Ethan Crawford

Chief Technology Officer, SoftCell Consulting Inc.

Bellevue, WA 98007

March 12, 2008

Executive Team

Macrofirm Software

1 Macrofirm Way

Bothell, WA 98011

Dear Macrofirm Executive Team,

Enclosed is the update to the SoftCell Consulting Inc. project proposal for the Test Case Manager system discussed at our recent meeting, containing full requirements analysis, including use-cases and other models. Please review at your earliest convenience.

We recognize that bidding on the Test Case Manager system is competitive, and are excited to have the opportunity to present to the entire Macrofirm management team just some of the reasons why SoftCell leads the industry in award-winning designs, efficient, predictable implementations, and reliable, maintainable architectures.

To ensure that our business proposal matches your business needs, we have performed a thorough review of similar competing systems and have created a set of internal user interface mockups. During our followup meeting, we will be supplying these prototypes to demonstrate specific user scenarios.

Our research has indicated that an affordable, maintainable, enterprise-class system can be efficiently build using Microsoft SQL Server 2005 and .Net framework version 3. A team of two developers and a tester should be able to complete the project in around 12 weeks for a total cost of $104,200, beginning in the second week of March, 2008, and finishing on the first of June. This proposal outlines the requirements, user scenarios, and risks, and contains a formal cost-benefit analysis that demonstrates the financial viability of the solution.

We look forward to inaugurating a season of successful partnerships between our two organizations, and are eagerly anticipating your reply.

Sincerely,

Ethan Crawford

Chief Technology Officer

SoftCell Consulting Inc

Revision Statement, 3/10/2008

1. **General Style**
   1. Removed New Century Healthcare references
2. **Cover page**
   1. Added spaces in between ‘TestCaseManager’
3. **Executive summary**
   1. Made more specific regarding technology
   2. Added justifications as to why certain technology choices were made.
4. **Problem Statement**
   1. Expanded level of detail in requirements
5. **Solution:**
   1. Added detail on software
   2. Added justification for cluster
   3. Improved number consistency
6. **Cost/Benefit analysis**
   1. Improved number consistency
   2. Added units to benefits
   3. Removed hard-to-quantify customer satisfaction benefit
   4. Updated expenditures and payback analysis tables
   5. Updated ROI
7. **Risks:**
   1. Made risks more specific
8. **Summary**
   1. Improved number consistency
9. **Requirements**
   1. Added justification for cluster requirement
   2. Added more detail

|  |  |
| --- | --- |
|  |  |
|  | Ethan Crawford  SoftCell Consulting Inc.  CSS 370  February 11, 2008 |

|  |
| --- |
| **[Macrofirm Test Case Manager business proposal AND ANALYSIS OF REQUIREMENTS]** |
| The Macrofirm Test Case Manager is a state-of-the-art, enterprise-level system for managing test cases within the Quality Assurance division of Macrofirm. This document outlines the benefits, costs, and required feature set. |

Table of Contents

Table of Contents vi

Revision History viii

1. Executive Summary 1

2. Problem Statement 1

2.1 Goals 2

2.2 Terminology 2

2.3 Key Stakeholders 2

2.4 Summary of Requirements 2

3. Solution 3

3.1 Architecture 3

3.1.1 SQL Server 2005 and front-end 3

3.1.2 Visual SourceSafe 4

3.1.3 C# front.end 4

3.2 User Interfaces 4

3.3 Project Plan and Schedule 4

4. Cost and Benefits 5

4.1 Total Cost of Ownership 5

4.2 Systems and infrastructure 5

4.3 Development costs 5

4.3.1 Human Resources 5

4.3.2 Development Schedule 5

4.3.3 Salaries 5

4.3.4 Overhead 5

4.4 Operations 6

4.4.1 Systems Administration and Maintenance 6

4.4.2 Training 6

4.4.3 Technical support 6

5. Benefits 6

5.1 Increased Productivity 6

6. Formal Cost/Benefit Analysis 7

6.1 Tangible Costs: 7

6.2 Payback Analysis and Return on Investment (ROI) 8

7. Risk Management 8

7.1 Technology 8

7.2 Business 9

7.3 Organizational 9

7.4 Industry 9

7.5 Total Exposure and Impact 9

8. Summary of Recommendations 9

9. Appendix 1: Specifications 10

9.1 Definitions and Change Process 10

9.1.1 Priority 10

9.1.2 Change Process 10

9.2 Functional 10

9.2.1 Test Case Manipulation 10

9.2.2 Test Passes 11

9.2.3 Test Pass Reports 12

9.2.4 Test Case Revisions 12

9.2.5 Systems Administration 13

9.2.6 User Interfaces 14

9.3 Non-Functional 15

9.3.1 Performance 15

9.3.2 Sarbanes-Oxley (SOX) Compliance 15

10. Appendix 2: Models 16

10.1 Data Flow Diagrams 16

10.1.1 Context Diagram (level -1) 16

10.1.2 Level 0 17

10.1.3 Level 1 18

10.1.4 Level 2 19

10.2 Use Case Diagram 20

10.3 Use Cases 21

10.3.1 UC01: Create a test case 21

10.3.2 UC02: Bulk-edit test cases 22

10.3.3 UC03: Import test cases 23

10.3.4 UC04: Create a test pass 24

10.3.5 UC05: Run a test pass 25

10.3.6 UC06: Branch Cases 26

10.3.7 UC07: Integrate cases 27

10.4 Robustness Diagram – UC03: Import test cases 29

10.5 Robustness Diagram – UC06: Branch test cases 30

10.6 Sequence Diagram – UC03: Import test cases 31

10.7 Sequence Diagram – UC06: Branch test cases 32

10.8 Collaboration Diagram – UC03 and UC06 33

10.9 State Transition Diagram 34

34

10.10 Class Diagrams 35

36

10.11 Data Dictionary 38

11. Appendix 3: Contract 41

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| John Q. Adams (project manager) | 2/6/2008 | Document creation | 0.1 |
| Ethan Crawford (CTO) | 2/10/2008 | Additional edits | 0.2 |
| Ethan Crawford (CTO) | 3/10/2008 | Incorporated Customer feedback and added requirements modeling diagrams | 1.0 |

# Executive Summary

Macrofirm Inc, a 200-person business focused on software test outsourcing, is in need of a solution for managing suites of test scenarios and tracking the results of executing those scenarios. As Macrofirm’s business has expanded, the internal process for creating test cases and running test passes has begun to consume much more employee time, costing the company money and impeding organizational productivity.

SoftCell Consulting proposes to build for Macrofirm a reliable, scalable system capable of storing test scenarios and associated project management data, and enabling execution and results storage of structured test passes. The system, the SoftCell TestCaseManager, will be built using commercially-available, industry-proven technologies from Microsoft, including SQL Server 2005, C# version 3.0, and .Net Framework version 3.0. These integrated technologies will allow rapid application development now and simplified maintenance costs down the road.

The project will begin in March 2008, and will run 10 working weeks, with an expected completion date of June 1, 2008.

The total estimated cost of the development is $104,200, and is expected to save Macrofirm around $37,500 per annum by allowing the company to employ fewer testers to perform case creation, maintenance, and test pass reports generation.

# Problem Statement

Macrofirm does a brisk business providing Quality Assurance resources and expertise to software firms both in the Bothell area and across the world. The services provided by Macrofirm allow software companies whose primary emphasis is development to outsource the testing effort. Macrofirm derives its competitive advantage both from efficiency of execution and economies of scale. Because its bids are mostly fixed price, operating efficiently is critical to the corporate bottom line.

The Quality Assurance verification process involves creating, organizing, and categorizing test scenarios (also known as test cases), and creating, executing, and updating collections of results,. Each result is associated with a test case, and the result state (pass/fail/block) indicates the overall quality of the tested product as measured by that case set.

Up to this point, Macrofirm has used a system involving Microsoft Word documents for case creation, and employee-written macros and Excel spreadsheets for results tracking. This has worked fairly well, but, as Macrofirm’s customer base has expanded, this has lead to scalability issues with the Word and Excel file formats and compatibility problems for repeat customers, whose cases across versions must be forked and integrated along with the source code. This has lead to much effort devoted to repeatedly re-organizing the document filesystem hierarchy, moving cases from one document to another, and managing data corruption issues in the meantime. The reporting structure has also lead to customer complaints and at least one major incident of a lost contract, due to the customer shipping a major bug that was not tracked in the reports, even though the test case was created.

Macrofirm is in need of a test case management system that scales well, saves time and money when handling repeat business, and enables rich report generation and results tracking. If the current solution is not upgraded, the following problems will continue to occur:

* Corrupted documents
* Loss of data due to failing workstation hard drives
* Loss of data due to ad-hoc process causing data to be saved in non-centralized locations
* Increased cost due to need for rewriting individual documents
* Increased cost due to the need to manually reorganize case documents within a filesystem
* Inability to leverage historical data for incremental process improvement
* Increased investment in throwaway tools and scripts to compile test pass data and generate reports

## Goals

* **Scalability:** test case management solution should expand gracefully with little to no increased overhead.
* **Context:** test cases should be stored in a long-term, maintainable location, to enable repeat customers to pick up where they left off.
* **Accountability:** results tracking system should store case state and tester information
* **Transparency**: results tracking system should generate reports with rich data to satisfy internal management and external customer requirements

## Terminology

* **Test Case:** the written representation of a user or system scenario
* **Test Pass:** A Quality Assurance team activity consisting of the execution of one or more test cases
* **Revision Control System**: A software system used to archive multiple revisions of a file
* **TcmApiWebService**: Test Case Manager Application Programming Interface: the server-side business logic between the UI and the database, exposed via a web service.
* **API**: Application Programming Interface, an external layer exposed by a software system for allowing systems to communicate with each other.

## Key Stakeholders

* **Macrofirm Quality Assurance engineers** – consumers of all test case manipulation and test pass execution infrastructure
* **Macrofirm managers** – consumers of test pass reporting infrastructure
* **External customers** – consumers of test pass reporting output

## Summary of Requirements

SoftCell Consulting proposes to eliminate case storage in documents and filesystem, replacing it with a Windows-based business application connected to a SQL Server 2005 back-end in a 4-machine cluster. The datastore will be connected to a commercial revision management system which will allow test cases to be forked and integrated for repeat customers. New user interface forms will be created for test case management and results reporting.

# Solution

## Architecture



*Figure 1: Architectural diagram*

### SQL Server 2005 and front-end

Macrofirm frequently runs large test passes for customers that involve the temporary employment of 20 or more people, all of whom will generate traffic on the server at the same time. During these times of heavy traffic, server responsiveness must be unimpeded, as delay will cause the test pass to be completed and hurt customer satisfaction. To address this performance requirement, Microsoft SQL Server 2005 will be used in a 4-machine cluster as the datastore. Custom stored procedures (SQL queries) will be created to access test cases and results.

A front-end server will act as a common connection point for data access. This layer will expose SQL stored procedures via a server-side web services API and provide business rules that manage server-side state and transactions.

One-time software cost: $3,500 for an unlimited access license.

### Visual SourceSafe

Visual SourceSafe will be used as the versioning software. Every night, an automated process will move changed test cases from SQL Server 2005 into Visual SourceSafe 2005, creating a new revision of each case.

One-time software cost: $1,500 for an unlimited access license.

### C# front.end

Using Windows Forms and .Net Framework version 3.0, the required forms will be created to communicate with the SQL Server 2005. A business logic layer and a public web-service (TcmApiWebService) will be created for the UI to communicate with, using Microsoft Windows Communication Foundation tools.

Cost: $0. (SoftCell Consulting owns licenses to Visual Studio and Windows Communcation Foundation development tools, and provides them without cost as a service to its customers.)

## User Interfaces

**NOTE:** See appendix for screenshot of sample forms and summaries of all major user interfaces. The precise visual appearance will be defined in a later style guide and used in the detailed design documentation.

* Test case hierarchy creation/view/modification
* Individual test case creation/modification
* ResultsManager collection creation/modification
* Results record modification
* Reports generator with customizable UI

## Project Plan and Schedule

The project will be a single phase, 12 week development effort consisting of 2 weeks of design, 8 weeks of development and 2 weeks of stabilization. It is scheduled to begin on March 10, 2008, and complete 12 weeks later at the beginning of June.

In the second week of March, the development lead will create an architectural document and present it to the Macrofirm collaboration committee. This document will undergo one feedback incorporation pass in the 3rd week of March, and the feature set will be locked in preparation for major implementation in April. The phases are outlined as follows:

* **Design:** March 14 – March 31
* **Implementation:** April 1 – May 14
* **Stabilization:** May 17 – June 1

# Cost and Benefits

## Total Cost of Ownership

Software creation is only a fraction of the total cost of any software system. To show our confidence in the efficacy of the proposed solution, we offer this breakdown of associated costs, both to assist with planning and provide a basis for comparison with competing systems.

## Systems and infrastructure

5 new Windows Server 2003 servers - $10,000

4 new Windows office workstations - $5,500

Misc peripherals - $1,000

-----------------------------------------------------------------------------------------

Total: $16,500

## Development costs

### Human Resources

1 development lead

1 developer

1 test lead

### Development Schedule

Design: 2 weeks by development lead

Test Design: 2 week by test lead

Implementation: 10 weeks by development lead and developer

Testing: 10 weeks by test lead

### Salaries

Development lead: 12 weeks at $65/hour (40-hour week): $31,200

1 developer: 10 weeks at $50/hour (40-hour week): $20,000

1 test lead: 12 weeks at $50/hour (40-hour week): $24,000

-----------------------------------------------------------------------------------------

Total: $75,200

### Overhead

To support the development effort, one Macrofirm test lead is requested to participate in the daily development effort during the 8-week milestone, to represent the customer perspective. This will be an approximate commitment of 10 hours/week for one month. We ask that Macrofirm cover the cost of their employee’s benefits and salary during this time of collaboration, a cost we estimate at roughly $5,000.

## Operations

### Systems Administration and Maintenance

Database maintenance and scheduled bug-fixes will be performed by SoftCell Consulting on an annual basis, per the operations contract. On-demand administration for high-priority issues is available at a per-incident rate of $75/hour.

### Training

One-time training with existing online documentation and training materials is included at no additional charge. Additional training is available on-demand, at a rate of $50/hour/student, plus $500 flat classroom fee to cover instructor time and transportation.

### Technical support

Technical support is available via phone and online support at a rate of $100/service request.

# Benefits

Each benefit comes with a relatively quantifiable monetary benefit, whether short-term or long-term. The financial benefit is taken from median employee wages as reported by the Federal Department of Labor (<http://www.dol.gov/dol/topic/wages/index.htm>).

**NOTE:** all metrics below are in single man-hours, and should be multiplied per tester for annual projections.

## Increased Productivity

By streamlining the old process of updating multiple Word documents, copying them to the correct location in the filesystem, and maintaining separate Excel spreadsheets for running test passes and reporting results, Macrofirm will shave incremental units off of each work activity. Over time, this will add up to significant and consistent cost-savings, enabling the company to either absorb more business or employ fewer engineers. The table below delineates the predicted itemized costs:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Activity** | **Current Duration** | **Predicted Duration** | **Delta** | **Predicted Savings** |
| Creating test cases | 10 minutes/case | 5 minutes/case | 5 minutes/case | $12,500/year |
| Running test passes and tracking results | 30 minutes/case | 20 minutes/case | 10 minutes/case | $25,000/year |

# Formal Cost/Benefit Analysis

## Tangible Costs:

The tangible costs will be highest during the first year in which the system development and implementation and most of the training will be performed. The costs incurred during this year are shown in the following table.

The costs for the remaining years are primarily expected to be for software support and hardware upkeep, as shown in figure 1:

|  |  |  |  |
| --- | --- | --- | --- |
| Figure 1: F i r s t Y e a r E x p e n d i t u r e s | | | |
| Software Development Costs: | |  |  |
|  | Lead Software Developer (12 weeks @ $65/hour) | $31,200 |  |
| Software Developer (10 weeks @ $50/hour) | $20,000 |
| 1 Lead Software Test Engineer (12 weeks @ $50/hour) | $24,000 |
| Associated fixed overhead costs | $2,500 |
|  |  |
|  |  |
|  |  |
| Total Software Development Costs | |  | **$77,000** |
| Hardware Costs | |  |  |
|  | Servers and workstations | $15,500 |  |
|  |  |
|  |  |
|  |  |
| Miscellaneous peripherals, cables, and accessories | $1,000 |
| Total Hardware Costs | |  | **$16,500** |
| Training | |  | $6,000 |
|  |  |  |  |
|  |  |  |  |
| Support | |  | $4,000 |
| **Total First Year Expenditures** | |  | **$104,200** |

The costs for the remaining years are primarily expected to be for software support and hardware upkeep, as shown in figure 2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Figure 2: C o s t B r e a k d o w n : E x p e n d i t u r e s b y Y e a r | | | | |  |  |  |
| Cost Description | Year 1 | Year 2 | Year 3 | Year 4 | Year 5 | Year 6 | **Total** |
| Software development costs | 77,000 | 0 | 0 | 0 | 0 | 0 | **77,000** |
| Hardware | 16,500 | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 | **21,500** |
| Training | 6,000 | 2,000 | 0 | 0 | 0 | 0 | **8,000** |
| Support | 4,000 | 3,000 | 2,000 | 2,000 | 2,000 | 2,000 | **15,000** |
| **Yearly Totals** | **104,200** | **6,000** | **3,000** | **3,000** | **3,000** | **3,000** | **122,200** |

## Payback Analysis and Return on Investment (ROI)

Figure three shows that the system will pay for itself within 4 years, with near break-even after 3.

Figure 3: Payback analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Year | Costs | Cumulative Costs | Benefits | Cumulative Benefits |
| 1 | 104,200 | 104,200 | 37,500 | 37,500 |
| 2 | 6,000 | 110,200 | 37,500 | 75,000 |
| 3 | 3,000 | 113,200 | 37,500 | 112,500 |
| 4 | 3,000 | 116,200 | 37,500 | 150,000 |
|  |  |  |  |  |

Return on Investment (ROI) = 29% over 4 years

# Risk Management

## Technology

* **Description:** SoftCell developers may need assistance with Macrofirm Quality Assurance terminology.
  + **Probability:** High
  + **Impact:** Medium
  + **Mitigation:** have customer on-site to answer questions or establish a remote liaison
* **Description:** Visual SourceSafe may not scale appropriately to meet business needs
  + **Probability:** Medium
  + **Impact:** High
  + **Mitigation:** Ensure that adequate VSS table sizes are monitored by Systems Administrator and daily backups created.

## Business

* **Description:** Undiscovered requirements may emerge
  + **Probability:** High
  + **Impact:** High
  + **Mitigation:** Macrofirm customer representative on-site to observe SoftCell development effort and provide real-time feedback

## Organizational

* Users may not quickly migrate from Word and Excel documents to the C# Win32 client/server architecture
  + **Probability:** High
  + **Impact:** Large
  + **Mitigation:** provide up-front training
* End-user resistance to change
  + **Probability:** High
  + **Impact:** Medium
  + **Mitigation:** provide staged and repetitive rollout; hold Q&As to allow a change for people to get their concerns addressed; enable small-scale beta test
* Staffing problems caused by need for retraining
  + **Probability:** Medium
  + **Impact:** High
  + **Mitigation:** provide staged and repetitive rollout; hold Q&As to allow a change for people to get their concerns addressed; enable small-scale beta test

## Industry

* Rapid rate of technological change may result in new fields being exposed for test case tracking, or other workflows implemented for incremental process improvement
  + **Probability:** High
  + **Impact:** High
  + **Mitigation:** Maintain business relationship; support incremental patches; enable end-user service configuration (version 2)

## Total Exposure and Impact

Total exposure is medium, and business impact is high.

# Summary of Recommendations

Macrofirm has a pressing need to improve its process for managing suites of test cases and handling results in order to ensure that maintenance and overhead costs are kept at a minimum as the company’s business continues to expand. The current internal processes span multiple tools and rely on inefficient, non-unified technologies, the results of which are unnecessary expense and inefficiency within the company.

SoftCell’s proposal of an enterprise-quality, scalable, database-backed system for managing test case and test pass result data meets this need perfectly. At a price of $104,200, including an investment in reliable off-the-shelf components, the system pays for itself within the first two years of operation and provides the necessary infrastructure for Macrofirm to expand its business without exponentially increasing its overhead costs.

The development time of 12 weeks and early development start time of March, 2008 will enable Macrofirm to have a stable, fully-tested system in place by the start of FY09. The cost savings can be clearly seen on the books after that point.

SoftCell is the local expert in database-backed 3-tier enterprise applications. Our firm has a proven track record of delivering maintainable, secure, efficient systems, built on time and within budget. Over 50 customers in the last 4 years have contributed positive feedback about the quality of our work. We stand behind what we do, and hope that this proposal demonstrates our commitment.

# Appendix 1: Specifications

## Definitions and Change Process

### Priority

Each enumerated feature is assigned a priority, defined as the overall importance of the feature within the system. Priorities are defined in the following table, and keywords are synonymous with priority numbers.

|  |  |
| --- | --- |
| Priority | Description |
| 1 | **Must** do – essential to core functionality |
| 2 | **Should** do – important secondary functionality |
| 3 | **May** do – nice to have or fit-and-finish |

### Change Process

Specification changes are part of every software development project. Please contact the document owner (listed in the change history) with specific questions and comments, and your feedback will be incorporated into the next draft of this proposal, or as an addendum to locked specifications, if the specification freeze deadline has passed.

## Functional

### Test Case Manipulation

The primary actor consuming test case manipulation scenarios is the Quality Assurance Engineer, a Macrofirm employee in charge of creating test cases and running test passes.

#### Test Case Creation, Retrieval, Updating and Deletion

#### The system **must** support test case creation, updating, and deletion. A test case is defined as a single user scenario documented as a title, setup, execution steps, and verifications that correspond to the steps. This support is provided via a set of forms in the client application. The Test Case Edit form allows the Quality Assurance Engineer to manually create a new test case or edit an existing one, and contains all the required edit controls, along with buttons to save and discard the changes. Test case retrieval is accomplished through the Test Case Summary form, and test case deletion via a UI option on the Test Case Tree View form.

#### All test case modification actions **may** be accompanied by client and/or server-side validation of submitted data, including description format. This feature involves configuring a predefined set of fields and their required values through custom UI, and is exposed via a single client-side extensibility hook that checks a registry key for a registered DLL and invokes custom UI on that DLL using standard Windows dialog APIs. When the dialog is closed and the return value from the dialog creation function is validated, the test case is passed to the Save API.

#### Test Case Bulk-Editing

#### The system **must** support test case bulk-edit from the Test Case Tree View form. This will be exposed via a users selecting multiple items in the tree view and selecting “bulk edit”. At this point, a special blank Test Case Edit form will open. Changes to this form will be propagated to all the selected items in the Test Case TreeView form. This functionality will be transactional, and a failure in data validation in any of the selected cases will cause the entire transaction to be rolled back, and appropriate error status reported to the Quality Assurance Engineer. This will ensure that the bulk-edit is not applied inconsistently, which could cause later unexpected problems if changes are being used to uniquely identify a set of cases.

#### Microsoft Word Import

#### The system **must** support test case import from the Microsoft Word 2007 document format. The import template must be in predefined format stored within the test case manager, with beginning and ending tags to delineate each case. The Import Cases Wizard contains the user interface for managing this process, and it drives its actions through an abstract IImporter interface, with an underlying implementation that understands how to manipulate the .doc/.docx Microsoft Word file format.

#### Test Case TreeView

#### The system **should** display a form for showing test cases in a tree structure. The Test Case TreeView form should contain an expandable TreeView control for showing the test case hierarchy, and a view pane for viewing a single test case within the hierarchy. The Test Case TreeView form serves as the starting form of the client application, and multiple entry points to other system features are exposed via a menu within this form.

#### Test Case Export

#### The system **should** support retrieval of unformatted full test case data, including test case contents. This will be exposed via an API that returns the full data for a single case as a web service message from the server business logic, via the TcmAPIWebService. Clients will communicate with the web-service via client proxy code, auto-generated using the WSDL.exe tool from Microsoft.

### Test Passes

A test pass is a collection of results generated from executing individual test cases on a customer’s software application and comparing a set of pre-defined verifications to the actual run-time behavior of the application. In the Test Case Manager system, collections of results are created with the status “Not Run”. A Quality Assurance Engineer then executes each case in the collection, setting the results status to the appropriate value. Valid values are “Skipped”, “Blocked”, “Failed”, and “Passed”.

#### Result Collection Creation, Retrieval, Update, and Deletion

#### The system **must** support the manipulation of a result collection, including creation, retrieval, update and deletion. Result collection creation involves an interaction between the client application and the server-side web services API to retrieve a collection of test cases and create a 1:1 mapping between a case and a result. Each result is associated with a collection ID and exists as a separate table within SQL Server 2005. A result record consists of the test case ID, a Result Status, a timestamp and a tester ID. The tester ID corresponds to a tester record in the user datastore and allows for tracking who ran the test case.

Result collections support adding and deleting results, and contain a Result Status field to indicate the state of the test pass.

#### Result Summary Form

#### The system **must** display a result form for executing an individual test case. This form will retrieve its data from the server-side API and display it along with the required buttons to mark the result as run.

#### Result Collection View Form

#### The system **must** display a form for viewing test pass progress. The form should contain a listview with configurable columns for customizing the available data from the underlying result collection.

#### Result Collection Export

#### The system **should** support retrieval of unformatted full test pass results data. This will be exposed via an API that returns the full data set for an entire test pass, including all the non-visible fields such as timestamp, from the server-side business logic.

### Test Pass Reports

#### Reports View Form

#### The system **must** display a report form showing simple data such as the number of cases run, the number remaining, and the pass rate. This data will be retrieved from the server-side business logic.

#### Result Collection Export

#### The system **must** expose a programming interface for retrieving full test pass data, for later report customization. This will be exposed via an API that returns the full dataset for an entire test pass, including all the non-visible fields such as timestamp, from the server-side business logic.

### Test Case Revisions

#### Revision Control Branching

#### The system **must** store the revision history of each test case in a Revision Control System. This system, Visual SourceSafe (VSS), is accessed via its API. On execution of this system function, each test case is added or updated as a new revision in a VSS archive and the change is batch-submitted as an atomic unit. The test cases now live as a separate copy from the previous version and are maintained in this location until multiple branches are merged together.

#### A Quality Assurance engineer **must** be able to integrate forked test cases into a single revision. This involves comparing branched cases and manually or automatically resolving conflicts between the revisions. Automatic revisions conflict resolution is handled by VSS, while manual conflict revision where automatic resolution is unavailable is handled by a human being.

### Systems Administration

#### Delete obsolete result collections

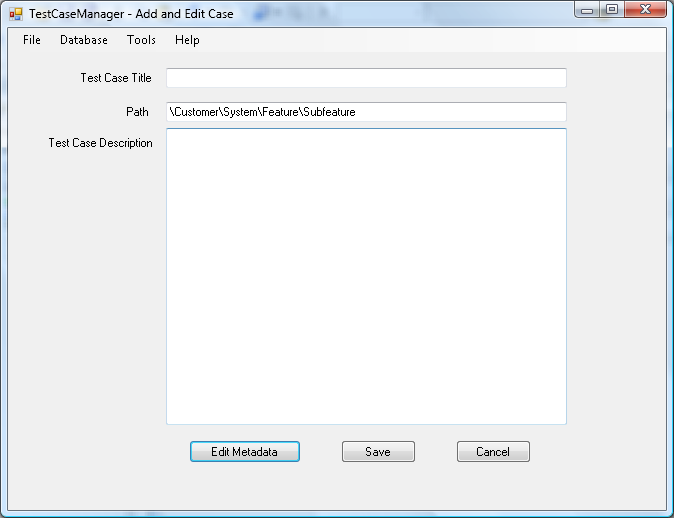
#### The system **must** support auto-aging (deletion) of obsolete test pass result collections. The Systems Administrator uses the System Administration Console form to review stale result collections. If she finds any that are no longer required for business purposes, the console provides her with the option to delete them. The auto-aging time interval is configurable and pre-set to 3 years.

#### Manage Users

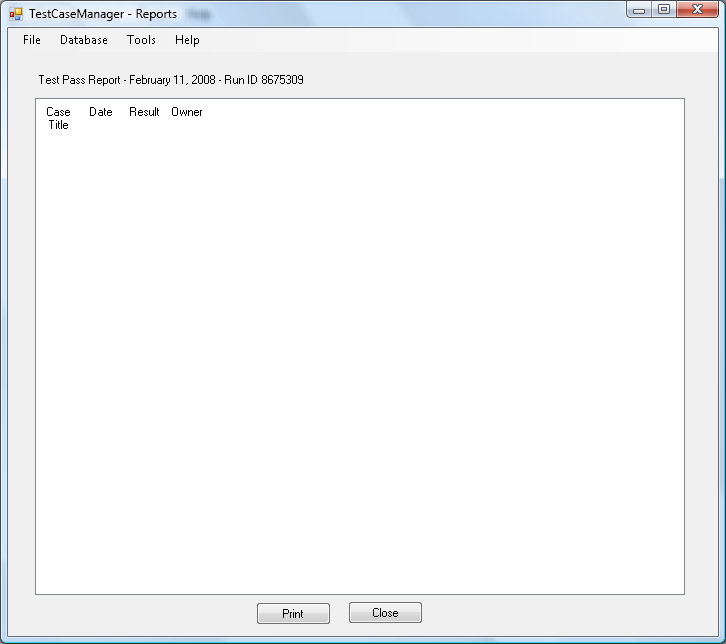
The system **must** support adding, modifying, and deleting user accounts. The Systems Administrator uses the System Administration Console form to perform these functions.

### User Interfaces

#### Figure 2: Test Case Edit Form



#### Figure 3: Result Collection View Form



## Non-Functional

### Performance

Required time to modify a test case or execute an individual result must be less than one second per case. The system must support the simultaneous access of up to 40 testers with no discernable lag in responsiveness, with hardware support added to ensure no loss of productivity during times of heavy load.

### Sarbanes-Oxley (SOX) Compliance

The TestCaseManager system will be complaint with the 5-year data-retention policies of the Sarbanes-Oxley act. This will be accomplished by tagging each deleted item for deletion instead of physically erasing it from the disk.

# Appendix 2: Models

## Data Flow Diagrams

### Context Diagram (level -1)



### Level 0



### Level 1



### Level 2



## Use Case Diagram



## Use Cases

### UC01: Create a test case

**Purpose:**

This use-case defines the behavior of the Test Case Manager system when a Quality Assurance Engineer creates a new test case using the client application’s Test Case Edit form

**Actors:**

Quality Assurance Engineer

**Preconditions:**

1. Quality Assurance Engineer is registered with and logged into the system
2. Test Case Edit form is populated with valid data

**Trigger(s):**

1. Quality Assurance Engineer submits the form

**Scenario:**

1. Client application validates the data against preset template and custom templates (if applicable)
2. Client application passes data to server
3. Server validates the data against preset template
4. Server adds the case to the Test Case store

**Alternative(s):**

1. Custom code directly communicates with server instead of the default form

**Postconditions:**

1. Test case is created on the server

**Exception(s):**

1. Test case data fails server-side validation
2. Network difficulties prevent server/client communication

### UC02: Bulk-edit test cases

**Purpose:**

This use-case defines the behavior of the Test Case Manager system when a Quality Assurance Engineer performs a bulk-edit from the client.

**Actors:**

Quality Assurance Engineer

**Preconditions:**

1. Quality Assurance Engineer is registered with and logged into the system
2. Test Case TreeView form is in the foreground
3. More than one item is selected

**Trigger(s):**

1. Quality Assurance Engineer invokes the “Bulk-Edit” verb through the UI

**Scenario:**

1. Client application creates an internal list of cases to be edited
2. Client application generates a blank Edit form
3. Quality Assurance Engineer fills out the blank form with the desired attributes and ends the editing session
4. Client application signals the server to begin a transaction
5. For each case in the list, client application retrieves each item in the list, modifies the desired fields, and submits the item to the server for updating.
6. Client application signals the server to end a transaction
7. Client application indicates success to the Quality Assurance Engineer

**Alternative(s):**

1. Custom code directly communicates with server instead of the default form

**Postconditions:**

1. Each selected case is modified with the desired attributes

**Exception(s):**

1. Test case data fails server-side validation
2. Bulk-edit transaction fails or times out, causing all changes within the transaction to be rolled back

### UC03: Import test cases

**Purpose:**

This use-case defines the behavior of the Test Case Manager system when a Quality Assurance Engineer imports test cases from a Microsoft Word document

**Actors:**

Quality Assurance Engineer

**Preconditions:**

1. Quality Assurance Engineer is registered with and logged into the system

**Trigger(s):**

1. Quality Assurance Engineer invokes the “Import Cases” verb through the UI

**Scenario:**

1. Client application opens the Import Cases Wizard
2. Quality Assurance Engineer provides the import filename and path
3. Client application loads appropriate import module for Microsoft Word documents
4. Import module parses Microsoft Word document, using the Office object model as an interface, returning test cases to client application
5. Client application validates the data against preset template and custom templates (if applicable)
6. Client application passes data to server
7. Server validates the data against preset template
8. Server adds the case to the Test Case Store, returning the test case ID
9. Import module adds the test case ID to the appropriate case in the Microsoft Word document
10. Client application indicates success to the Quality Assurance Engineer

**Alternative(s):**

**Postconditions:**

1. Each case in the Microsoft Word document is created on the server

**Exception(s):**

1. Import filename is invalid
2. Import filename cannot be opened for read-write access
3. Import module does not exist for the file format
4. Test case data fails server-side validation

### UC04: Create a test pass

**Purpose:**

This use-case defines the behavior of the Test Case Manager system when a Quality Assurance Engineer creates a result collection to be used to run a test pass

**Actors:**

Quality Assurance Engineer, Quality Assurance Manager

**Preconditions:**

1. Quality Assurance Engineer is registered with and logged into the system

**Trigger(s):**

1. Quality Assurance Engineer invokes the “New Result Collection” verb through the UI

**Scenario:**

1. Client application opens the New Result Collection form
2. Quality Assurance Engineer adds one or more cases to the form
3. Quality Assurance Engineer submits the collection for approval
4. Quality Assurance Manager examines the test pass, modifies the collection, approves the changes, and sends an approval request to the system. The system returns the collection to the Quality Assurance Engineer
5. Quality Assurance Engineer sets the collection status to “To Run”

**Alternative(s):**

**Postconditions:**

1. A new collection with status “To Run” is created on the server

**Exception(s):**

1. 0 test cases are added to the result collection
2. Quality Assurance Manager rejects the result collection creation
3. Quality Assurance Engineer does not set the collection status To Run within 30 days

### UC05: Run a test pass

**Purpose:**

This use-case defines the behavior of the Test Case Manager system when a Quality Assurance Engineer logs a complete set of results created for a test pass

**Actors:**

Quality Assurance Engineer, Quality Assurance Manager

**Preconditions:**

1. Quality Assurance Engineer is registered with and logged into the system

**Trigger(s):**

1. Quality Assurance Engineer invokes the “Run Result Collection” verb through the UI

**Scenario:**

1. Quality Assurance Engineer selects the desired Result Collection from the Result Collection List form.
2. For each case in the result collection, Quality Assurance Engineer performs the actions associated with each case on a client’s software system. After completing the case, the Quality Assurance Engineer sets the completion state on the result.
3. Client application sends the result status to the server
4. Server updates the result status in the Results Data Store.
5. When the final case is complete, Quality Assurance Engineer marks the result collection as complete.
6. Client application sends collection status to the server
7. Server sends email to Quality Assurance Manager notifying that a result collection is complete
8. Quality Assurance Manager validates the results and marks the result collection as complete.

**Alternative(s):**

**Postconditions:**

1. Each case in the result collection is marked as complete
2. The result collection is marked as complete

**Exception(s):**

1. Some results in the collection contain Unknown status flags
2. Quality Assurance Manager rejects completion state of the collection, returning it to the Quality Assurance Engineer
3. Quality Assurance Manager fails to sign off on results

### UC06: Branch Cases

**Purpose:**

This use-case defines the behavior of the Test Case Manager system when a Quality Assurance Engineer branches (copies inside a revision management system) a set of test cases.

**Actors:**

Quality Assurance Engineer

**Preconditions:**

1. Quality Assurance Engineer is registered with and logged into the system

**Trigger(s):**

1. Quality Assurance Engineer invokes the “Branch Cases” verb through the UI

**Scenario:**

1. Client connects to Visual SourceSafe to query the available source branches
2. Quality Assurance Engineer selects the source and destination branches from the Branch Selection form.
3. Client application opens the Branch Case Selection form.
4. Quality Assurance Engineer selects the desired set of test cases from the Branch Case Selection form.
5. For each case in the test case collection, the client application copies the case from the source branch into the destination branch.
6. The client application serializes test case data to XML and passes it to Visual SourceSafe.
7. After all changes are submitted, the Quality Assurance Engineer submits the branch commit request to Visual SourceSafe.
8. Visual SourceSafe notifies the Quality Assurance Manger of a change awaiting her approval.
9. After the Quality Assurance Manager approves the change, Visual SourceSafe commits the branched files and stores the change history.
10. The client application displays branch status to the Quality Assurance Engineer.

**Alternative(s):**

1. Quality Assurance Engineer creates a new branch through the client application prior to selecting the set of cases to be branched.

**Postconditions:**

1. Each selected test case is stored as a new revision in the destination branch inside Visual SourceSafe.

**Exception(s):**

1. Network problems prevent connection to Visual SourceSafe
2. Source branch or destination branch does not exist or has been deleted when Branch Case Selection form is launched.
3. Quality Assurance Manager rejects completion state of the collection, returning it to the Quality Assurance Engineer
4. Visual SourceSafe runs out of storage space prior to branch operation being completed

### UC07: Integrate cases

**Purpose:**

This use-case defines the behavior of the Test Case Manager system when a Quality Assurance Engineer integrates two separate test case branches, merges changes and resolves conflicts

**Actors:**

Quality Assurance Engineer, Quality Assurance Manager

**Preconditions:**

1. Quality Assurance Engineer is registered with and logged into the system

**Trigger(s):**

1. Quality Assurance Engineer invokes the “Integrate Cases” verb through the UI

**Scenario:**

1. Client connects to Visual SourceSafe to query the available source branches
2. Quality Assurance Engineer selects the source and destination branches from the Branch Selection form.
3. Client application opens the Branch Case Selection form, prefilled with cases from the source branch.
4. Quality Assurance Engineer selects the cases to be integrated.
5. For each case in the test case collection, the client application runs an “integrate” command inside Visual SourceSafe.
6. Visual SourceSafe inspects the XML-encoded test case and compares it to the destination branch. If an identical file does not exist in the destination branch, it is added as a new file.
7. If the file does exist, Visual SourceSafe attempts an automatic merge of detected changes, using built-in tools.
8. If a conflict cannot be automatically resolved, Visual SourceSafe prompts the Quality Assurance engineer for a hand-merge of the conflicting changes.
9. After all changes are submitted, the Quality Assurance Engineer submits the integrate commit request to Visual SourceSafe.
10. Visual SourceSafe notifies the Quality Assurance Manger of a change awaiting her approval.
11. After the Quality Assurance Manager approves the change, Visual SourceSafe commits the integrated files and stores the change history.
12. The client application displays branch status to the Quality Assurance Engineer.

**Alternative(s):**

**Postconditions:**

1. Each selected test case is integrated into the destination branch inside Visual SourceSafe.

**Exception(s):**

1. Network problems prevent connection to Visual SourceSafe
2. Source branch or destination branch does not exist or has been deleted when Branch Case Selection form is launched.
3. Quality Assurance Manager rejects completion state of the collection
4. Visual SourceSafe runs out of storage space prior to integrate operation being completed

## Robustness Diagram – UC03: Import test cases



## Robustness Diagram – UC06: Branch test cases



## Sequence Diagram – UC03: Import test cases



## Sequence Diagram – UC06: Branch test cases



## Collaboration Diagram – UC03 and UC06



## State Transition Diagram

## 

## Class Diagrams

See next two pages

## 

## Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Stereotype | Description | Datatype | Container |
| Approval Request | Flow (Parameter) | A set of parameters requesting approval | data |  |
| Approval Request Status | Flow (Parameter) | The status of an approval request | data |  |
| BranchCollection | Entity (Class) | Collection of BranchInfo objects |  |  |
| BranchInfo | Entity (Class) | Information about a branch |  |  |
| Client::BranchCaseSelectionForm | Entity (Class) | implements the case selection UI for branching and integration |  |  |
| Client::BranchManager | Entity (Class) | controls communication with the Visual SourceSafe API |  |  |
| Client::BranchSelectionForm | Entity (Class) | implements the branch selection UI |  |  |
| Client::ImportCasesWizard | Entity (Class) | implements the Wizard (multi-dialog UI) for importing test cases from an input source |  |  |
| Client::ImporterFactory | Entity (Class) | creates new Importer objects specific to a filetype |  |  |
| Client::NewResultCollectionForm | Entity (Class) | implements the New Result Collection UI |  |  |
| Client::ReportsManager | Entity (Class) | controls reports generation |  |  |
| Client::ReportsViewForm | Entity (Class) | implements the report viewing UI |  |  |
| Client::ResultCollectionViewForm | Entity (Class) | implements the Result Collection viewing UI |  |  |
| Client::ResultDataAccess | Entity (Class) | handles connection between the client-side Web Services API and the ResultManager |  |  |
| Client::ResultManager | Entity (Class) | controls result manipulation, client-side |  |  |
| Client::ResultSummaryForm | Entity (Class) | implements the single Result view UI |  |  |
| Client::StatusNotificationForm | Entity (Class) | implements the Status notification UI |  |  |
| Client::SystemAdministrationConsoleForm | Entity (Class) | implements the Systems Administration UI |  |  |
| Client::SystemAdministrationManager | Entity (Class) | controls system administration tasks, client-side |  |  |
| Client::TcmAPIProxy | Entity (Class) | the client-side communication with the WCF web services API |  |  |
| Client::TestCaseDataAccess | Entity (Class) | handles connection between the client-side Web Services API and the TestCaseManager |  |  |
| Client::TestCaseEditForm | Entity (Class) | implements the Test Case Edit UI |  |  |
| Client::TestCaseManager | Entity (Class) | controls test case manipulation tasks, client-side |  |  |
| Client::TestCaseSummaryForm | Entity (Class) | implements the single Test Case view UI |  |  |
| Client::TestCaseTreeViewForm | Entity (Class) | implements the Test Case TreeView UI |  |  |
| Client::VssAPIProxy | Entity (Class) | the client-side communication with VSS |  |  |
| Client::WordDocumentImporter | Entity (Class) | controls test case import from Word documents |  |  |
| Collection | Entity (Class) | Collection abstract class |  |  |
| CreatedDate | attribute | The date a test case was created | DateTime | TestCase |
| Credentials | Flow (Parameter) | username and password | data |  |
| Description | attribute | The detailed description of a test case, including setup, steps, and verification | string | TestCase |
| Id | attribute | The ID of a test case | int | TestCase |
| Import Module | Entity (Class) | performs test case import from different file formats |  |  |
| ITcmAPI | Entity (Interface) | An interface used to define a common web services communication scheme between client and server |  |  |
| Metadata Store | Store | A store containing the metadata associated with a single test case |  |  |
| Microsoft SQL Server | External | Relational Database Management System used as the data store |  |  |
| Name | attribute | The name of a test case | string | TestCase |
| New Results Collection form | Entity (Class) | The User Interface class that enables the creation of a new Results collection, used for running a test pass |  |  |
| New Test Case form | Entity (Class) | The User Interface class that enables the creation of a new Test Case |  |  |
| NotificationManager | Entity (Class) | controls sending and receiving notifications |  |  |
| Obsolete Result Collections | Flow (Parameter) | Results collections that are older than a predefined date | data |  |
| Owner | attribute | The owner of a test case | string | TestCase |
| Quality Assurance Engineer | Actor | Actor who will consume most of the Test Case Manager Features |  |  |
| Quality Assurance Manager | Actor | Actor who will consume the management-related features of the Test Case Manager |  |  |
| Result Collection | Flow (Parameter) | A collection of results | data |  |
| ResultCollection | Entity (Class) | contains the results of running a test case |  |  |
| ResultCollection | Entity (Class) | Collection of ResultInfo objects |  |  |
| ResultInfo | Entity (Class) | Information about a test case run result |  |  |
| Results Store | Store | A store containing the test pass results for a set of test cases |  |  |
| Revision information | Flow (Parameter) | Information about a specfic file | data |  |
| Selected Test Cases | Flow (Parameter) | A set of cases selected by a user | data |  |
| Server::ResultsDataAccess | Entity (Class) | server-side communication with SQL Server 2005 |  |  |
| Server::ResultsManager | Entity (Class) | controls result manipulation, server-side |  |  |
| Server::SystemAdministrationManager | Entity (Class) | controls system administration tasks, server-side |  |  |
| Server::TcmAPIWebService | Entity (Class) | exposes the server-side API to clients |  |  |
| Server::TestCaseDataAccess | Entity (Class) | server-side communication with SQL Server 2005 |  |  |
| Server::TestCaseManager | Entity (Class) | controls test case manipulation tasks, server-side |  |  |
| Server::UserDataAccess | Entity (Class) | server-side communication with SQL Server 2005 |  |  |
| Status | attribute | The status of a ResultCollection | ResultCollectionStatus (enum) | ResultCollection |
| Systems Administrator | Actor | Actor who will consume the systems administration features of the Test Case Manager |  |  |
| TcmAPI | Entity (Class) | implements ITcmAPI and contains the business logic on the Test Case Manager server. |  |  |
| TcmAPIProxy | Entity (Class) | implements ITcmAPI and encapsulates the client-side web service calls to the server. It is responsible for converting client-side objects into XML, adding a SOAP wrapper, and consuming the appropriate helper classes to complete a remote Web Services call. |  |  |
| Test Case IDs | Flow (Parameter) | A set of numbers identifying a set of test cases | data |  |
| Test Case Store | Store | A store containing the data required for a single test case |  |  |
| Test Cases | Flow (Parameter) | A collection of information about a user scenario | data |  |
| Test Pass Report | Flow (Parameter) | Formatted report containing information about a test pass | data |  |
| Test Pass Results | Flow (Parameter) | A list of results describing a test pass | data |  |
| TestCase | Entity (Class) | contains the data that represents a new test case |  |  |
| TestCaseCollection | Entity (Class) | Collection of TestCaseInfo objects |  |  |
| TestCaseInfo | Entity (Class) | Information about a test case |  |  |
| User Data | Flow (Parameter) | Information about a user | data |  |
| User Input | Flow (Parameter) | user input | data |  |
| UserCollection | Entity (Class) | Collection of UserInfo objects |  |  |
| UserInfo | Entity (Class) | Information about a user |  |  |
| Visual SourceSafe 2005 | External | Programming subsystem to handle revision control for test cases |  |  |
| Windows Communication Framework | External | Programming subsystem that enables web service communication |  |  |
| XML-formatted test cases | Flow (Parameter) | Test cases, stored as XML, for revision control purposes | data |  |

# Appendix 3: Contract

We, the undersigned, do agree on the particulars of this business proposal.

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Names \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

Company \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_