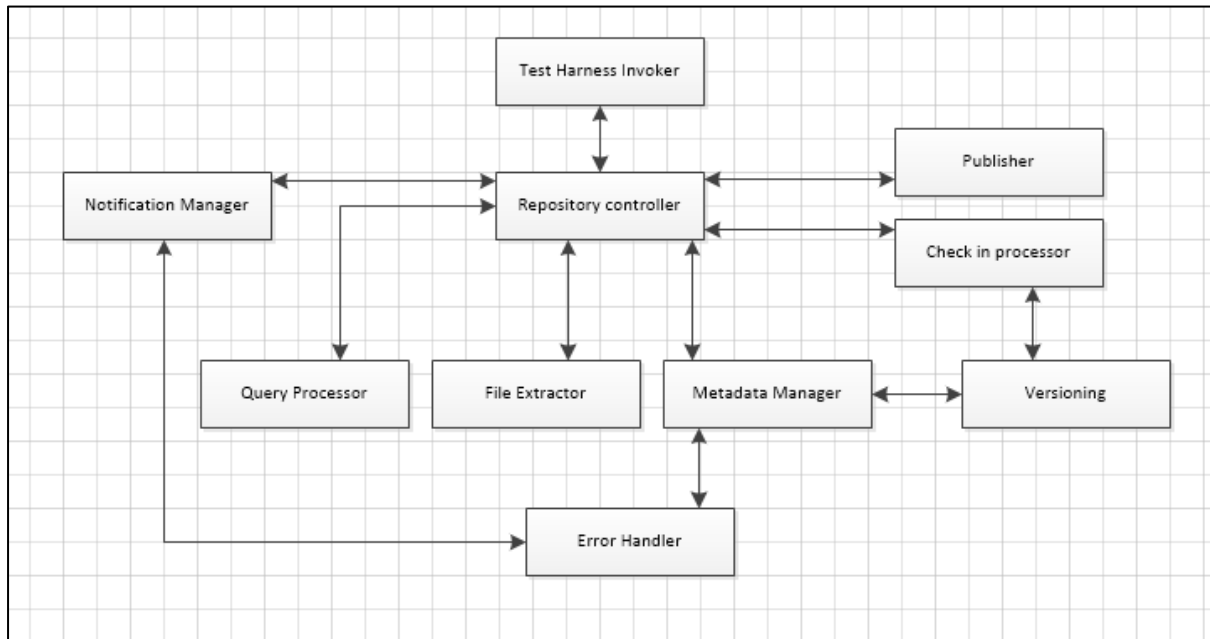


Figure 6: Structure of Repository

The above figure gives us an idea of the Client architecture; it would contain the following components:

- **File extractor:** It will provide the facility for extracting the file from the repository.
- **Notification Manager:** It would be responsible to display all the notifications to the client, depending on the communication of the client with the servers of SCF, it would receive notifications regarding team schedules, meetings, work progress etc. The notifications could be check in/checked out code and new version of the file available
- **Repository Controller:** It's the controller package of the repository server. All the packages in this server are monitored and coordinated.
- **Versioning:** The server will accept an unversioned file and update it into a versioned file. It also accepts versioned files and increments its version.
- **Test Harness Invoker:** It supplies the test drivers and test codes to the test harness and expects a test results log to be returned back once the testing has been completed.
- **Publisher:** It will publish the details of the tests results once the files have been successfully tested.
- **Check-in processor:** Once the file has been successfully tested, the metadata manager will generate the metadata file and return the appropriate message to the file check in processor package on completion. If the file is already existing , the metadata manager will update the metadata file.

- **Error Handler:** It will handle all the exceptions that occur in any of the packages discussed and appropriate error messages will be displayed on the client's GUI. It will do the same for unhandled exceptions occurring at the server.
- **Query Processor:** It's responsibility is to route the query to the right query processor. For instance, all text based queries from the user should be sent to the text query processor. This could be handled through the GUI where the end user could select a text based query option and also expect the result for the same on the GUI. Similarly for a metadata based query, the metadata query processor would accept the query and will return the output to the GUI for the end user.
- **Metadata Manager:** It accepts metadata details and adds tag elements to the XML file, this file is then forwarded to the XML Processor package to create a metadata file. Hence, this package can add and delete the metadata file.

#### 9.4 Interfaces:

The Repository interacts with the Client(s), Build Server and the Test Harness Server:

- **Interface between Repository and Client(s):** The Client displays file information with the help of a GUI whereas the Repository consists of code files and their respective metadata. The Repository it is responsible for servicing the end user queries. It also allows the user to check-in and extract files to a remote location and display the file information. The Repository allows the user to search for text and metadata from the files present on the server and presents the client with a set of files. The user can edit or create a new metadata file and upload it into the Repository Server. This can be done with the help of metadata manager that will offer such services to the end user. The client sends a request to be authenticated into the Repository by enter user details such as username, password. Based on the login and user role of the end user, the Repository creates a virtual server on the client's local machine. The user can then insert or extract files out of Repository by checking-in or checking-out respectively.
- **Interface between Repository and Test Harness Server:** The Repository interacts with the test harness server to provide code that needs to be tested. Upon the completion of testing, the test harness server sends the test results to the Repository, so that test results can be posted. When the test harness requires all of the file dependencies to test, it requests for all these files from the Repository. On receiving these files, tests are conducted and the results are sent to both the Client and the Repository.

## 9.5 Responsibilities

- Allows extraction of packages to the user and maintaining record of the files that are check out and checked in.
- Manage collection of source code and files dependent to it. Managing the check in of these files and maintaining a version history.
- Generating metadata files related to the metadata information of each file checked in and maintaining the parent child relationship information of the files.
- Responding to the client's text based or metadata based queries.
- Provide the test harness with source code files and the dependent files, test drivers and test codes; to run the tests and accept the logs returned by the test harness.
- Return the test results when a client requests for it from the repository.
- Sending notifications to clients whenever a newer version of file is available on the server.

## 9.6 Activities

Below mentioned are the activities performed by the Repository:

- The client has checked in files in the repository server.
- The server checks if the file exists or not by checking for the file name. If already exists, we obtain the latest version of the file and then increment the version depending on how major the release is.
- The metadata file's version is also incremented the same way. The client then notified from the server that the file has been successfully uploaded in the repository server.
- If the file doesn't exist in the repository server, then the file is committed with an initial version and its metadata file is also saved in the repository with the same version.
- The client s made the owner of that file and is notified that the file has been uploaded successfully to the Repository server.

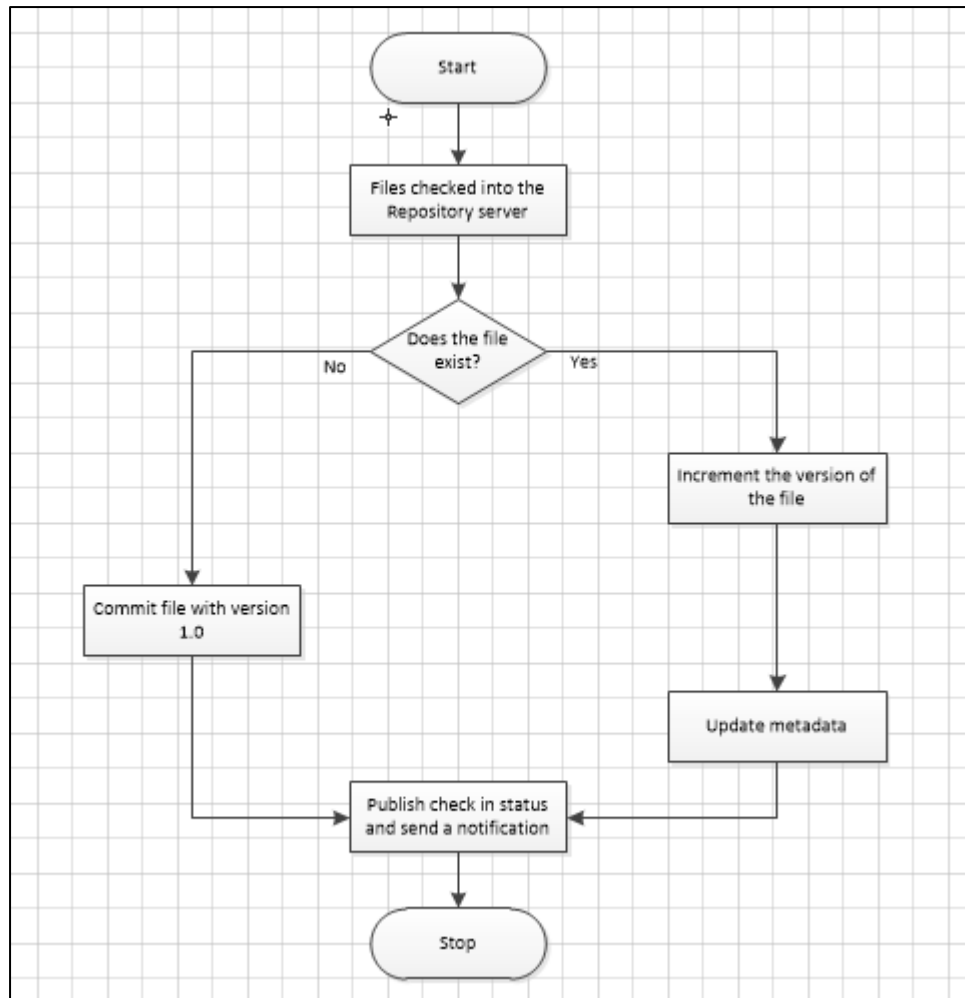


Figure 7: Activity diagram for Versioning

- Clients have checked in source code files.
- The test harness is invoked to get the source files tested and all the files are sent to the test harness.
- The test harness will build the files and then test it. The test harness will inform once the testing is done and the test results and logs are sent to the repository and client who requested for the testing.
- This way the test results are stored in the repository and the client is notified with the test results.

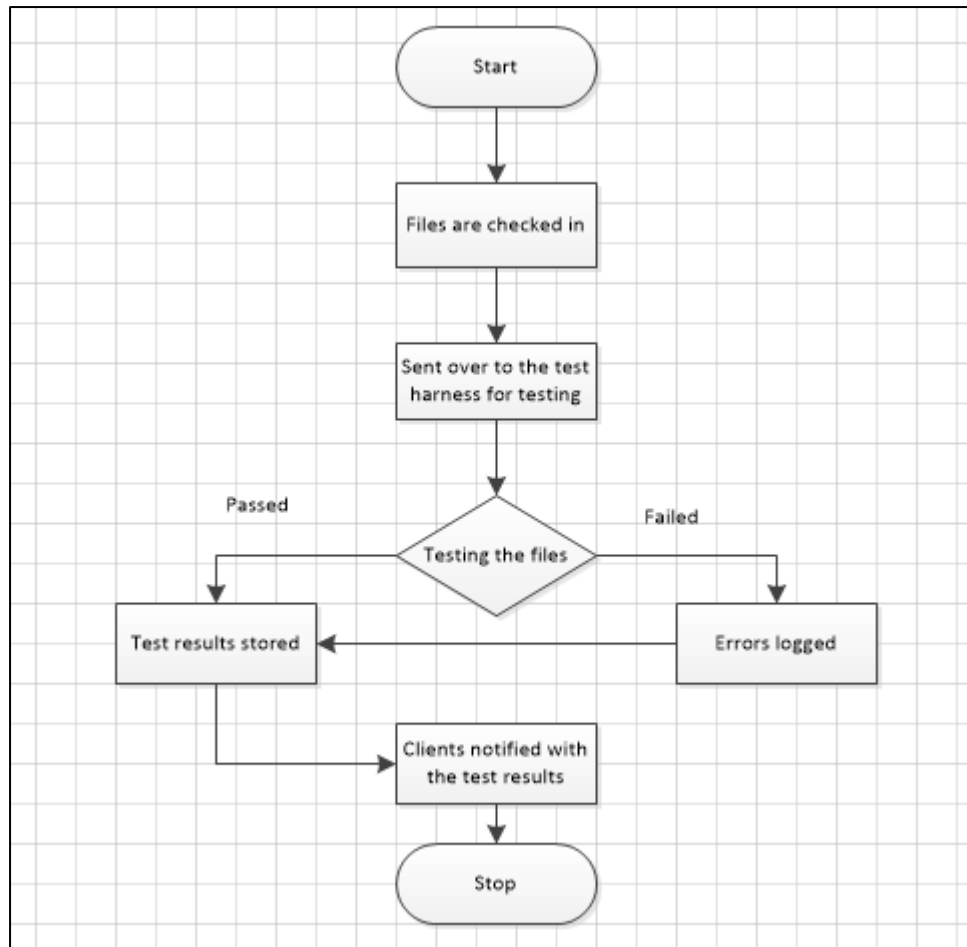


Figure 8: Activity diagram for check-in

## 9.7 Critical Issues

### 9.7.1 File Search

- Since all the files are stored here, searching a file involves going through a large set of files present in the repository.
- Caching the results on the server set would reduce the overhead of searching the entire repository for files that are regularly accessed. This would speed up the execution time.

### 9.7.2 File Deletion

- There could be scenarios where a file may be accidentally deleted and that file could have had multiple dependencies involved.
- Caching the results on the server set would reduce the overhead of searching the entire repository for files that are regularly accessed. This would speed up the execution time.

### 9.7.3 File Insertion

- There could be a scenario where the user would upload a file which might not be of the valid format into the server, inserting might not be an issue but an exception occurs on trying to open such a file.
- The resolution to this issue could be having a filter of acceptable extensions for a file, if the file passes this filter, it can go ahead and be uploaded in the repository server; else we can display a message to the user saying that the file uploaded is not of the acceptable format.

### 9.7.4 Parallel Access

- There could be a scenario where one or more clients may try to access the same file in the repository, leading to a deadlock or an exception where one user is trying to access the file while the other is trying to edit at the same time.
- The resolution to this issue is file locking, whenever a file is accessed by a user, the file is locked and no other user can access or edit it. A blocking queue also serves the purpose of tackling parallel file access issues.

## 10 Build Server

Build server uses a build script to build libraries for a specified set of packages. Every source code checked-in to the Repository requires a build and if the build fails the check-in fails. Every test configuration also requires a build before executing tests in the configuration. In this system, the build server would be a subsystem of the Test harness.

### 10.1 Responsibilities

- Processes build requests from the test harness.
- Builds libraries as requested and uploads to the test harness server for testing if successful.
- Caches previously used successful builds of source code files.

### 10.2 Users

#### 10.2.1 Users of the test harness

- They check in the source code for build and would require a clean build to run successful tests in the test harness.



### 10.3 Activities

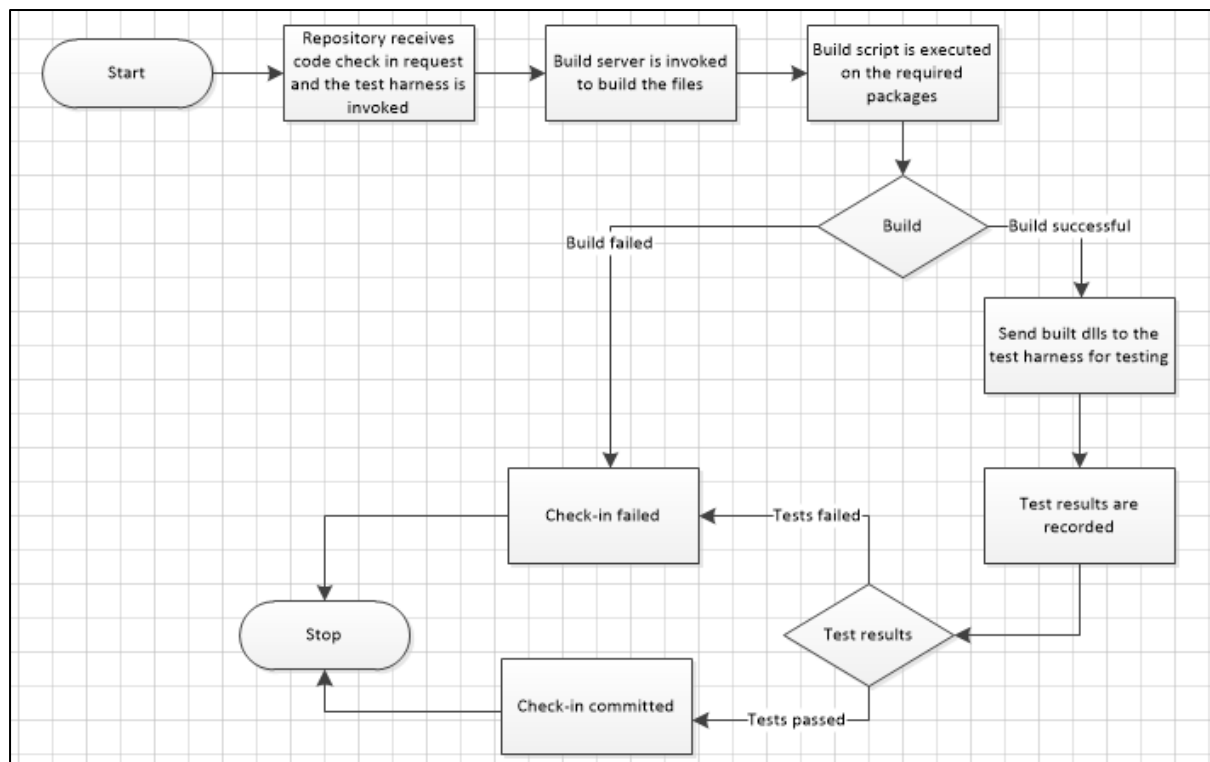


Figure 9: Activity diagram for build server

Below mentioned are the activities related to the build server:

- After the repository server gets a check in request for a source package, the test harness is notified and receives a test request for the same with the package name.
- The test harness receives all the information regarding the files its dependent on from the repository, after which the build server is invoked to build the required files for testing.
- The builder server builds the required dynamic load libraries (dlls) using the build script and sends them over to the test harness for testing.
- If the build fails, the code is not checked in, the check in would even fail if the successfully built code fails I one of the tests.
- If all the tests pass, the repository server ha the approval to commit the check in of the package.

## 11 Test Harness

The sole purpose of the Test Harness would be to check that the application under test is functioning properly even after the addition or modification of code. The new code would be tested against the existing baseline code and the test harness would ensure that the code is meeting all the requirements and not breaking any functionality of the application. The test results would be logged into a file and whenever there are any exceptions thrown in any tests, the client could open log file and match the details of the timestamp and the error thrown; so as to get the error description and accordingly work on the fix. The test harness would receive the test requests in xml format through a blocking queue which would be explained later in this document. All the test code and test drivers are stored in the repository. Hence, we can access and load all the dlls required for testing from the repository.

### 11.1 Organizing principles

#### 11.1.1 Building source code

The build server is a subsystem in test harness that would build the source code from the repository server before its tested in the test harness.

#### 11.1.2 Generate test results

The test harness server generates the test results after the source code is built and the results are sent over to the client the repository server.

#### 11.1.3 Automated tests

The automated tests will be run on the source code against the baseline code after the source code is built. The tests on the newly added components against the existing baseline code make sure everything is working fine.

### 11.2 Users

Mentioned below are the users who would need to utilize and interact with the test harness:

#### 11.2.1 Developers

- They are the primary users of this tool.
- They would mostly be focussed on the ease of use of the tool.
- Developers could use the Test Harness to test their own test code and test drivers in order to check the functionality of the Test Harness.

- Since they are also the developers of the application under testing, they can assist the QA personnel when it comes to briefing or giving them a so called knowledge transfer of the application under test prior to any of the testing taking place.
- The Test Harness would be used to check if application under test meets all the functional requirements; in other words it would undergo functional testing to check if the system works correctly.

### 11.2.2 Quality Assurance (QA) Personnel

- They would focus on the performance aspect of the tool.
- Performing several tests against the entire baseline, checking the performance during large scale testing with an eye on the code quality.
- They would possess the role of checking for any faults or performance issues in the functionality of the test harness.
- And if at all, there are any issues experienced during this testing process, the concerned QA personnel could report the same to the Manager.

### 11.2.3 Managers

- The manager would mostly require access and check the test logs of the tool.
- The manager would look for a good logging activity in the tool and hence would time and again check the test activity data before he is confident enough of giving a demo and using the tool for client applications.

### 11.2.4 Clients

- The clients would be interacting with the Test harness through the GUI where they would send the XML files that would contain the list of the test codes and drivers.
- After the tests are executed, the clients would receive the results of the tests in logs which are stored in the repository.
- Before sending the test requests to the test harness, the client would also be sending the test codes and test drivers to be tested to the repository server.

## 11.3 Uses

Below is a list of the possible uses of the test harness for the users:

### 11.3.1 Developers will be able to write code faster with fewer bugs

- The introduction of the test harness would relieve the hardships of performing labour intensive testing on the code. Rather the developers can now concentrate on coding and need not worry much as continuous automated testing would take place each time the an addition or modification to the code is done.

### 11.3.2 Addition and updating of tests by the developers

- The developers can modify or update the tests according to their requirements, but then they would be required to run all the tests on the application again. This check need to be done to ensure that the application is working fine even after updating or adding new tests for the application.

### 11.3.3 Application under test remains stable:

- Whenever an application is modified or changed, quite a lot of effort is spent in maintaining or stabilizing the application if at all it breaks any functionality in the program which is highly likely. The Test Harness through its concept of supporting continuous integration after performing a set of automated tests keeps the system stable and under check.

### 11.3.4 Check if Application doesn't have any bugs accidentally introduced:

- There could be scenarios wherein a piece of code might have been accidentally deleted, in such cases it's very tedious to track the change and bug that has been introduced. This is where the Test Harness comes into the picture, as it runs all its tests on the application and the test logs would give us the exact details as to where the bug has been introduced or where exactly the functionality has been broken.

### 11.3.5 Tracking the bugs is so much easier:

- Since these tests would be executed regularly while testing the new code against the existing baseline. Due to the continuous integration in the application, we can keep a track of the bugs. For instance, if test harness ran the tests on the previous day and all tests passed; when it ran on the second day after new code was introduced again and only this time there were bugs reported. We could resolve this by pulling out the test logs and checking the details as to where the functionality broke.

Similarly, the Test Harness could be used to check if any old bug is not recurring in the application. Hence, through the test harness we can also undergo the tests on the application to check if previously recorded bugs are not present or reoccur in the system.

#### 11.4 Structure

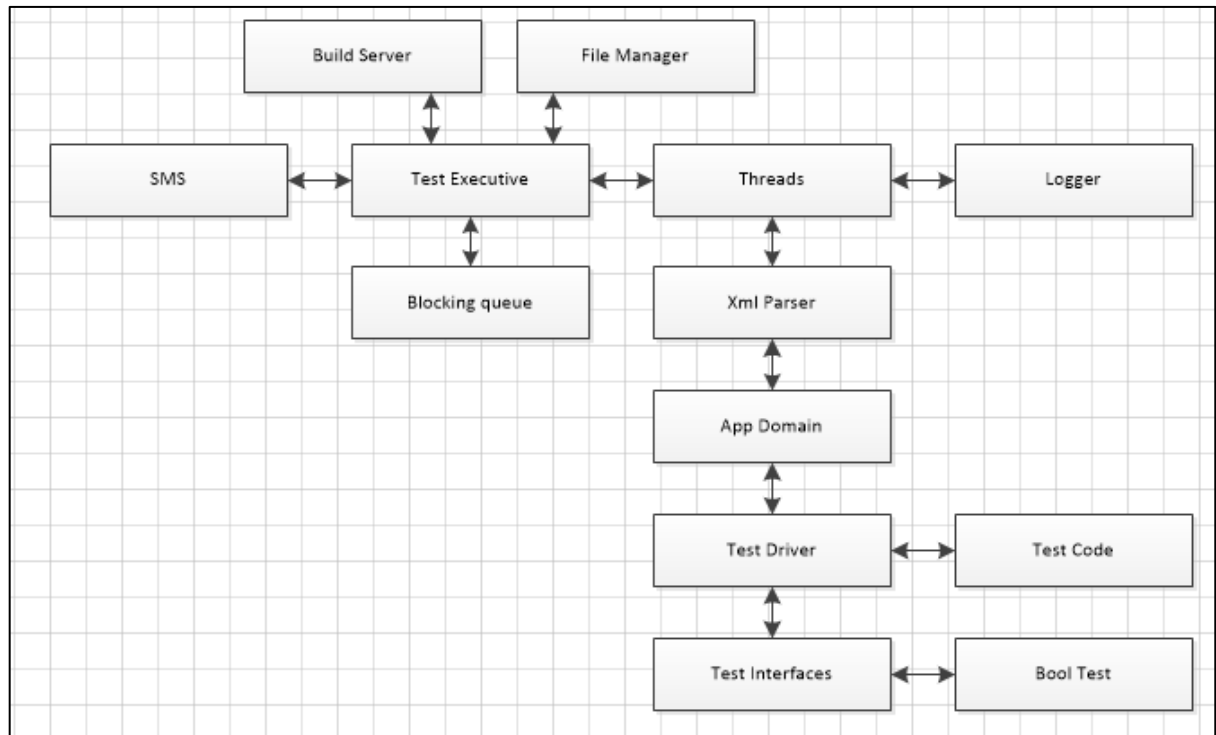


Figure 10: Test Harness Structure

- **Test Executive:** The text executive would be managing the Test Harness, basically kicking off the process. It would keep a track of all the processes basically from creating of a child app domain to passing the XML messages to the newly created child thread, which would then fetch the tester code and oversee the execution of the tests which would then send over the test results to the Client server. The communication between Client and Repository will be done using WCF communication.
- **App Domain Manager:** After the Client submits the test request (XML file). The XML file would be parsed by the child thread and would provide the details of loading which libraries out of which directory, followed by creating the test objects from the factory function and triggering the test. The App Domain Manager would have the role of creating child app domains; so that each and every test request can be executed in isolation. The App Domain manager basically has to manage app domains for different tests. The app domain manager would manage events like the load and shutdown events. For instance; in the latter case when an app domain dies due to an unhandled exception.

- **Child Thread:** The child thread would have the role of loading dynamic link libraries from the repository using the service; the child thread creates an instance of a callback object to get Child AppDomain results and logs. In the end, the child thread then sends the test results to the client and result logs to the repository.
- **Logger:** Based on the test results; the details of the test results executed will be logged and saved in the Repository server. The logger would have the responsibility of the displaying the error and failures in a different way.
- **Xml parser:** This Xml parser will parse the xml test request on the child thread and then the body of the Xml will be passed to the child AppDomain so that the child AppDomain knows which files to load for each test.
- **SMS:** The test harness will communicate with the repository and the client based through SMS.

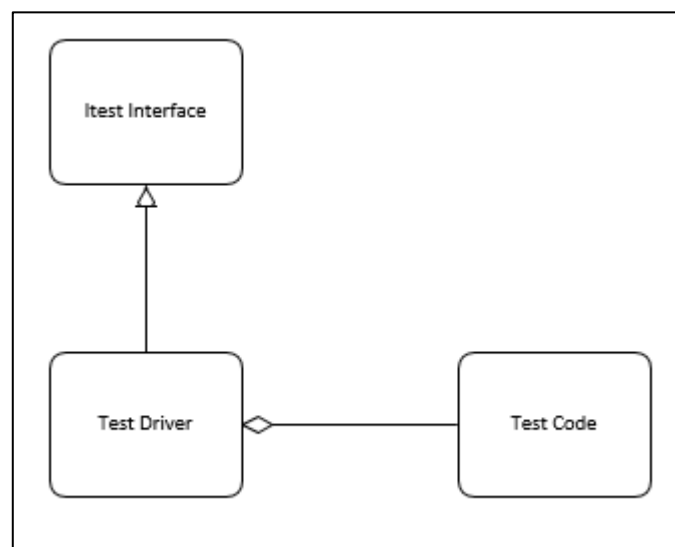


Figure 11: Test Driver

- **Test Driver:** The test driver executes the tests against the code you want to exercise and its dependencies. The test harness is going to load this driver with the test code and then run the driver. It consists of a factory function, which would create all the objects that are required to run the test. It has an ITest interface which contains the Bool test() function. It is an entry point for DLL loading. The test harness conspires to get that test function call in the test driver.
- **Blocking queue:** The basic concept of the blocking queue is to block enqueueing of items into the queue if the queue is full and similarly it blocks you when you try to dequeue items from an empty queue.

- **Xml Request:** It will name one or more of the test drivers it needs. Suppose it names 3 test drivers; the test harness runs them one by one and then logs the results. An XML file could be treated as a client's test request.
- **Test logs:** The test logs are stored in the Repository server. It can be retrieved using the `getlog()` function.
- **Build Server:** Explained in detail after test harness.

### 11.5 Interfaces:

The tester uses the ITest interface to start the testing process. The ITest Vector Generator is used to generate the test case input for the test class. The Test harness interacts with the Client and the repository.

- **Interface between Clients and the Test Harness:** When Clients want to check in code to the Repository Server, they will first invoke the Test Harness Server. Then the required libraries will be loaded in the Test Harness server to run the test cases on the files. The Server will post messages to the Clients, by using the method declared in the interface and according to the details mentioned in the Data Contract members. This would notify the clients whether the files met the requirements or not.
- **Interface between Repository server and Test Harness:** After the Repository Server supplies the required code to the Test Harness server; the tests will be run. And after testing, the test results are posted to the Repository Server, by using the method declared in the interface and Data Contract members and also sent to the clients, the logs are stored in the repository.

## 11.6 Activities

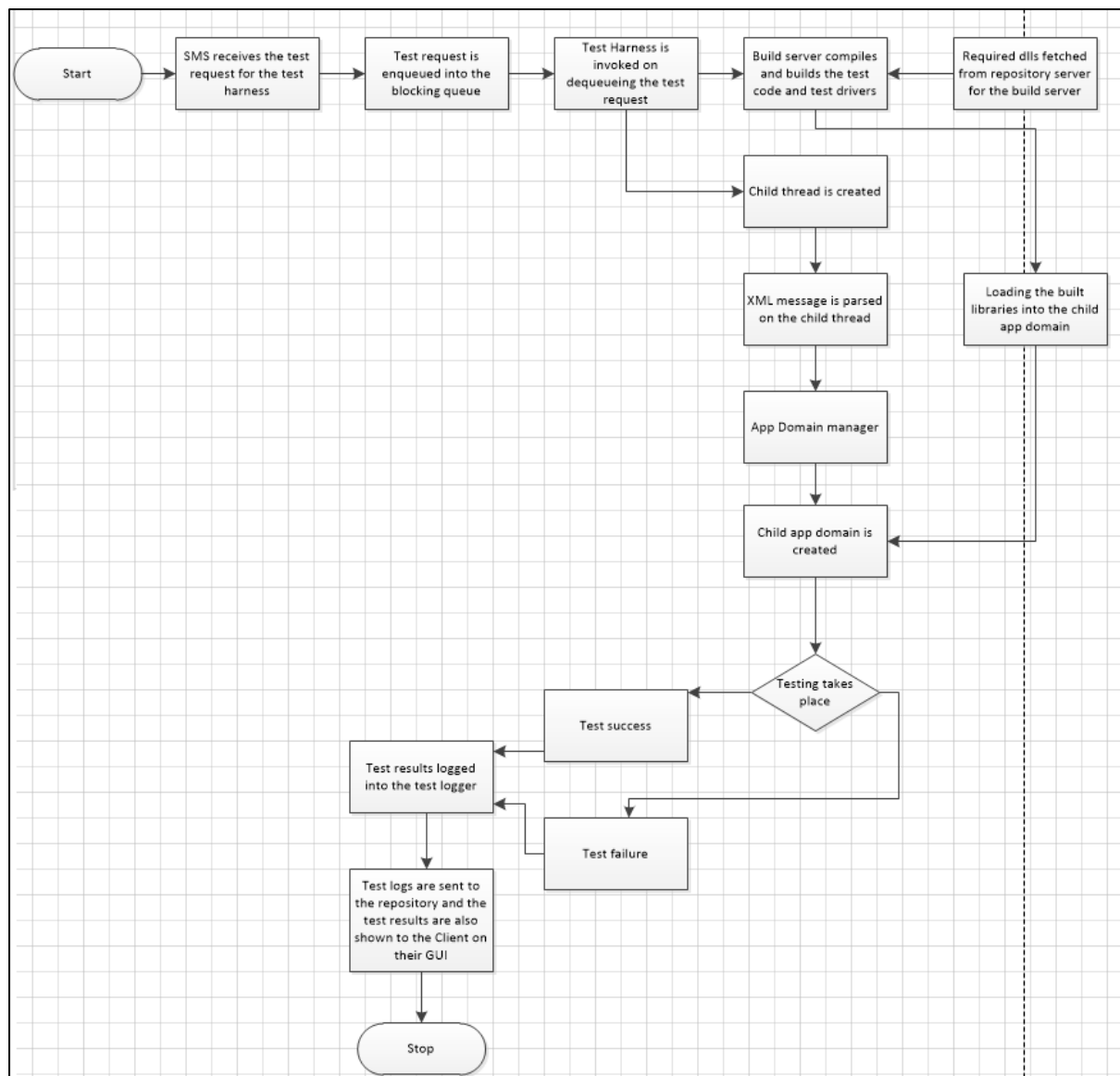


Figure 12: Activity Diagram for test harness

The above visio diagram is an activity flow diagram of how the test harness would function and below is a description of the above diagram:

- The client would be sending a test request which would be received by the SMS subsystem of the test harness..
- The XML messages which are actually test requests will be sent to the test harness via the blocking queue using the WCF service in SMS. These messages would contain the information of which test code and test drivers need to be loaded for testing.



- After the test harness dequeues the incoming test requests, it creates a thread. Unlike the previous project, in this project, the client and the test harness are not in a single process, hence there is a need for WCF channels for communication. Also, the dequeued requests would be picked up one by one by the child thread for testing. Only after one test request is done, will the next test request be picked up.
- Meanwhile, the required source code files and the dependent files are compiled and built into dynamic loaded libraries which would be later sent to the test harness for testing.
- On the child thread, the test request is parsed and then it creates a directory using the author and message time to make the directory name. Then the service will be used to download the required files from the build server. The child thread creates an instance of a callback object to get the Child AppDomain results and logs.
- The App Domain Manager will now create child app domains based on the number of test drivers specified in the xml. Each test driver along with its test code would have a separate child app domain. The reason being that child app domains are like isolated containers; and if for any reason a test throws an unhandled exception in one child app domain, it won't affect the neighbouring child domains since they are all independent.
- The child thread also passes the body of the test request so that the child AppDomain knows which files to load for each test. After all the tests are executed in the child AppDomain, the results are sent back to the child thread using its callback.
- Based on the test results, success and failure messages of the tests are recorded in the logs. The child thread then sends the test results to the client and the results and logs to the repository server. The logs would contain details like time and author stamped, identifying the test developer, code tested including version.
- Finally when the test is done, the next test request in the queue would be fetched by a child thread after a check for any more test requests in the queue is passed and the whole process would be repeated. If there are no more test requests in the queue, the blocked queue would block any dequeuing of items as it would be thrown in a deadlock state. Dequeuing would only be enabled once any test request is placed in the queue.

## 11.7 Critical Issues

Below discussed is a list of critical issues related to implementing a test harness along with their proposed solutions:

### 11.7.1 Ease of Use

- The easier we make this tool to be used, that many times complex it's going to be in set it up. The more features we add to make the test harness user friendly; the more we would have to work on the structure of the system, so as to incorporate the features that we think would make this tool as user friendly as possible.
- Hence we need to carefully go about designing the system, thinking about the various aspects on how to make the tool as user friendly as possible. On the same note, what features would the solution require to make the test harness that user friendly.

### 11.7.2 Performance related to test objects

- There could be a scenario where the solution may have almost 4000 packages for the Test Harness to test. This load could be a very daunting task for testing and would test the very limits of the test harness. Not to mention that the test would take a huge amount of time to be completed.
- Hence, the tests should take place on the test harness itself; check the maximum number of test cases it can run. Then provide more test cases to run and check the behaviour. This way we can examine and record it's capabilities during the breakpoint.

### 11.7.3 Demonstration

- To showcase that the system meets all the requirements. This is a critical issue in itself, as the developers and QA would provide a demo of the tool and would be checking each and every functionality of the tool. To make the process easier, the QA could have a document which has a list of all the tests or documents all the tests and the expected results that need to take place and then just cross check the logs for the test results.

### 11.7.4 Invalid XML file

- It could be a very common scenario where a developer may provide an invalid XML file, the test drivers and code mentioned in the file may be invalid or do not exist.
- There needs to be some input checks to avoid such a situation and preventing the system from crashing before the actual testing of the code takes place.

### 11.7.5 Appropriate access to the path of the dynamic link libraries

- Sometimes the location of the repository which is a folder in our project would require the right admin access to access the files. Suppose we have successfully tested the application in one platform using the test harness. But when we try the same on another platform, it threw a failure; on checking the logs we realise that it was an access denied issue for the files in the repository since we did not have the right path or admin access to the files located in the Repository folder located in the different platform.
- The solution is that we access the folder with administration rights in our code. This way whichever platform the test harness is running there won't be an issue.

### 11.7.6 Huge log file size

- There could be a scenario where there are a lot of tests executed and the log file is huge. In this case it would be hard to view the file due to its huge size.
- We can code a solution to fragment the log file after a certain size, this way all the log files will be of the same size and we wouldn't face any issues in accessing the logs due to their huge size.

## 12 Collaboration Server

The collaboration server provides tools for users to collaborate with each other, view reports and other collaboration related activity. It maintains data related to managing the project, it provides access to different collaboration resources like chats, video conferencing and tools that are crucial components for software development.

### 12.1 Organizing principles

#### 12.1.1 Tools need to be user friendly

The tools need to be fault intolerant and available on demand to the users, availability of any tool is a key feature.

#### 12.1.2 Access control

The tools related to project management should only be confined to the higher management to access. This way not just any user can change the task delegation activities or change the project estimation.

### 12.1.3 Project Management

The server makes it easy for the project manager and team leads to delegate tasks and looking at the progress of the packages in development, they should be able to set estimations to the tasks that are delegated. This way the collaboration server makes it easier for the higher management to assign and schedule tasks.

## 12.2 Uses and Users

### 12.2.1 Developers

- The developer can use the collaboration server to check his own individual tasks and timelines. The developer could also be assigned any errors that need to be fixed related to his module; the developer could also communicate with the other development teams located in remote locations through the virtual display system and hence collaborate their tasks.

### 12.2.2 Technology Architect (Tech Arch)

- The tech arch could communicate with the remote development teams and discuss any changes or clarifications regarding the software design.
- The tech arch can use the server to assign roles, responsibilities, tasks to the team members.

### 12.2.3 Project Manager

- The project manager would have the same level of permissions as the tech arch.
- He can create tasks, prioritize the tasks based on any requirement changes and also track the progress of the project.

### 12.3 Structure

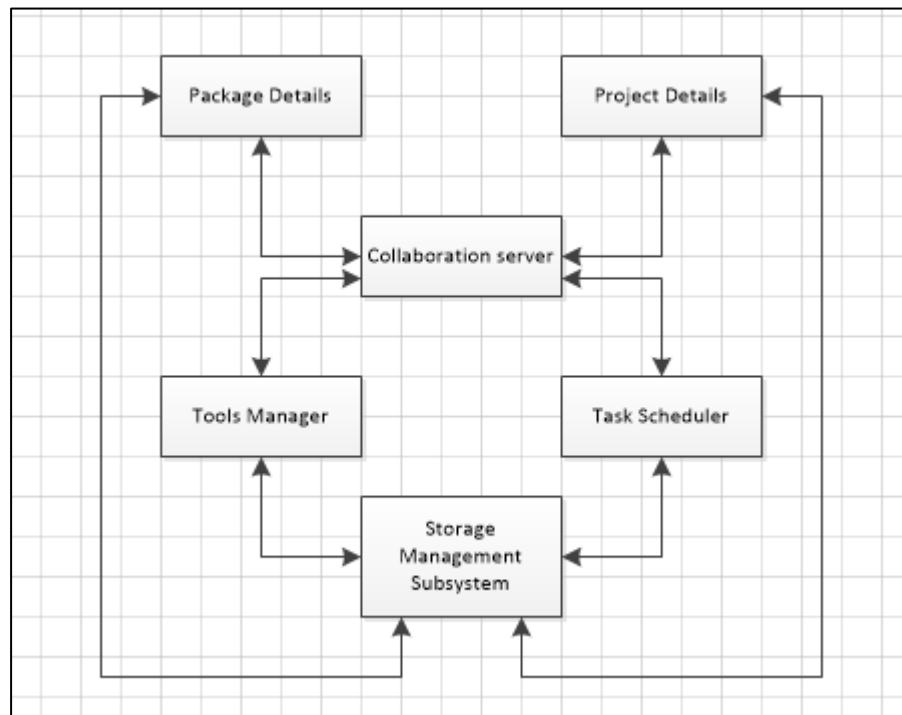


Figure 13: Structure of Collaboration server

The above figure gives us an idea of the Collaboration server architecture; it would contain the following components:

- **Collaboration Manager:** It will handle and monitor the activities performed by the other server packages.
- **Package details:** It would have the information related to all the work packages and access to them.
- **Project Details:** It provides details related to the progress of the software project development. It stores details related to the team's tasks, completion rate, milestones achieved and tasks left open or closed.
- **Tools Manager:** It will enable access to the various tools that would be provided through the visual display system.
- **Task scheduler:** This can create project related schedules like team meeting and weekly status update alerts.

### 12.4 Responsibilities

- The progress related to tasks or the project as a whole can be viewed. The Milestones can be set and notifications can also be set up to alert on completion of the milestones.

- It provides collaboration tools and services based on the user group the user belongs to. At first, the user would have to login, followed by which the user is provided a restricted access to the tools available based on the user group their user details belong to.
- The users with the right access can modify the user roles for access, they can add/ delete members to/from the user groups available or even change the permissions related to some user groups if required.
- It allows users like the project manager or team leads to create and assign tasks to the team members, those even located in a remote location would receive tasks and would also receive notifications related to the tasks assigned to them along with the progress of the task on a regular basis.

## 12.5 Activities

Below mentioned are the activities performed by the Collaboration Server:

- The collaboration server receives the message through the SMS, through the SMS the message is analysed and based on the events that need to be handled in the message; the request is routed to the concerned package.
- If the request is for the project details, then a request for the project status is sent to the repository, the status of the project and other project related details are generated into a report that would be sent over to the client.
- The package details would guide the team leads and the manager in allocating resources to the tasks in hand.
- If the request is for tools required for collaboration, the collaboration server will display the tools to the client through the virtual display system client.
- If the request is for the task scheduler, display an interface where the manager/team leads can schedule jobs.

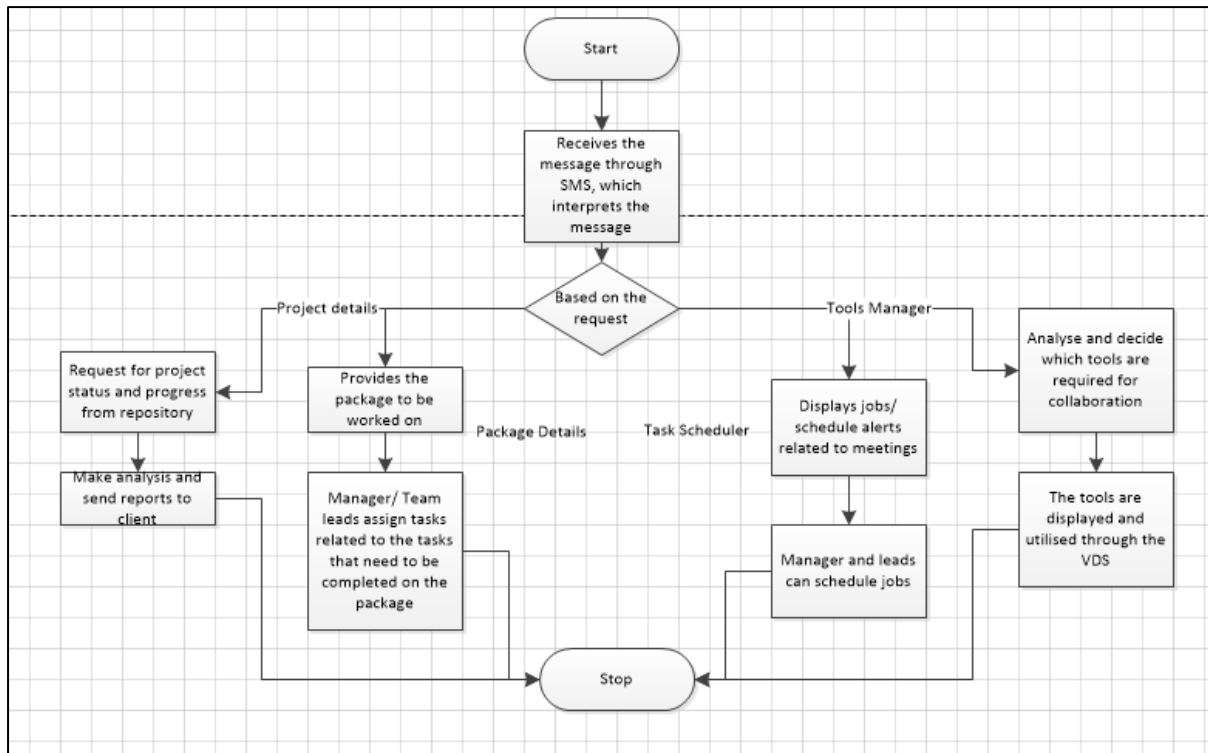


Figure 14: Activity diagram of Collaboration server

## 12.6 Critical Issues

Below discussed is a list of critical issues related to implementing a Client along with their proposed solutions:

### 12.6.1 Data Security

- The collaboration server is a very crucial component of SCF as it contains very important project management related information. If the data gets corrupt or lost, it might be an abrupt stop for the project.
- This can be resolved by making the data available in this server available to a selected few user roles. Also the data backup should be taken whenever the documents are updated so that in the situation of a corrupt system, we have disaster recovery measures in the form of back up data.

### 12.6.2 Connection drop

- The collaboration server could suffer a connection drop due to either tool crashing or a network connectivity issue.

- This can be resolved by having an offline feature available for the tool, so when the connectivity is back, normal operation resumes.

### 12.6.3 Task overlap

- The manager might overlap a task to a team member, unaware of a task already assigned to the team member through the team lead.
- This can be resolved by having a notification set up during overlapping. For instance, when the manager assigns the task like in the above scenario, the manager should receive a notification stating that a task has already been assigned to the team member; also the details of who has assigned the task can be shared with the manager.

## 13 Messages

For the components of the software collaboration federation (SCF) to collaborate, there needs to be a message passing communication to provide a reliant software development environment. Messages would hold the address information that gives us an idea of where the message has been sent from and where it needs to be delivered to, the data that needs to be delivered will be in the message body. Before sending any message, it's serialized and sent over the network. And the receiver will deserialize the received message before accessing it.

### 13.1 Organizing principles

#### 13.1.1 Message Structure

For deserialization purposes at the receiver's end, the message structure needs to be consistent. The structure could be defined using an XML template which any sender would need to follow.

#### 13.1.2 Sender and Receiver addresses

By including both the sender's and receiver's addresses in the message, the receiver would easily know who was the sender and to who the receiver needs to send a response to.

#### 13.1.3 Metadata related information

The message needs to contain details regarding where it originated from, what does it hold, when was it created and where is it heading to. These metadata related information needs to be included in the message.



## 13.2 Uses and Users

### 13.2.1 Comm Service

- The comm service which would be implemented by the SMS to communicate between different components of the Software Collaboration Federation (SCF) will utilize the messages for communication. The communication channel will use the source and destination addresses to deliver the messages between the different components of the SCF.

## 13.3 Types of messages

### 13.3.1 Repository-Test Harness messages

- **Request for getting the source code to be built and tested:** The message should include the file name to be tested as well as the dependencies should be listed out in the message, so that the test harness would know what packages are required to perform the test.
- **Test results message from test harness:** The message should include the file name that was tested as well as the dependencies should be listed out. The test harness will attach the test results in the body of the message or even list out the log file that was sent to the repository server.

### 13.3.2 Client-Test Harness messages

- **Test Requests message:** The message should include the file names to be tested as well as the dependencies should be listed out in the message, so that the test harness would know what packages are required from the repository server.
- **Request for test results from test harness:** The message should include the file name that was tested. The test harness will attach the latest test results executed for that file and it will be displayed to the clients on their GUI.

### 13.3.3 Client-Repository messages

- **Check-in message:** The message should include the file name that needs to be checked in and also the check in comments need to be provided regarding any changes made to the existing file or a check in of a new file.
- **Check-in status message:** The message should be returned to the client stating the status of the check in of the file.

### 13.3.4 Client-Collaboration messages

- **Documents request message:** The message should include the document name that the client has requested for access.

## 13.4 Critical Issues

Below discussed is a list of critical issues related to Messages along with their proposed solutions:

### 13.4.1 Incorrect format

- The message may not get recognized by a receiver due to invalid format of the message; this may result in loss of data or a required event not getting handled.
- This can be handled through the messages following a common structure so that the receiver knows well beforehand what to expect and it's easier for the receiver also to deserialize it and access the message.

### 13.4.2 Secure transmission

- There may be scenarios where the intruder gets hold of a message that contains critical information intended for another receiver, the intruder could change addresses or modify the message which could harm the system or introduce bugs.
- This message passing channel should be a secure channel, or we could even encrypt the message during transmission and decrypt the same at the receiver for it to access the message.

## 14 Comm Service

The components in the SCF application communicate with each other through communication channels. The service has a sender and receiver channel. The receiver consists of a host which would host the web service and a receiver blocking queue from where the requests will be handled. The sender also consists of a notification manager, proxy list and a sending blocking queue which would send messages to the receiver. As discussed earlier, since we would be integrating the SMS into the Software Collaboration Federation, this service would be moved from the component to the SMS subsystems of each of the components.

## 14.1 Organizing principles

### 14.1.1 Proxy list

This list is maintained by the sender channel so that each time the sender wants to communicate with the same receiver; he need not create a new proxy instance. This would save a lot of time.

### 14.1.2 Queues

Both the sender as well as receiver channels have queues that handle the messages on a first come first serve basis and hence there won't be any issues related to concurrency.

## 14.2 Uses and Users

The comm service would have been used by every component of the software collaboration federation (SCF), but thanks to the storage management system (SMS) which would be a subsystem for each of the SCF components, we now would have only one user but that user would be a subsystem of each SCF component.

### 14.2.1 Storage Management Subsystem

- Since SMS consists of the data and event managers to transmit data between the SCF components, it would be the primary and only user of the comm service.
- The SMS would compose and send messages with the sender's and receiver's address over the communication channel for processing.

## 14.3 Activities

Below mentioned are the activities performed by the Comm Service:

- The sender communication channel receives the message from the components.
- The message gets enqueued into the sender blocking queue. Once the thread is free, it will dequeue the message from the sender queue and process it further.
- The received message will have all the details regarding who sent it and where should the message be delivered. Once, the message is dequeued, the receiver's details would be checked for in the proxy list, if found then it would use the existing one. If not found, then it would be created.
- After the proxy is available, the message can be posted by the sender to the receiver through the WCF call. A notification manager on the sender's side also publishes notifications when events occur at the sender's end

- The web service host on the receiver will receive the sender's message and it will be deserialized into a C# object by the web service. The receiver will also be using a blocking queue in order to manage request messages from multiple senders. The received message which is pushed into the queue is dequeued, after which the message is processed and a response can be sent to the sender as the message has the sender details.

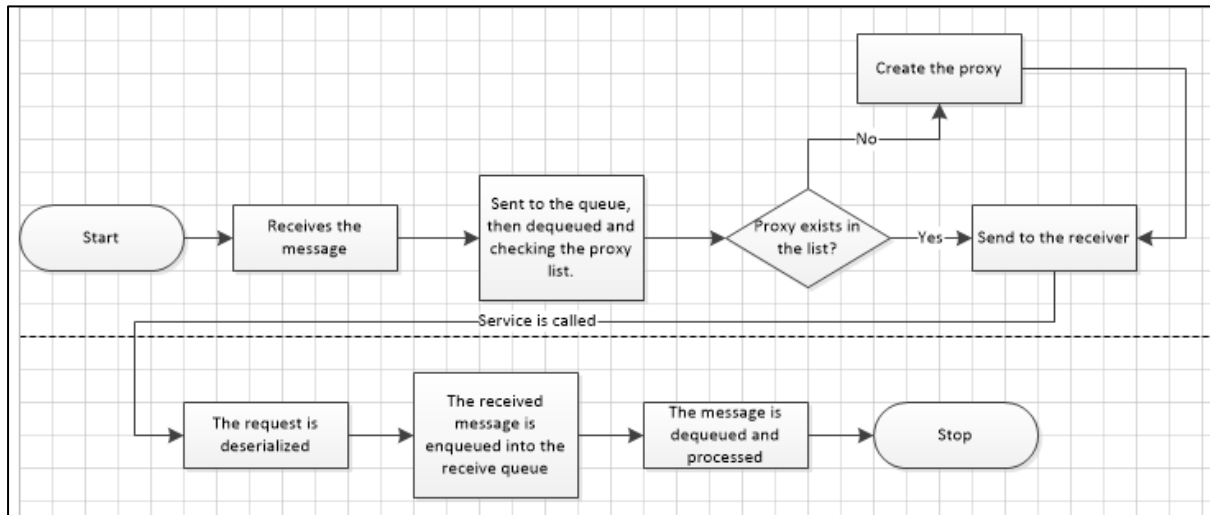


Figure 15: Activity diagram of Comm service

## 14.4 Critical Issues

Below discussed is a list of critical issues related to implementing a Client along with their proposed solutions:

### 14.4.1 Multiple messages at the receiver's end at the same time

- The receiver could be overloaded with too many messages from various senders.
- The receiver will maintain a blocking queue so that all the messages get enqueued in it, the messages are later dequeued and processed on a first come first serve basis and due to a single thread dequeuing the messages, no two messages will be processed in parallel and hence the concurrency issue is avoided.

### 14.4.2 Transmission of messages securely

- The client server data transfer needs to be encrypted so that the data is not visible to any intruder.
- We could use a public key cryptography protocol to send encrypted data over the network and then decrypt it at the receiver's end.

## 15 Core Services and Policies

### 15.1 Core Services

#### 15.1.1 Blocking queues

The blocking queue is a very useful tool that provides a thread safe implementation and avoids any deadlock situations. It helps maintain synchronization between the readers and writers of the queue, avoiding any deadlock scenarios. For instance, it might happen that a reader attempts to dequeue an item from the queue but the queue is already empty; this would lead to a deadlock. By using this tool, the reader is blocked from dequeuing unless a thread enqueues an item into it. This way the blocking queue serves on item at a time by dequeuing through one thread at a time.

#### 15.1.2 Managing metadata

Metadata is used to describe the content of the data files; the metadata query uses tag names for querying purposes. Using metadata for querying purposes improves the search performance since it contains useful results that improve the quality of search. The tag names normally used by the metadata are date modified, dependencies, file formats etc.

#### 15.1.3 Notifications

Notifications are the instant messages that are exchanged between components of the SCF. For instance, when the end user accesses the repository server through the client by entering the user credentials, based on the credentials being authorized, a notification will be sent back to the end user. For example, if login has passed, the end user would receive a notification stating successfully logged in. When there is any update related to the versioning of the file or an update in the code, the end user will be notified of the same at the client as well as the build server which would have to build the updated code for testing purposes. The test harness sends test results as notifications to the client stating clearly how the tests have either passed or failed.

#### 15.1.4 WCF Communication

Communication in SCF is based on the services provided by WCF and its implemented through the SMS. The endpoints would be defined for each component of the SCF through SMS. The bindings that would normally be preferred would be a one with security features as well as network interoperability; hence we could prefer the WSHttpBinding which fulfils this criterion. Basic Httpbinding is fast but devoid of default security features, NetTcpBinding on the other hand has no network interoperability. Hence our best option would be WSHttpBinding.

## 15.2 Policies

### 15.2.1 Ownership Transfer policy

There might be a scenario a developer might be on a leave, and since each developer has ownership of his respective package. There is a facility for ownership transfer or else no one could continue development on the package and the project would be stalled. Therefore, the developer needs to transfer his ownership of the file to a new developer, who can complete the remaining work. For the transfer to be successful an approval is required from the project manager or team leads. In total, the transfer would require two approvals; one from the higher management and another from the new developer accepting the transfer.

### 15.2.2 Single Ownership policy

Users can extract the file, but they have no authority to check-in. Hence, there is no need for check-out in single owner policy because only the owner is responsible for all the changes made to the file. When the single owner checks-out the file, till the time he checks-in the file, all the references to that file point to the version of file before the user checked out. When the single owner of the file finally checks-in the version gets updated. With regard to ownership transfer, the transfer can be done only for one user here since it has only a single owner; multiple users cannot be transferred the ownership by the developer.

### 15.2.3 Group Ownership policy

Users other than the members of the owner group can extract the file to their local machine, but have no authority to check-in that file. Concurrent check-outs are not allowed, when a member of the group accesses a file, the file gets locked thereby prohibiting further access of the file to other users in the group. So, no other group member can check-out that item until the member holding lock checks in the file. When a member of the group accesses a file owned by the group, notifications can be sent to other owners that the file is accessed. With regard to ownership transfer, the transfer can be done only to multiple users but the other developers in the group will be notified of the changes in ownership.

### 15.2.4 Caching policy

Caching needs to be enabled since there will be repeated requests to the server for particular information. The addresses of the servers to which the clients connect to can be cached on the client for faster connection in the future with the servers. The user details of the end user will have certain amount of user permissions and user role associated with it, these details can be cached on the

client so that the client can interact with any component of the SCF when it wants to. We can cache files which accessed from the server on the client side, this way the client doesn't have to approach the server each time for a file request. But the client will also be notified if a new version of the file is available has been checked in to the repository, in that case the old cache regarding the file details has to be replaced with the new details and it needs to approach the server for this.

### 15.2.5 Versioning policy

Versioning keeps track of modifications to a file and helps in the scenario of a rollback. All details related to the versioning are stored in the metadata. Every valid check-in to the code repository creates a new version of the file on the repository and updates the associated metadata files. Any dependencies which were relying on the past version of the file, remain attached to the past version. A valid check-in creates a version of the file and notifies the concerned user roles associated with that file. Users can then select whether they want to update their dependent files to refer to the new version of the file. This creates a new version of the associated metadata file and updates dependency metadata file versions.

The repository server enables users to extract all past versions of the file and rollback the system back to working conditions. This is done with the help of the metadata file

## 16 Error handling

Error handling is a very big deal for any application to be user friendly and not shut down due to an unexpected exception. Unexpected errors would result in bad user experiences. For instance, suppose a connection cuts off between a client and a server during a file transfer, the file transferred is partial and possibly corrupted. Both the client and server would be unaware of the situation. This could lead to loss of data as it didn't reach the destination. This could either be resolved by checking the last message and notifications sent from the client and server respectively. The client and server both should be aware of the status of the current file transfer as it happens. Messages can be passed to and fro to determine file transfer completion. When the end user uploads a file, the server should return a message like "receiving file" and once a file has been completely transferred, a message like "file received" must be sent to the end user. In case, the end user doesn't receive the message, the user should try uploading the file once again and the server should delete the partially received file.

## 17 Administration and security

The communication between clients and the distributed servers of the SCF is facilitated by using WCF services. For the clients to access the servers of SCF, they would need to provide login credentials for authentication. Based on the user roles of the credentials provided, the user is provided with the services of the server. If the user is invalid or not present in the list of users authorized to access the system, a notification will be displayed “Invalid user” and it will redirect to the login page. The user details and the user groups they are related to are saved in an XML file. Similarly, the addresses of the servers the client requests are meant for are also saved in an XML file; this makes it easier to distribute servers as well as route client requests for the concerned server.

## 18 Baseline Management

The baseline management represents the agreed-upon, reviewed, and approved set of requirements committed to a specific product release. When stakeholders “sign off” on requirements, what they’re really doing is agreeing and committing to a specific requirements baseline. The project team establishes a requirements baseline; the team should follow a pragmatic change control process to make good business and technical decisions about adding newly requested functionality and altering or deleting existing requirements. Change control is not about stifling change. It’s about providing decision makers with the information that will let them make timely and appropriate decisions to modify the planned functionality. That planned functionality is the baseline.

### 18.1 Responsibilities

- It should be a reference for future releases.
- It keeps track of all the deliverables along with their status.
- It marks the milestones in progress in the software development lifecycle.
- The baseline will be the basis for change control in the next releases.

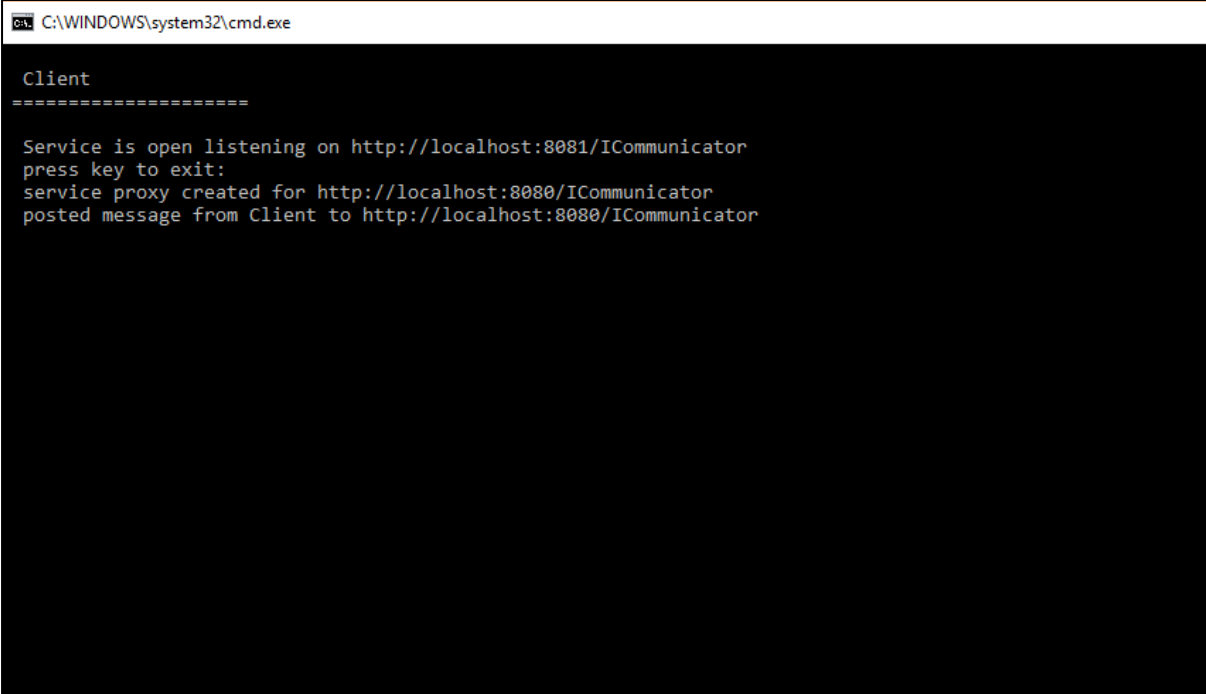


## 19 Appendix

### 19.1 Message Encryption and Decryption Prototype (Critical issue prototype)

The prototype shows how the message body of the message sent from the client to server is encrypted; this way no intruder can access the information related to the message passed. Once the server receives the message, the message body is decrypted and normal operation related to the message resumes. This prototype is related to the client sending a test request in an xml format to the server. The message is encrypted when sent, on decryption, the message details containing the test request is displayed on the server console.

Client console:



```
C:\WINDOWS\system32\cmd.exe

Client
=====

Service is open listening on http://localhost:8081/ICommunicator
press key to exit:
service proxy created for http://localhost:8080/ICommunicator
posted message from Client to http://localhost:8080/ICommunicator
```

Encrypted message sent to the server:

```

C:\WINDOWS\system32\cmd.exe

Server
=====

Service is open listening on http://localhost:8080/ICommunicator
press key to exit:
Server received encrypted message:
formatted message:
  type: default
  to: http://localhost:8080/ICommunicator
  from: http://localhost:8081/ICommunicator
  author: WbVnVcf7EL4CyxJkKtEiSA==
  time: 07-12-2016 18:29:16
  body:
7esLoB1g++Qi9jB0+hypwhRyPL45j2dBFDeqoch4G3bmrS1YyVVCHmB+Ysb9bh0197dKHSwGQLLoRwsB/NrE/TRo33frghZ09+SFUw1SgYqB8LHpM0917HMa
uw3cz229pY0GTkkVTmSK3Xw3G+P36FHTCINorARCJMx+Y5reTJOPiRpf17tXxE3Wr/loK3y3GFzpnJ7Fvd1kxcbyqeyiqk6lJohwChaMMEpubruN7FbroCtW
k7vbBTP11LWzFOnho70UAXrPTno9GR0y9o4KrazVJdEpInu1vJw7Z324RN9o/QAwSMvBqRvcP18b37hdhMDq1059QY44Zo/IYybZuBQi5GUG0d1Dw5dW59Po
e5tZKTBlMrAUDY6nmWhMfAWBH3DpEoIFE3mS8qJ04pTdByYgSsH6+ay7xac6KqrdRwX71ghb4wFPeN6L1YM00Mxf0oymwm64ctepTKmoJnZuj9QP9/5sQWD
sejDw3Gsdbyom1/AjvtujAQg083DYBSZJ5x4pY6887ImB6zb+YjRn8DjpDiZM4eECLJihXGkRUWoPagTeB/SCbtQYaN8TiKPsfoFev6ndlqyamKw57ZG07iJ
Sh5Fo71ZFRgaX7A3iM3P+NgwS6m71Xuoe1EfVG0bn3+XJ72SbC5ng5WvnmKGI8qx5El0mW/CbgYFPNtdndNQ4H2QLnS1h70fsJInaj1GL2/ythS/6JQGBZV3
ocS0iYI6EIm2QjKC/0WLUj+UcAXTiBpdCYuMfaDbNikwKmp/

Decrypting the message body:

<?xml version="1.0" encoding="utf-16"?>
<TestRequest>
  <author>Karthik Bangera</author>
  <tests>
    <TestElement>
      <testName>test1</testName>

```

Complete decrypted message on the server:

```

C:\WINDOWS\system32\cmd.exe

Decrypting the message body:

<?xml version="1.0" encoding="utf-16"?>
<TestRequest>
  <author>Karthik Bangera</author>
  <tests>
    <TestElement>
      <testName>test1</testName>
      <testDriver>td1.dll</testDriver>
      <testCodes>
        <string>t1.dll</string>
        <string>t2.dll</string>
      </testCodes>
    </TestElement>
    <TestElement>
      <testName>test2</testName>
      <testDriver>td2.dll</testDriver>
      <testCodes>
        <string>tc3.dll</string>
        <string>tc4.dll</string>
      </testCodes>
    </TestElement>
  </tests>
</TestRequest>

```

## 19.2 Test Harness prototype

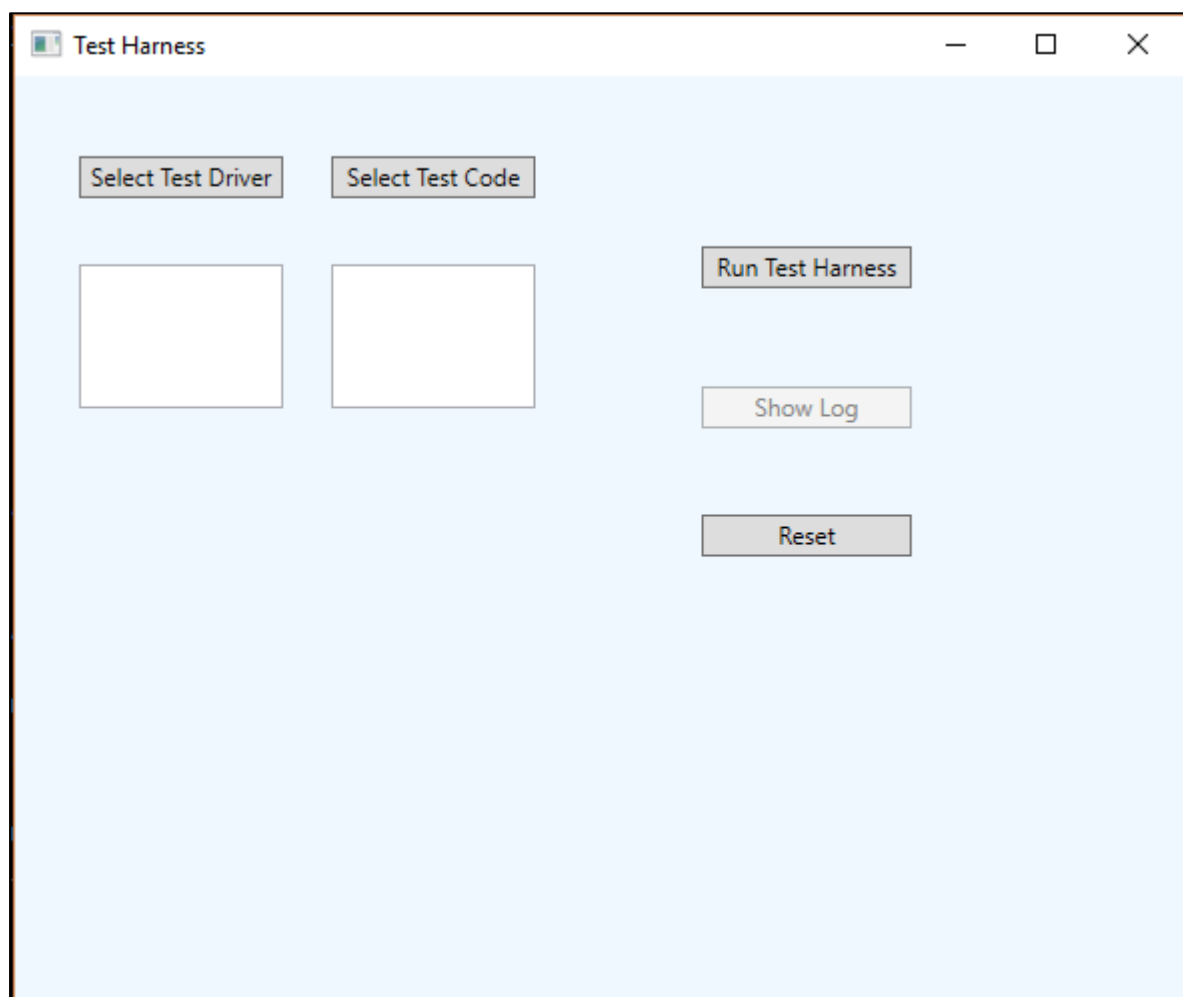
The test harness prototype showcases the functionality of the test harness, a client user selects the test code and test driver through the GUI shown below, the selected files are sent to the repository. At the same time a test request is sent over from the client to the test harness. The test harness performs the required tests with the DLLs of the test driver and test code; followed by which test result logs are created and stored in the repository and the results of the test are displayed to the client on the click of show log. The below guide describes on how to go about using the test harness prototype.

### Batch Files:

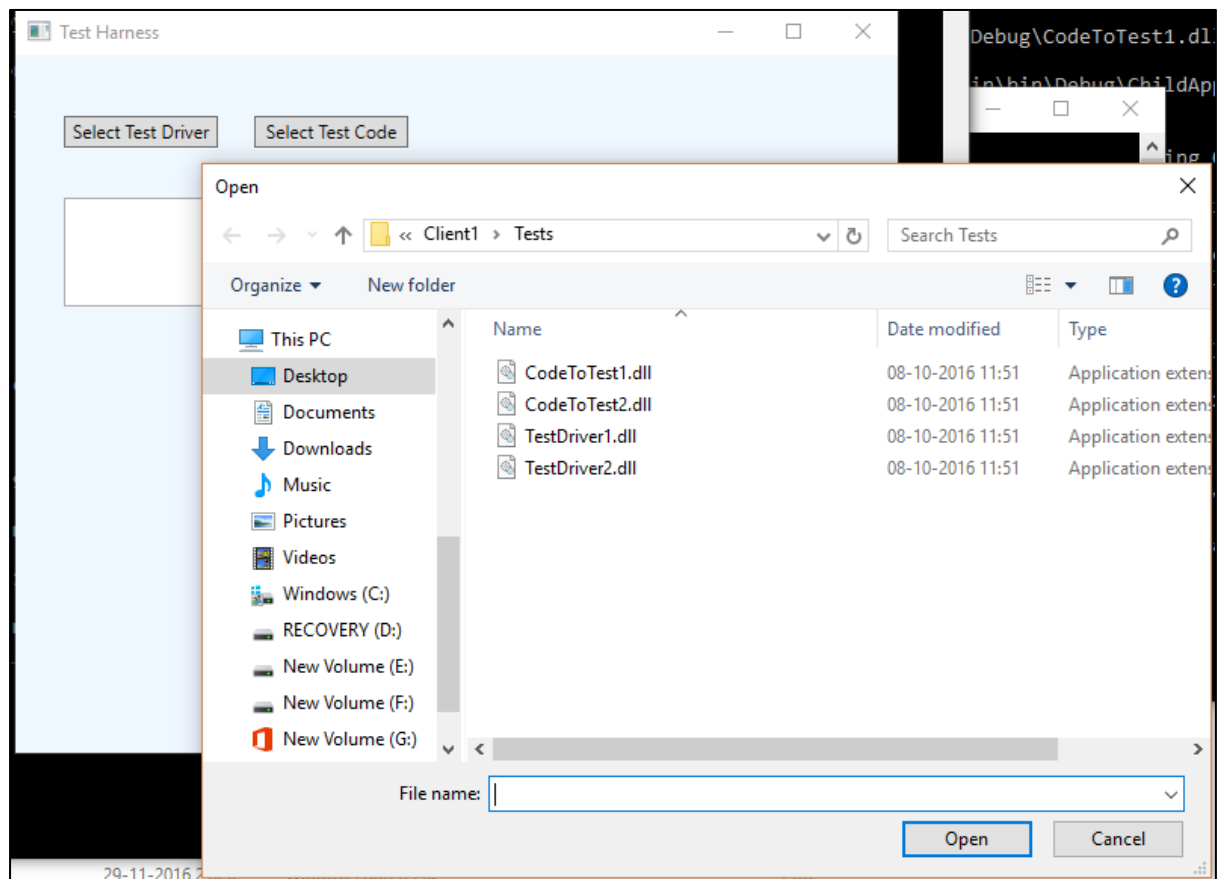
Compile.bat – Compiles all the packages.

Run.bat – Starts the test executive GUI, the repository, the client and the test harness servers.

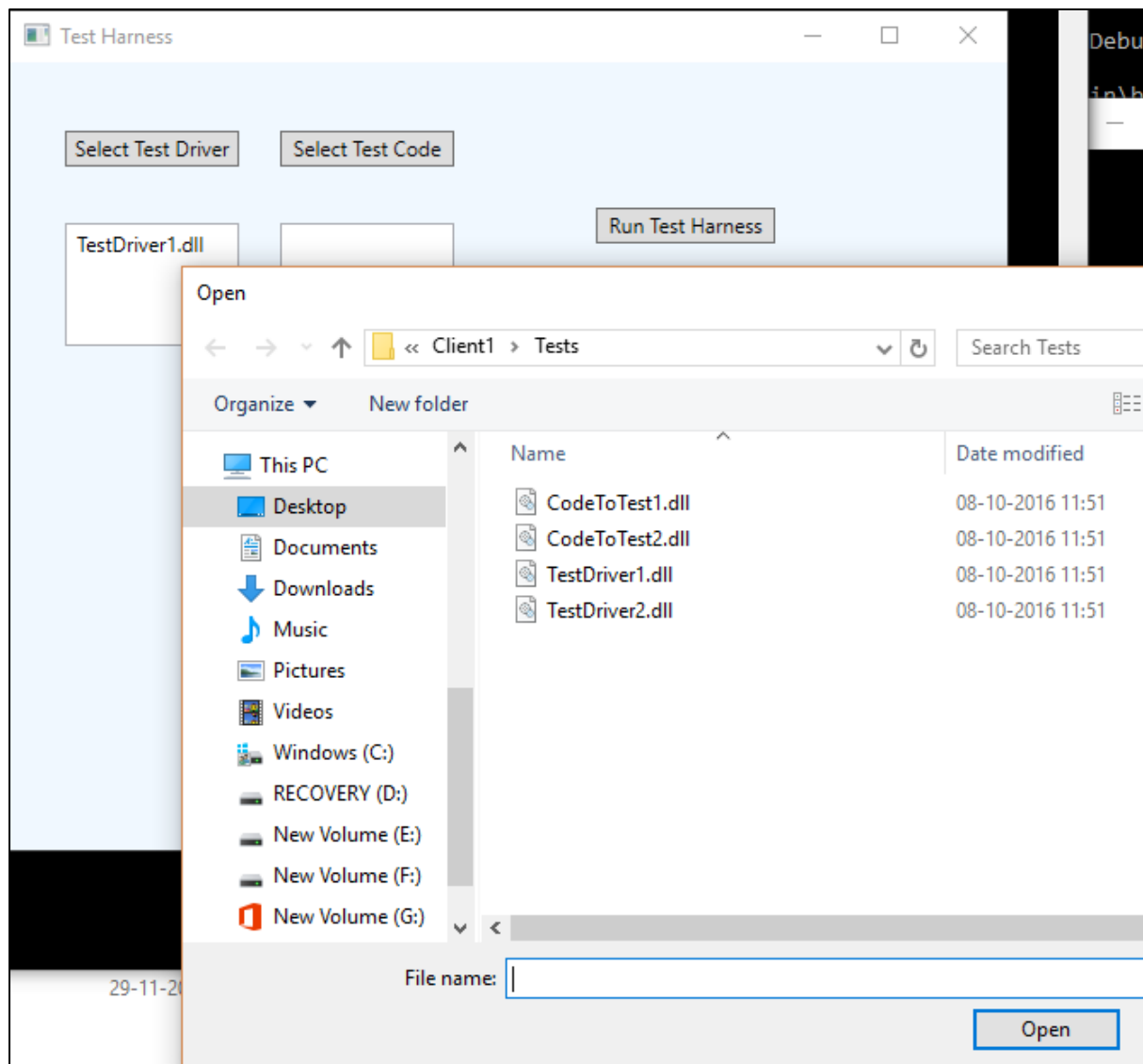
Screen1: GUI



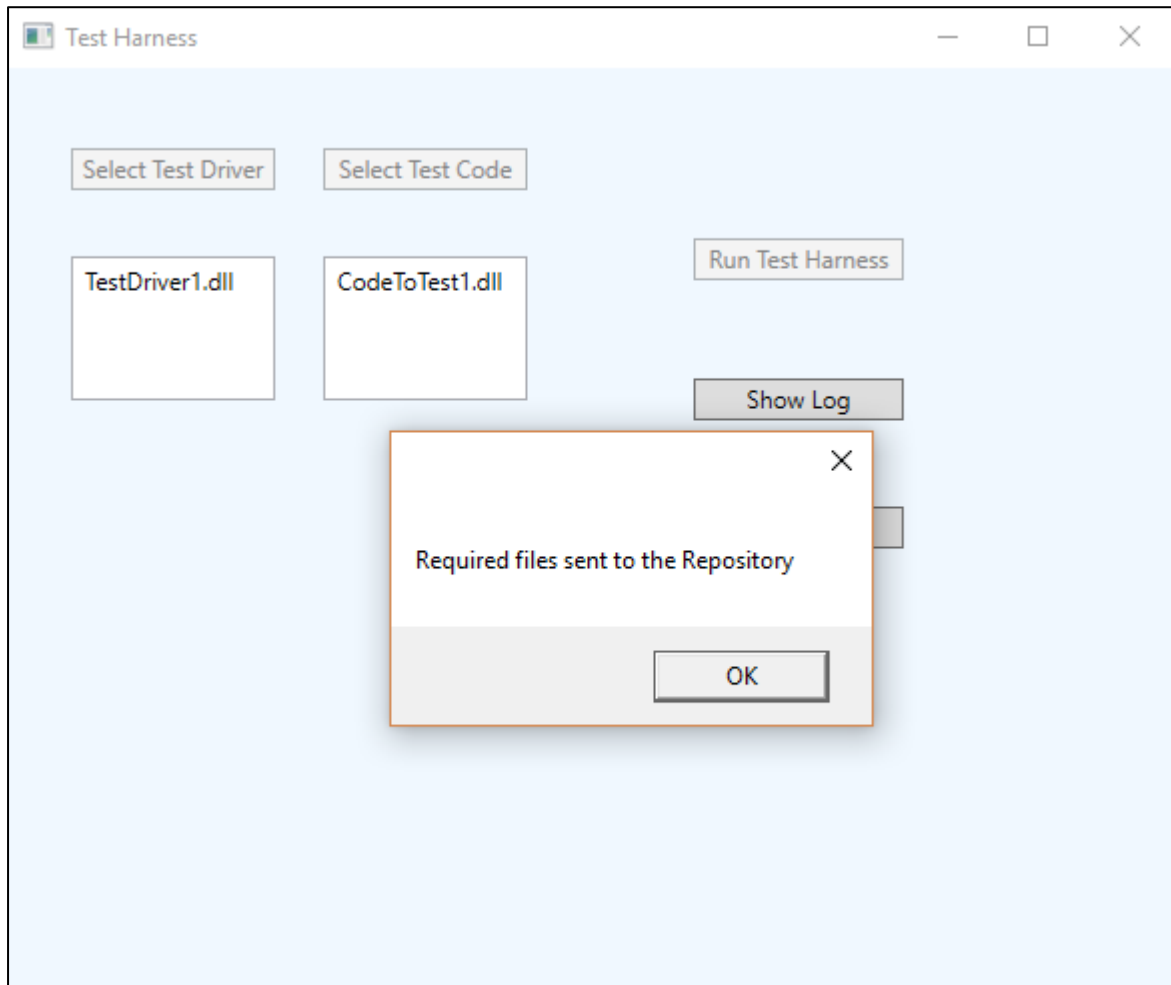
Screen 2: For test case 1, select TestDriver1.dll after clicking Select Test Driver



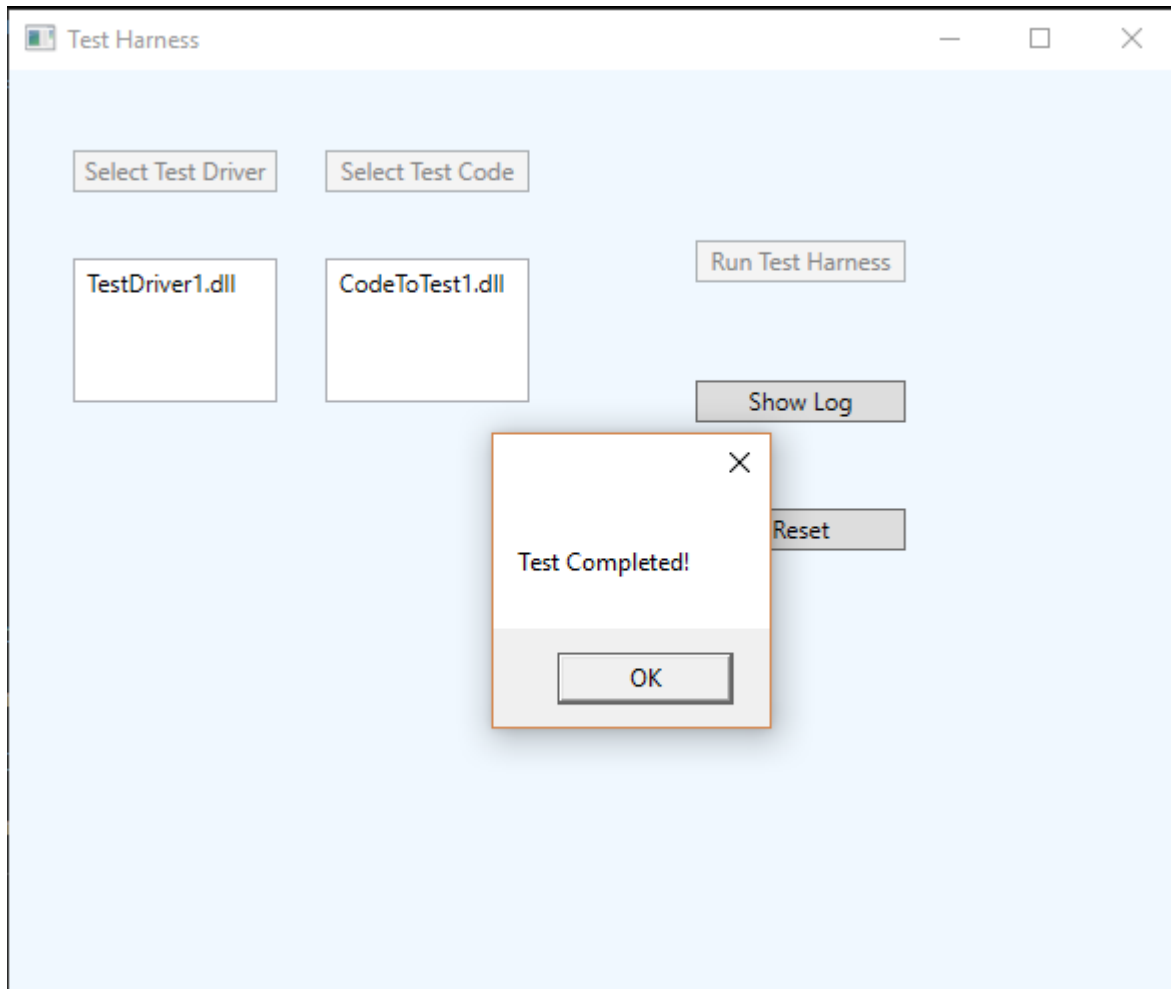
Screen 3: For test case 1, select CodeToTest1.dll after clicking Select Test Code.



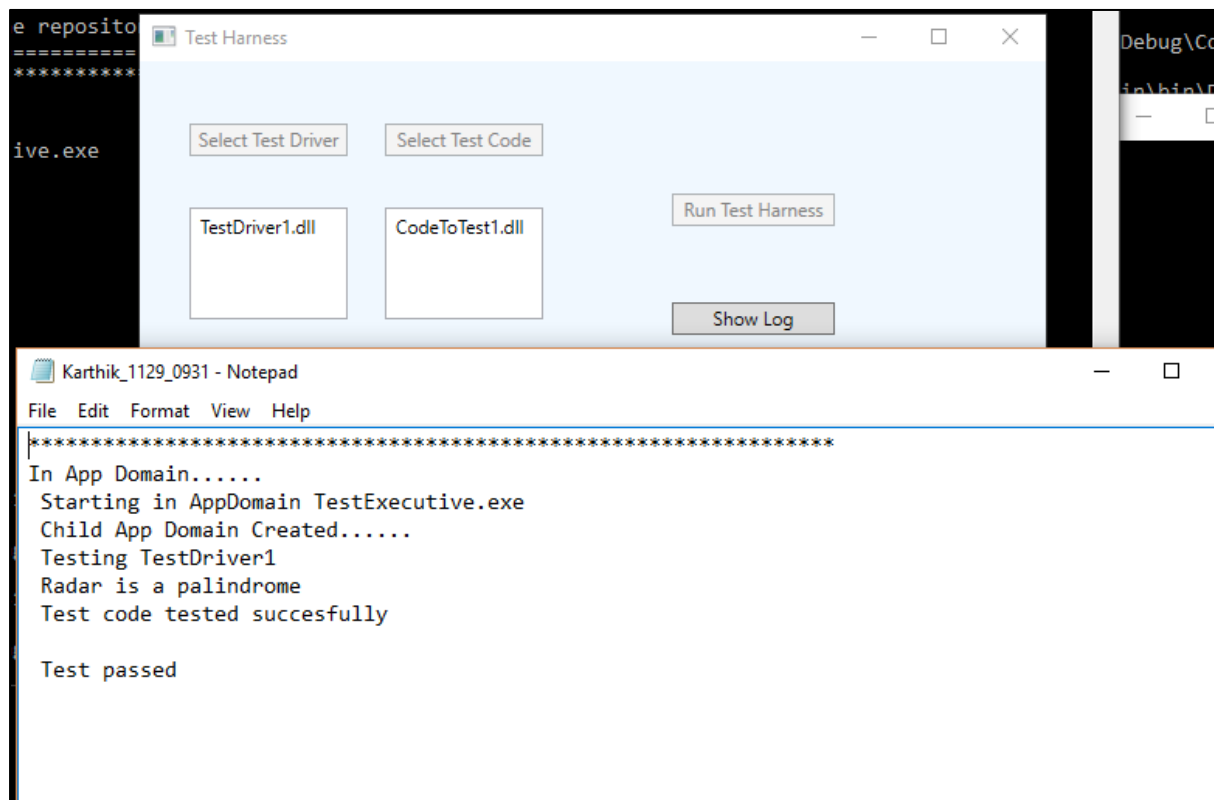
Screen 4: After clicking Run Test Harness, the test files would be sent to the Repository server and an xml test request would be sent to the Test Harness server.



Screen 5: The test harness gets the files from the repository and executes it, following which we get the below screen.

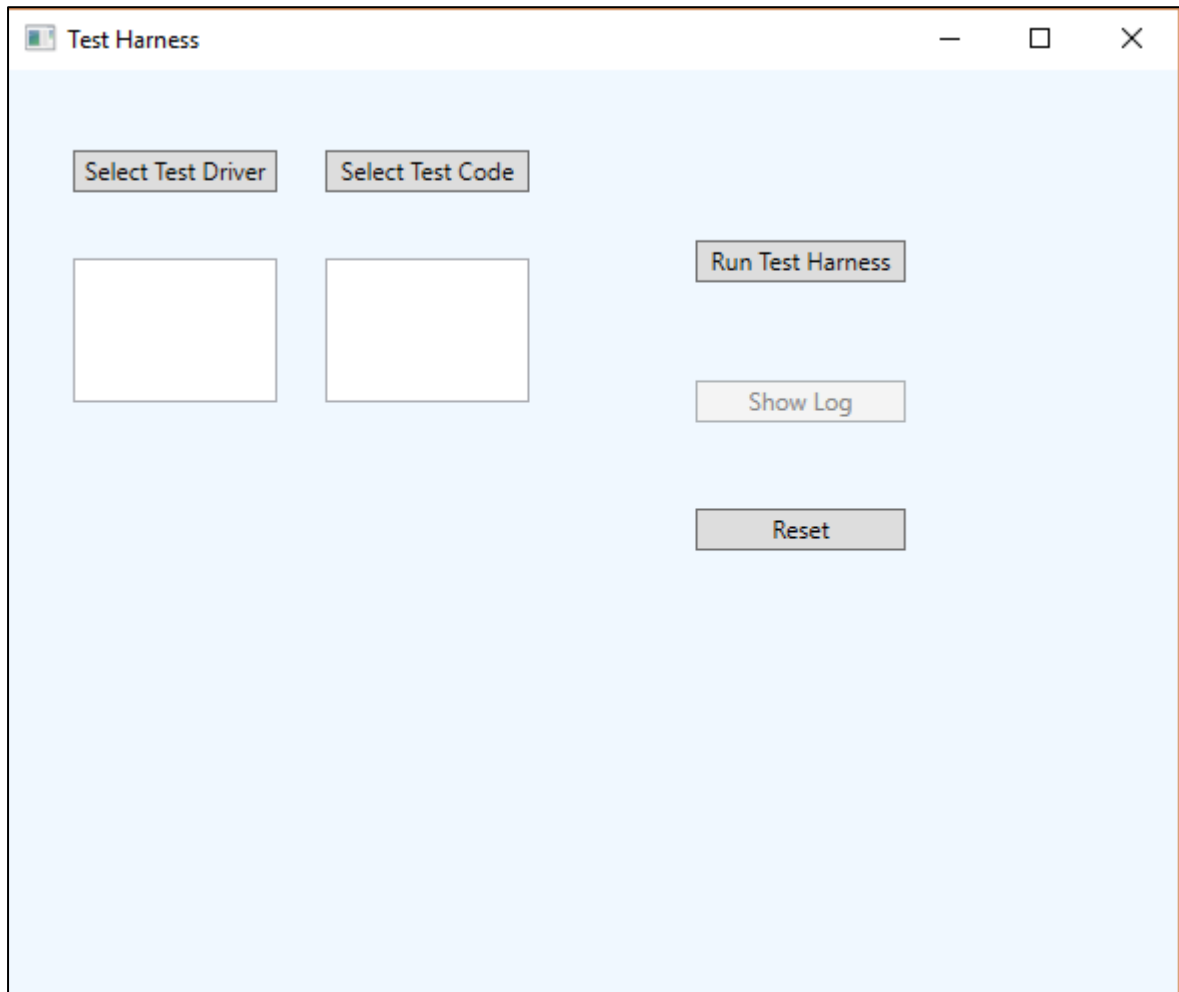


Screen 6: On click of Show Log, we get the log file of the executed test.

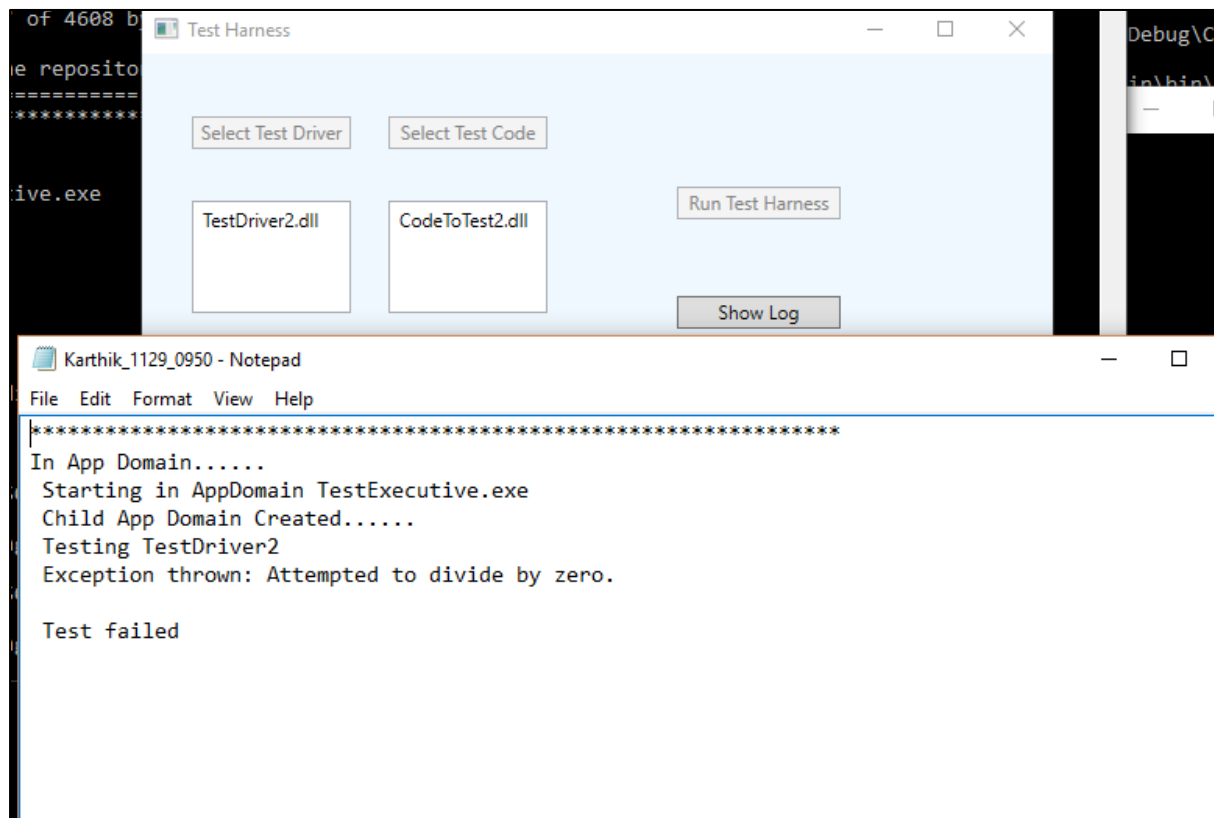




Screen 7: On click of Reset, we can go forward with the test case 2, by selecting the TestDriver2 and CodeToTest2 dlls respectively.



Screen 8: After test case 2 execution.



Screen 9: Console of the test harness server receiving the xml test request from the client server.

```
Starting Test Harness
=====

Service is open listening on http://localhost:8082/Communicator
press key to exit:
TestExecutive received message:
formatted message:
  type: default
  to: http://localhost:8082/Communicator
  from: http://localhost:8081/Communicator
  author: Karthik
  time: 29-11-2016 21:30:05
  body:
<?xml version="1.0" encoding="utf-16"?>
<TestRequest>
  <author>Karthik Bangera</author>
  <tests>
    <TestElement>
      <testName>test1</testName>
      <testDriver>TestDriver1.dll</testDriver>
      <testCodes>
        <string>CodeToTest1.dll</string>
      </testCodes>
    </TestElement>
  </tests>
</TestRequest>

  Tests to be run in the test harness
  =====

Test Name: test1
Test Driver required: TestDriver1.dll
Received file "TestDriver1.dll" of 5120 bytes in 19157 microsec.
Test Code required: CodeToTest1.dll
Received file "CodeToTest1.dll" of 4608 bytes in 6172 microsec.

Required files received from the repository
=====
*****
```

Screen 10: Extended console of test harness server showing the tests run.

```
Tests to be run in the test harness
=====

Test Name: test1
Test Driver required: TestDriver1.dll
Received file "TestDriver1.dll" of 5120 bytes in 19157 microsec.
Test Code required: CodeToTest1.dll
Received file "CodeToTest1.dll" of 4608 bytes in 6172 microsec.

Required files received from the repository
=====
*****
In App Domain.....

Starting in AppDomain TestExecutive.exe

Child App Domain Created.....
Loading:CodeToTest1.dll
Loading:TestDriver1.dll

Testing TestDriver1

Radar is a palindrome

Test code tested succesfully

Test passed

Uploaded file "Karthik_1129_0905.txt" in 1351 microsec.
total elapsed time for uploading = 149635 microsec.

Uploaded file "Karthik_1129_0905.txt" in 5167 microsec.
total elapsed time for uploading = 149635 microsec.
```

Screen 11: Similarly for test case 2.

```
TestExecutive received message:
formatted message:
  type: default
  to: http://localhost:8082/Communicator
  from: http://localhost:8081/Communicator
  author: Karthik
  time: 29-11-2016 21:30:30
  body:
<?xml version="1.0" encoding="utf-16"?>
<TestRequest>
  <author>Karthik Bangera</author>
  <tests>
    <TestElement>
      <testName>test2</testName>
      <testDriver>TestDriver2.dll</testDriver>
      <testCodes>
        <string>CodeToTest2.dll</string>
      </testCodes>
    </TestElement>
  </tests>
</TestRequest>
```

Screen 12: Extended test harness console of test case 2.

```
Tests to be run in the test harness
=====

Test Name: test2
Test Driver required: TestDriver2.dll
Received file "TestDriver2.dll" of 5120 bytes in 13788 microsec.
Test Code required: CodeToTest2.dll
Received file "CodeToTest2.dll" of 4608 bytes in 2544 microsec.

Required files received from the repository
=====
*****
In App Domain.....

Starting in AppDomain TestExecutive.exe

Child App Domain Created.....
Loading:CodeToTest2.dll
Loading:TestDriver2.dll

Testing TestDriver2

Exception thrown: Attempted to divide by zero.

Test failed

Uploaded file "Karthik_1129_0930.txt" in 763 microsec.
total elapsed time for uploading = 56655 microsec.

Uploaded file "Karthik_1129_0930.txt" in 1374 microsec.
total elapsed time for uploading = 56655 microsec.
```

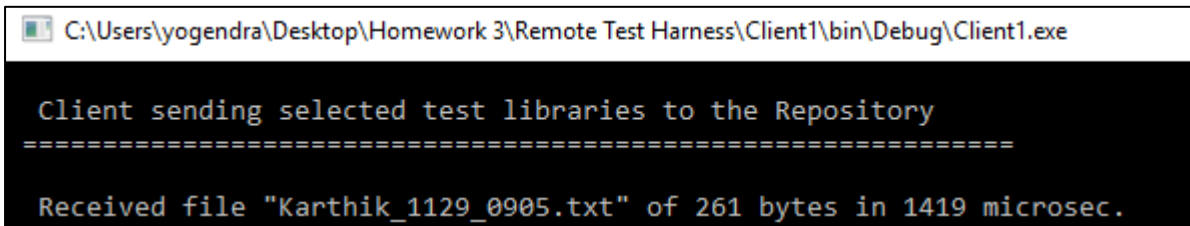
Screen 13: Repository console which receives the files at first from the Client server, sends it over to the Test Harness server on its request and then receives the log file from the test harness server

```
C:\Users\yogendra\Desktop\Homework 3\Remote Test Harness\Repository\bin\Debug\Repository.exe
Repository started
=====

Press key to terminate service:

Received file "CodeToTest1.dll" of 4608 bytes in 18911 microsec.
Received file "TestDriver1.dll" of 5120 bytes in 22090 microsec.
Sent "TestDriver1.dll" in 236 microsec.
Sent "CodeToTest1.dll" in 232 microsec.
Received file "Karthik_1129_0905.txt" of 261 bytes in 1596 microsec.
```

Screen 14: Client server console of receiving the log file from the Test Harness server



```
C:\Users\yogendra\Desktop\Homework 3\Remote Test Harness\Client1\bin\Debug\Client1.exe

Client sending selected test libraries to the Repository
=====
Received file "Karthik_1129_0905.txt" of 261 bytes in 1419 microsec.
```

## 20 Conclusion

- The software collaboration federation can be effectively used for continuous automated integration of developed source code with the baseline code. The document gives an insight on the design and the concept of the SCF and its uses for various users like the developers, QA personnel, project managers and team leads.
- We have discussed the activities, responsibilities, structure, organizing principles, critical issues and resolutions to those critical issues of each of the SCF servers.
- We have discussed the functionality of the Storage Management Subsystem (SMS), its uses and its users. SMS would provide the data management and communication services between the different components of the SCF. It will be available for each component of the SCF.
- The users have facilities such as notifications, metadata files to view information related to the files. It is a really effective tool meant for the purpose of large software projects with development teams setup in remote locations.
- The tool provides effective modes of communication through collaboration tools for teams located in remote locations and creates a perfect software development environment for the teams irrespective of their locations.



## 21 References

- <http://ecs.syr.edu/faculty/fawcett/handouts/>
- <https://social.msdn.microsoft.com/Forums/vstudio/en-US/d6a2836a-d587-4068-8630-94f4fb2a2aeb/encrypt-and-decrypt-a-string-in-c?forum=csharpgeneral>
- <http://tutorials.jenkov.com/java-concurrency/blocking-queues.html>
- <http://blog.mindjet.com/2012/10/3-workflow-benefits-of-collaboration-software/>
- [https://technet.microsoft.com/en-us/library/hh393566\(v=sql.110\).aspx](https://technet.microsoft.com/en-us/library/hh393566(v=sql.110).aspx)
- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/cse681codeL25.htm>
- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project5HelpF15/HelpForProject5.pdf>
- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project5HelpF15/>
- <http://www.jamasoftware.com/blog/defining-requirement-baseline/>