Kaleidoscope Training System

Software Development Plan

Version 1.0

Release Date: 29 June 2020

Document Number: DI-IPSC-81427

CDRL: NA

Prepared for: Kaleidoscope Case Study, JHU 605.60.81.SU20

Authors:

Brian Hoelzer

Emily List

Brian Loughran

Jason Zhao

Table of Contents

[**1.0 Scope**](#_7ff3n332tjhp) **5**

[1.1 Identification](#_j7czytw0eo7c) 5

[1.2 System Overview](#_jzm03dbwpuvz) 5

[1.3 Document Overview](#_nkoq2nf5gooa) 5

[1.4 Relationship to Other Plans](#_3oqqw7l35sj0) 6

[**2.0 Reference Documents**](#_5apttko8rh8w) **6**

[**3.0 Overview of Required Work**](#_1bc631ij224n) **6**

[3.1 Project Management](#_vt0ychkplb44) 6

[3.2 System Engineering](#_l5xrb9ui13l6) 7

[3.3 Software Engineering](#_m380q9s5epmb) 7

[3.4 Quality Assurance](#_j0yzshuklszr) 9

[3.5 Configuration Management](#_10zj9bq6ywao) 9

[**4. Plans for Performing General Software Development Activities**](#_rrpzgx1qtffh) **9**

[4.1 Software development process.](#_ize2oaivmjrm) 9

[4.2 General plans for software development.](#_rwihpxoevb1o) 10

[4.2.1 Software development methods.](#_ugfhmue33ofq) 10

[4.2.2 Standards for software products.](#_b2uj2293lot9) 11

[4.2.3 Reusable software products.](#_nnwlrrjbdlya) 11

[4.2.3.1 Incorporating reusable software products.](#_9vtomfys5olz) 11

[4.2.3.2 Developing reusable software products.](#_95xi6d4oz3jw) 11

[4.2.4 Handling of critical requirements.](#_cc2h5ew07nua) 11

[4.2.4.1 Safety assurance](#_9buqegq2ah3y) 11

[4.2.4.2 Security assurance](#_sohz4wp1t647) 11

[4.2.4.3 Privacy assurance](#_sohz4wp1t647) 12

[4.2.4.4 Assurance of other critical requirements](#_sohz4wp1t647) 12

[4.2.5 Computer hardware resource utilization.](#_au3j282ojtql) 12

[4.2.6 Recording rationale.](#_au3j282ojtql) 12

[4.2.7 Access for acquirer review.](#_au3j282ojtql) 12

[**5. Plans for performing detailed software development activities**](#_bdsxd23jo73) **13**

[5.1 Project Planning and Oversight](#_p5wn2zi8vemq) 13

[5.1.1 Software Development Planning](#_vgxqwlc1hxq3) 13

[5.1.2 CSCI Test Planning](#_k3b06tayh8p) 14

[5.1.3 System Test Planning](#_whboz79mzxio) 14

[5.1.4 Software Installation Planning](#_cbswdrdvbyt1) 14

[5.1.5 Software transition planning](#_v9sfyinoqpma) 14

[5.1.6 Following and updating plans, including the intervals for management review](#_o43m8xw9mqix) 15

[5.2 Establishing Software Environment - To Be Updated](#_3k1234jb7lpn) 15

[5.2.1 Software Engineering Environment](#_wdejg3ujig21) 15

[5.2.2 Software Test Environment](#_roxd9u1f9fvc) 15

[5.2.3 Software Development Library](#_lyad3nz5wy5s) 15

[5.2.4 Software Development Files](#_tufg38po49au) 15

[5.2.5 Non-Deliverable Software](#_bmwla5blsb40) 15

[5.3 System Requirements Analysis](#_je3sysfgieeu) 15

[5.3.1 Analysis of User Input](#_ltmn8ey7354a) 15

[5.3.2 Operational Concept](#_4dzij93e576v) 15

[5.3.3 System Requirements](#_t4dm52sa1av6) 16

[5.4 System Design - To Be Updated](#_g4sjwxhucqqd) 16

[5.4.1 System-Wide Design Decisions](#_vg4f7e2v9aft) 16

[5.4.2 System Architectural Design](#_u8ky8ay98zkx) 16

[5.5 Software Requirements Analysis](#_vx3hpe3uldyv) 16

[5.6 Software Design](#_k18l42j3z94y) 16

[5.6.1 CSCI-wide design decisions](#_atkrsddobbyj) 16

[5.6.2 CSCI architectural design](#_q28y1n2x4gka) 17

[5.6.3 CSCI detailed design](#_2wnhkj5q5i56) 17

[5.7 Software Implementation and Unit Testing - To Be Updated](#_qc75xkf1q777) 17

[5.7.1 Software Implementation](#_elpa6ak7z1hw) 17

[5.7.2 Preparing for Unit Testing](#_jbqyks7lmll7) 17

[5.7.3 Performing Unit Testing](#_xkmkcvks2enk) 17

[5.7.4 Revisioning and Testing](#_3g5uq8y8wtg) 17

[5.7.5 Analyzing and Recording Unit Test Results](#_g5pykd4bke0t) 17

[5.8 Unit Integration - To Be Updated](#_wumls2lv684t) 18

[5.8.1 Preparing for Unit Integration and Testing](#_71u3ekbw9p9x) 18

[5.8.2 Performing Unit Integration and Testing](#_aow6xau0om16) 18

[5.8.3 Revision and Retesting](#_etuflslch0m7) 18

[5.8.4 Analyzing and Recording Unit Integration and Test Results](#_9730zyurmyd0) 18

[5.9 CSCI Qualification Testing - To Be Updated](#_6ek72s3azr38) 18

[5.9.1 Independence in CSCI Qualification Testing](#_h1vd7uj6vso8) 18

[5.9.2 Testing on the Target Computer System](#_j096ad6cexqj) 18

[5.9.3 Preparing for CSCI Qualification Testing](#_pkzwpdgxz2ck) 18

[5.9.4 Dry Run of CSCI Qualification Testing](#_39nccucxxeq6) 19

[5.9.5 Performing CSCI Qualification Testing](#_ac8vc3mqm8p7) 19

[5.9.6 Revision and Retesting](#_3v7rtog95qj4) 19

[5.9.7 Analyzing and Recording CSCI Qualification Test Results](#_iw2jj3uf9jwj) 19

[5.10 CSCI/HWCI integration and testing - To Be Updated](#_29f2iro38ryx) 19

[5.10.1 Preparing for CSCI/HWCI Integration and Testing](#_hh892dcwes0j) 19

[5.10.2 Performing CSCI/HWCI Integration and Testing](#_rste26asf6br) 19

[5.10.3 Revision and Retesting](#_kpkmzes1y3e) 20

[5.10.4 Analyzing and Recording CSCI/HWCI Integration and Test Results](#_irwom14m7p0p) 20

[5.11 System qualification testing - To Be Updated](#_3uednnqk63j0) 20

[5.11.1 Independence in system qualification testing](#_y4p5pe1fy9mb) 20

[5.11.2 Testing on the target computer system](#_ldqk9a3h8j0o) 20

[5.11.3 Preparing for system qualification testing](#_w36jmfnnaeiq) 20

[5.11.4 Dry run of system qualification testing](#_69nirj61ym50) 20

[5.11.5 Performing system qualification testing](#_ej7ti1hv5eju) 20

[5.11.6 Revision and retesting](#_sl2f1nqwkrfk) 21

[5.11.7 Analyzing and recording system qualification test results](#_86llp4cmg68m) 21

[5.12 Preparing for software use](#_5c5zfwuie3wr) 21

[5.12.1 Preparing the executable software](#_gukkj1qu43ir) 21

[5.12.2 Preparing version descriptions for user sites](#_ikvlbzkg0wsw) 21

[5.12.3 Preparing user manuals](#_agpehoewko5o) 21

[5.12.4 Installation at user sites](#_8p8risr315h8) 21

[5.13 Preparing for software transition - To Be Updated](#_1tlcfuo2so6) 21

[5.14 Software configuration management - To Be Updated](#_ywanfe4os8co) 22

[5.14.1 Configuration Identification](#_uvp8dnlw99y9) 22

[5.14.2 Configuration Control](#_nkcon3e5hhj) 22

[5.14.3 Configuration Status Accounting](#_i183lavgjc2u) 22

[5.14.4 Configuration Audits](#_os8lfrstdcjg) 22

[5.14.5 Packaging, Storage, Handling, and Delivery](#_d5ngberd7j6m) 22

[5.15 Software product evaluation - To Be Updated](#_k7rtx77k5277) 22

[5.15.1 In-process and final software product evaluations](#_xyy199op853) 22

[5.15.2 Software product evaluation records, including items to be recorded](#_2wxoptcw85mo) 22

[5.15.3 Independence in software product evaluation](#_eiav0r5nowp) 23

[5.16 Software quality assurance - To Be Updated](#_k7rtx77k5277) 23

[5.16.1 Software quality assurance evaluations](#_1x735h4xy8vy) 23

[5.16.2 Software quality assurance records, including items to be recorded](#_19mztdxehqcz) 23

[5.16.3 Independence in software quality assurance](#_er1vx05rfry1) 23

[5.17 Corrective Action](#_2g8uo6hdxab6) 23

[5.18 Joint Technical and Management Reviews](#_fkq56h6912j3) 23

[5.18.1 Joint Technical Reviews](#_ieyz989nx8zn) 23

[5.18.2 Joint Management Reviews](#_7xqfodef37zm) 23

[5.19 Other Software Development Activities](#_pmn6pvku01nm) 23

[5.19.1 Risk Management](#_2cdkwrfd57lt) 24

[5.19.2 Software Management Indicators](#_hj1s11oawwwr) 24

[5.19.3 Security and Privacy](#_z4y689rkrovl) 24

[5.19.4 Subcontractor Management](#_txvoi1j8m661) 24

[5.19.5 Interface with Software IV&V Agents](#_mvwiccx6zsgx) 25

[5.19.6 Coordination with associate developers](#_ognefynmhrdi) 25

[5.19.7 Improvement of Project Processes](#_98nwjhxub7r9) 25

[5.19.8 Other Activities](#_4iq9x0fto1bz) 25

[**6. Schedule and Activity Network - To Be Updated**](#_ia4q6avgxsar) **25**

[**7. Project Organization and Resources**](#_6tvmzdz3lq5z) **26**

[7.1 Project Organization - To Be Updated](#_c4ea4bwzh9i2) 26

[7.2 Project Resources - To Be Updated](#_4k8hi4tw7pxb) 26

[**8. Notes**](#_cdics312k7l4) **26**

[**Appendixes**](#_ydde3z7mrg96) **27**

# 1.0 Scope

## 1.1 Identification

This document applies to the Kaleidoscope Software Project.

## 1.2 System Overview

The Kaleidoscope Training System (KTS) can support simultaneous training of six students. Each student occupies a student workstation (Windows) and progresses through computer based training (CBT) material. The training course consists of five units, each requiring six to eight hours of student interaction with the workstation. Each group of six students (a class) is enrolled on Monday morning and graduates on Friday afternoon.

The system is administered by an instructor/operator who uses a dedicated workstation. The course is totally automated, therefore the instructor/operator is not required to provide guidance to the students but does assist when exceptions occur. After logging onto the system, student registration, presentation of each unit, test presentation, test grading, and remedial course presentation, and retest (if necessary) occur automatically.

The six student workstations and instructor/operator workstation are connected via local area network to a Training Management System (TMS) which is hosted on a Sun Workstation. The TMS is used to collect and archive student information (registration, test results, and time required to complete each unit). Software developed to run on the TMS produces reports and performs statistical analysis of student performance and results. Students must not have access to TMS hosted functionality or other student data.

The system is unclassified. Each build is to be developed at the contractor's facility, deinstalled, shipped, and re-installed at the operational site. The contractor has set aside $100,000.00 of capital funds for the computer aided software development environment (CASE tools.)

## 1.3 Document Overview

The purpose of this document is to describe “the approach to the development of the software and is the top-level plan generated and used by managers to direct the development effort”[5]. This document contains the general plans for software development activities including project planning and development methods, software design, software implementation and testing, software reviews, scheduling, and organization. This document and the software system development is UNCLASSIFIED. All data generated is proprietary to the company.

## 1.4 Relationship to Other Plans

N/A

# 2.0 Reference Documents

1. Kaleidoscope Statement of Work
2. Google C++ Style Guide - <https://google.github.io/styleguide/cppguide.html>
3. Git Flow Model - <https://nvie.com/posts/a-successful-git-branching-model/>
4. Systems Engineering Principles and Practice
5. Software Development Plan (Small Project) - <https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/webtmpl/templates/mgmnt/rup_sdpln_sp.htm>

# 3.0 Overview of Required Work

The Kaleidoscope project is current within the Concept Definition of the Concept Development Phase within the system life cycle model. Below are the defined requirements and constraints for this project:

## 3.1 Project Management

3.1.1 The contractor shall establish a project management office and identify a project manager responsible for all aspects (technical, cost, and schedule) of the Kaleidoscope Training System (KTS) development and installation effort.

3.1.2 The Project Manager shall produce and deliver a monthly Project Status report which summarizes cost, schedule, and technical issues.

3.1.3 The Project Manager shall conduct quarterly management reviews which address cost, schedule, risk issues, software management indicators, and very high level technical issues.

3.1.4 The quarterly review agenda shall be published one week before each review and minutes shall be published within two weeks of the conduct of each review.

3.1.5 The Project Manager shall conduct joint technical reviews following significant milestones throughout the project but not less than one every six months.

3.1.6 The joint technical review agenda shall be published one week before each review and minutes shall be published within two weeks of the conduct of each review.

3.1.7 The contractor shall provide office space and electronic access to all engineering products throughout development effort for use by the acquisition organization.

### 3.1.1 Program Metrics

#### 3.1.1.1 Earn Value Management (EVM)

For this project, several metrics will be tracked to ensure that a high quality product is produced within the time and scope of the project. Most of the MITRE metrics that will be tracked are explained in the following section. This section will focus on how this program plans on using Earned Value Management (EVM). EVM is a way to track a project’s progress by looking at cost and schedule together. This is a large enough project where the developer feels that the level of insight provided by EVM will be extremely beneficial.

For this project the developer will use a 25/75 approach to taking credit for milestones. This is a low overhead solution that still allows the program to take partial credit once a milestone is started. Since the SOW has provided an estimate for LOC for each build and capability, the developer will tie earned value to LOC. The developer will plan on having monthly EVM meetings with the customer. These regular reviews will allow the developer and the customer to track progress as it relates to the schedule and budget. If there are any inefficiencies within the program, EVM should identify them early on, allowing the developer to adjust the plan earlier rather than later.

#### 3.1.1.2 MITRE Metrics

Software Size – Software size as a management indicator means tracking two things. The first is an estimate for total project size. This is an estimate that has been done already as part of software development planning and scheduling. The other part of the software size metric is the actual software size. Tracking actual software size against the expected software size should produce results that show how effective each line of code is against initial estimates, reflect any additional requirements, or better understanding of the requirements. The initial estimate for software size should be one which is adjusted to become more accurate as more information comes about actual project size, and should eventually represent actual data.

Software Personnel – Software personnel, similar to software size, means tracking two things. The first is the planned number of staff members on the project, and the second is the actual number of staff members involved in development. Tracking the expected number of staff members vs. actual can be indicators of productivity, predictors of schedule delays and cost overruns, and can be used to make more accurate staffing predictions in the future.

Computer Resource Utilization- An estimation and actual amount of the target systems computer resources being used, including CPU, storage, and communication capacity. This project is developing software that is hardware dependent. It is vital to make sure that the software is using the resources efficiently and does not exceed the threshold for the intended hardware. Estimates for Computer Resource Utilization should be used during the design phase and actuals should be tracked during coding and integration.

Schedule Progress – The schedule progress is measured as a ratio of total completed schedule vs. the ratio of budget spent. If budget is being spent faster than the project is progressing, especially outside of the initial project phases, absent a change in operations, budget overruns can be expected. This upward trend of budget spent vs. schedule progress may indicate things such as poor code quality, inadequate testing, or an initially overambitious schedule.

Life Cycle Dependent Metrics:

During the lifecycle of the program, different metrics will be tracked. During the design phase, Design Progress should be tracked, to make sure that all of the requirements are being properly designed and to keep track requirements stability. Computer Software Unit (CSU) Development Progress should be tracked during the coding phase, to make sure that the capabilities planned for each build are being implemented. Testing Progress should be tracked during testing. This will not only be a good indicator for schedule, but also for code quality. Throughout all three builds, Incremental Release Content will be tracked. This will allow the program to track the capabilities that were planned for each release and the actual capabilities released.

## 3.2 System Engineering

3.2.1 System engineering shall be responsible for System Qualification Testing of each completed build at the contractor's facility. In addition, following the completion of the 3rd build, System Qualification Testing shall be conducted at the operational site.

3.2.2 System engineering shall be responsible for delivery of the KTS and installation at the operational site.

3.2.3 System engineering shall assess performance of the hardware and software system to ensure that computer components support performance requirements.

3.2.4 System engineering shall produce and deliver the following documents:

• System Qualification Test Plan

• System Qualification Test Procedures

• System Qualification Test Reports (pre-installation and post-installation)

• Installation Plan

• Performance Analysis report

## 3.3 Software Engineering

3.3.1 The software engineering organization shall support the program manager in preparation for and conduct of quarterly management reviews and joint technical reviews (refer to SOW paragraphs 3.1.3 through 3.1.5).

3.3.2 The software engineering organization shall support the system engineering organization during the system requirements analysis and system design phases.

3.3.3 Software development shall be conducted using an evolutionary development model consisting of three incremental deliveries as follows:

|  |  |  |
| --- | --- | --- |
| **Build** | **Student Console Functionality** | **Training Management System Functionality** |
| 0 | · Unit 1 CBT  · Communications Management  · Registration  · System Startup / Shutdown | · Archive Management  · Communications Management  · System Startup / Shutdown |
| 1 | · Unit 2 CBT  · Unit 3 CBT | · Student Support  · Report Processing  · Inst/Operator Support |
| 2 | · Unit 4 CBT  · Unit 5 CBT  · Error Handling | · Statistical Analysis  · Error Handling |

Table 1: Deliverables of Kaleidoscope Project

3.3.4 All software development shall utilize the C++ programming language. Object oriented analysis and design shall be the method used for requirements analysis and software design.

3.3.5 As a by-product of the software engineering process, the Software Engineering organization shall produce and deliver the following documents for build 0:

• Software Development Plan

• Software Requirements Specification

• Interface Requirements Specification

• Software Design Document

• Interface Design Document

• Software Product Specification

• CSCI Qualification Test Plan

• CSCI Qualification Test Description

• CSCI Qualification Test Report

• Student User Manual

• Instructor User Manual

The Software Development Plan shall be compliant with to ISO 12207 tailored as specified in this paragraph and elsewhere in this SOW. The contractor is not responsible for preparing for software transition and related parts of the SDP shall be tailored out. Software product evaluations and software quality assurance shall be the responsibility of the quality assurance organization and both topics shall be addressed within the software quality assurance paragraph of the SDP. The contractor is not required to develop a separate quality assurance plan.

3.3.6 The software engineering organization shall produce a Version Description Document (VDD) for each build.

3.3.7 The software engineering organization shall enhance appropriate Build 0 documentation (rather than originate new documents) for subsequent builds, thereby minimizing the total documentation set.

## 3.4 Quality Assurance

3.4.1. An independent Quality Assurance authority shall review documentation and other software products (software product evaluation) to ensure that a sound, comprehensive software development program is implemented.

3.4.2. The QA authority shall review processes and activities throughout the development of KTS to ensure compliance with plans and objectives (software quality assurance.)

3.4.3. The QA authority shall implement continuous process improvement initiatives within the budget constraints of the quality program. Capability Maturity Model Integration®(CMMI®) level 3 objectives shall be emphasized.

## 3.5 Configuration Management

3.5.1 Configuration management shall be employed to control the product baseline for all builds.

3.5.2 The CM organization shall establish and maintain a corrective action reporting system to track Problem/change reports from identification to closure. This system shall utilize Sybase to record and track items and generate appropriate reports.

3.5.3 The CM organization shall maintain a description of the configuration of each build including COTS software products and shall be able to recreate any build if required.

# 4. Plans for Performing General Software Development Activities

## 4.1 Software development process.

As discussed in Section 3: Overview of Required Work, the project shall be broken up into builds. The builds shall be broken up according to the following table:

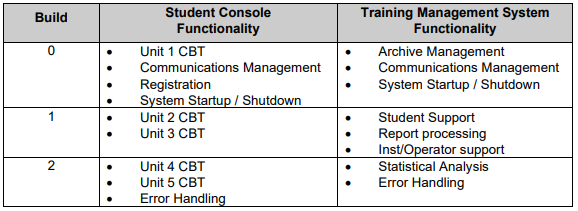


Table 2: Build Breakdown

In addition to software deliverables, the software engineering organization shall produce and deliver the following for build 0:

* Software Development Plan
* Software Requirements Specification
* Interface Requirements Specification
* Software Design Document
* Interface Design Document
* Software Product Specification
* CSCI Qualification Test Plan
* CSCI Qualification Test Description
* CSCI Qualification Test Report
* Student User Manual
* Instructor User Manual

Each of the above documents will also be updated rather than recreated for each subsequent build, resulting in each of the above documents being current with the final build 2.

## 4.2 General plans for software development.

### 4.2.1 Software development methods.

The software development process to be employed for this project is the evolutionary model. The evolutionary model is an extension of the waterfall model allowing for a subset of the functionality to be delivered with additional functionality to subsequent builds. Due to the staged nature of the builds this will allow for incremental features to be added to previous builds. The evolutionary model also integrates nicely with other systems and subsystems which will be important in integrating with the other COTS packages described in section 7.2.

### 4.2.2 Standards for software products.

All software development shall utilize the C++ programming language. Object oriented analysis and design shall be the method used for requirements analysis and software design. C++ coding standards are not specified by the customer, so internal coding standards shall be used in development. Some good standards to be applied include CamelCase for classes and function names, and underscore\_spacing for variable names. The header comment of each file should identify the software maintainer, code version number, and notes on the data. Each function should have a function header describing the function, inputs and outputs. Modular code design is always preferred as well.

### 4.2.3 Reusable software products.

#### 4.2.3.1 Incorporating reusable software products.

N/A

#### 4.2.3.2 Developing reusable software products.

N/A

### 4.2.4 Handling of critical requirements.

### 4.2.4.1 Safety assurance

Safety assurance shall be handled by the systems engineering team. System engineering shall be responsible for System Qualification Testing of each completed build at the contractor's facility. In addition, following the completion of the 3rd build, System Qualification Testing shall be conducted at the operational site. Additionally in an effort to ensure total software safety, the following test plans and qualification reports shall be produced:

* System Qualification Test Plan-
* System Qualification Test Procedures
* System Qualification Test Reports (pre-installation and post-installation)
* Installation Plan
* Performance Analysis report

#### 4.2.4.2 Security assurance

While the customer does not make any specific requests concerning security, implied with any software project is the assumption of a secure system. Since there are no specific customer requests, software development will proceed with standard security practice internal to the software developers. Standard security practices include routine software package updates, enforcement of least privilege, segmented networks, documented security policies, and more.

#### 4.2.4.3 Privacy assurance

Similar to the lack of customer requests concerning security, there are no specific requests concerning privacy, however the assumption for any software systems is a private connection. While the instructor/operator should have access to each student workstation, no students should have access to the instructor/operator workstation, nor should students have access to another student’s workstation. The software system should have automated write and read operations to archive data storage, however tasks outside of those automated write and read operations should not be allowed to prevent corruption of archive data storage.

#### 4.2.4.4 Assurance of other critical requirements

Robust documentation will be provided to the customer throughout the duration of the project.The documentation to be provided to the customer can be referenced in sections 4.1 and 4.2.1. All requirements of the project will be met, and continuous documentation and testing will provide regular updates on the status of those critical requirements. Further, an independent quality assurance authority shall review documentation and software products to ensure a sound, comprehensive software system.

### 4.2.5 Computer hardware resource utilization.

N/A

### 4.2.6 Recording rationale.

Throughout the course of the project, many software decisions will have to be made by the software engineers, thus recording rationale for those decisions will add value to the general software product. Rationale for design decisions will be made a part of the associated Software Design Document and the Interface Design Document which are required documents as part of the software engineering activities.

### 4.2.7 Access for acquirer review.

The majority of review shall be conducted by an independent quality assurance authority. The independent quality assurance authority shall review documentation and other software products (software product evaluation) to ensure that a sound, comprehensive software development program is implemented. Further, the QA authority shall review processes and activities throughout the development of KTS to ensure compliance with plans and objectives (software quality assurance) and shall implement continuous process improvement initiatives within the budget constraints of the quality program.

# 5. Plans for performing detailed software development activities

## 5.1 Project Planning and Oversight

### 5.1.1 Software Development Planning

|  |  |  |
| --- | --- | --- |
| **Build** | **PC Environment CSCIs (LOC)** | **Sun Environment CSCIs (LOC)** |
| 0 | 1. System Startup / Shutdown (500) 2. Registration (1000) 3. Unit 1 CBT (2000) 4. Communications Management (1000)   Total LOC: 4500 | 1. System Startup / Shutdown (1000) 2. Communications Management (1000) 3. Archive Management (4500)   Total LOC: 6500 |
| 1 | 1. Unit 2 CBT (1500) 2. Unit 3 CBT (2200)   Total LOC: 3700 | 1. Student Support (1800) 2. Inst/Operator Support (1200) 3. Report Processing (6500)   Total LOC: 9500 |
| 2 | 1. Unit 4 CBT (1800) 2. Unit 5 CBT (1500) 3. Error Handling (500)   Total LOC: 3800 | 1. Statistical Analysis (3000) 2. Error Handling (1000)   Total LOC: 4000 |

Table 3: Build Structure

Development of the Kaleidoscope project is broken up into three builds, with the CSCIs broken up into two parallel development teams: one for the PC environment, and one for the Sun environment.

Within the time frame of each build, the evolutionary model of software allows for phases of planning, development, testing, and delivery periods. The team should aim to budget time for each of the phases described above, as well as time to write the documentation of the functionality within each build. Because development is split up between builds, lessons learned from earlier builds can help inform the planning phases for subsequent builds, and team leads should utilize the knowledge and lessons learned in order to guide future planning meetings.

For this project, the software development team will be split up between the two aforementioned development teams as well as a QA team and an Integration team.

### 5.1.2 CSCI Test Planning

Each project build is divided into several distinct CSCIs, as shown in table

As each CSCI is completed by the development team, it is handed over to the quality assurance (QA) team for systems testing before integration. Before hand-off, the development team should ensure that the CSCI is in a stand-alone functional state, with the test interfaces used for development clearly marked and documented. Since the evolutionary model calls for the overlap of development activities, the development team is free to start development of the next CSCI once this hand off occurs, but the team should set aside time to fix any issues that arise during the QA.

For each CSCI, the QA team is responsible for testing the functionality of the item against the documented list of requirements and ensuring there is a one-to-one match. Discrepancies should be flagged for the development team to review.

The issues raised during QA must be addressed by the development team at a higher priority level than the development of the next CSCI. Failure to do so at this stage delays integration of the software into the current build of the project. Once all issues are addressed, the QA team passes the CSCI to the integration team.

### 5.1.3 System Test Planning

The integration team is responsible for integrating the CSCIs handed off from the development team into a working system and are responsible for testing the links between CSCIs. If issues arise during this integration testing, development team members may be brought into the loop to assist in fixing the issues. Once all CSCIs planned to be developed in the current build have been integrated, the build is considered complete from a development perspective.

### 5.1.4 Software Installation Planning

The integration team is responsible for installing the integrated CSCIs as they become available to ensure the system functions as a whole.

### 5.1.5 Software transition planning

NA

### 5.1.6 Following and updating plans, including the intervals for management review

Every 3 months, the project manager is responsible for reviewing the current state of the project in order to assess progress of the project. The goals of this review is to both assess status indicators such as cost, schedule, risk, and technical issues, but also to provide these indicators in a report to the client.

## 5.2 Establishing Software Environment - To Be Updated

### 5.2.1 Software Engineering Environment

NA

### 5.2.2 Software Test Environment

NA

### 5.2.3 Software Development Library

NA

### 5.2.4 Software Development Files

NA

### 5.2.5 Non-Deliverable Software

Test scripts and other temporary software created to test the development code should be integrated into a separate git repository.

## 5.3 System Requirements Analysis

### 5.3.1 Analysis of User Input

The team is required to review project documentation and customer surveys, as well as meet with the client as needed in order to clarify the requirements of this project.

### 5.3.2 Operational Concept

The Kaleidoscope Training System is intended to serve as an integrated hardware and software solution to provide a training environment for an instructor to teach and train up to six students simultaneously; these systems are to be built on the Windows OS and are able to communicate with one another. The system should track and access student progress and allow the instructor to intervene during training lessons whenever a student requires help. Finally, the system should be able to analyze statistical data collected during training sessions and output the results.

### 5.3.3 System Requirements

Student and instructor workstations are to be developed on the Windows operating system. These workstations must be able to communicate with each other over a LAN network.

* + Up to 6 student workstations that can operate simultaneously
  + 1 instructor workstation to monitor and assist students

Each student workstation gives the student access to course units that include coursework as well as evaluation tests. A total of 5 course units will be developed during the course of this project development.

A training management system connected to each workstation via LAN that tracks student progress and is able to output progress reports and historical statistics.

## 5.4 System Design - To Be Updated

### 5.4.1 System-Wide Design Decisions

NA

### 5.4.2 System Architectural Design

NA

## 5.5 Software Requirements Analysis

The development team will be required to produce a detailed requirements document during the planning phases of each build cycle by analyzing the information available from the client as well as brainstorming desired functionality. Once this document is available, it will be shared with the client for review and updated as appropriate. If additional requirements are identified during the development process, the requirements document should be amended and reported to the client.

## 5.6 Software Design

### 5.6.1 CSCI-wide design decisions

Due to the incremental development project schedule, the interconnections between each CSCI must be planned out ahead of time in order to standardize the connections between CSCIs created early in the development stage and the CSCIs created later. In order to facilitate this standardization of interconnection, the project will be designed in a network graph structure, with the necessary connections between each CSCI mapped out in advance.

### 5.6.2 CSCI architectural design

Each CSCI should be designed with a high modularity in mind in order to facilitate parallel development amongst members of the team.

### 5.6.3 CSCI detailed design

Code design should follow the Google C++ Style Guide [2].

## 5.7 Software Implementation and Unit Testing - To Be Updated

### 5.7.1 Software Implementation

As various software units are created, the development team should review and ensure that it matches the required functionality specified in the design documentation. Before the unit can be integrated into its assigned CSCI, it must be subjected to formal testing.

### 5.7.2 Preparing for Unit Testing

In preparation of submitting the software unit for testing, the developer(s) responsible for the unit should ensure that the unit is functional and that the documentation for the unit is fully up to date and available to the testing team.

### 5.7.3 Performing Unit Testing

The testing team, with the aid of the documentation and the specified functionality, should ensure that said functionality is implemented correctly.

### 5.7.4 Revisioning and Testing

If the testing team identifies significant issues with the software unit, the unit should be sent back to the development team for correction and revision. This cycle should continue until all issues are addressed. Updates to the unit documentation should be made where appropriate.

### 5.7.5 Analyzing and Recording Unit Test Results

A list of issues identified in testing, and their subsequent resolutions, should be noted in order to help give guidance on future development. In addition, the details of the tests should be recorded.

## 5.8 Unit Integration - To Be Updated

### 5.8.1 Preparing for Unit Integration and Testing

Once the software unit has undergone unit testing, the integration team can prepare to integrate the software unit with the rest of the CSCI. The team should review the documentation from the development of the unit and that from the unit testing.

### 5.8.2 Performing Unit Integration and Testing

As per the design documentation, the team is responsible for integrating the unit with other units of the CSCI. The CSCI, as it currently exists, should be tested to ensure the interlinks between units are working as designed.

### 5.8.3 Revision and Retesting

If the integration team identifies significant issues with the software unit, the unit should be sent back to the development team for correction and revision. This cycle should continue until all issues are addressed. Updates to the unit documentation should be made where appropriate.

### 5.8.4 Analyzing and Recording Unit Integration and Test Results

A list of issues identified in testing, and their subsequent resolutions, should be noted in order to help give guidance on future development. In addition, the details of the tests should be recorded.

## 5.9 CSCI Qualification Testing - To Be Updated

### 5.9.1 Independence in CSCI Qualification Testing

CSCI qualification testing will be performed by an independent testing team contracted for the purpose of ensuring that all the requirements for each CSSI are correctly implemented.

### 5.9.2 Testing on the Target Computer System

Qualification testing should occur on the hardware designed by the hardware team.

### 5.9.3 Preparing for CSCI Qualification Testing

Each CSCI must be installed on the hardware designed by the hardware team, as well as have rudimentary user documentation available to allow the testing team to utilize each requirement designed. The testing team should utilize the documentation in order to design a test plan for each tester to run through.

### 5.9.4 Dry Run of CSCI Qualification Testing

A dry run of the test plan should be performed to ensure that the documentation given is correct and that the test plan can be implemented.

### 5.9.5 Performing CSCI Qualification Testing

Each tester should follow the test plan and document the results of each test. As each requirement is tested, it should be checked off if it passes every test or marked for revision if not.

### 5.9.6 Revision and Retesting

The development team responsible for the CSCI marked for revision should review the test results and implement corrections necessary. Once all corrections are made, the CSCI should be subjected to another round of testing.

### 5.9.7 Analyzing and Recording CSCI Qualification Test Results

Qualification test plans and results should be archived to allow the development team time to analyze between each build in order to determine what can be improved during the development of the next build.

## 5.10 CSCI/HWCI integration and testing - To Be Updated

### 5.10.1 Preparing for CSCI/HWCI Integration and Testing

As each CSCI during a build completes development, the integration team, with user documentation from all existing CSCIs, should prepare an integration plan to make the necessary connections between each CSCI. A similar process should be made for each HWCI developed.

During this integration process, the testing team should create a test plan of the CSCI/HWCI system that tests to ensure the proper connections have been made.

### 5.10.2 Performing CSCI/HWCI Integration and Testing

Each tester should follow the test plan and document the results of each test. As each connection is tested, it should be checked off if it passes every test or marked for revision or not.

### 5.10.3 Revision and Retesting

The development team responsible for the CSCI/HWCI marked for revision should review the test results and implement corrections as necessary. Once all corrections are made, the CSCI should be subjected to another round of testing.

### 5.10.4 Analyzing and Recording CSCI/HWCI Integration and Test Results

Integration test plans and results should be archived to allow the development team time to analyze them between each build in order to determine what can be improved during the development of the next build.

## 5.11 System qualification testing - To Be Updated

### 5.11.1 Independence in system qualification testing

System qualification testing will be performed by an independent testing team contracted for the purpose of ensuring that all the requirements for the build are correctly implemented.

### 5.11.2 Testing on the target computer system

Qualification testing should occur on the hardware designed by the hardware team.

### 5.11.3 Preparing for system qualification testing

The build should be installed on the hardware designed by the hardware team, as well as have rudimentary user documentation available to allow the testing team to utilize each requirement designed. The testing team should utilize the documentation in order to design a test plan for each tester to run through.

### 5.11.4 Dry run of system qualification testing

A dry run of the test plan should be performed to ensure that the documentation given is correct and that the test plan can be implemented.

### 5.11.5 Performing system qualification testing

Each tester should follow the test plan and document the results of each test. As each requirement is tested, it should be checked off if it passes every test or marked for revision or not.

### 5.11.6 Revision and retesting

The development team responsible for the system environment being tested that is marked for revision should review the test results and implement corrections as necessary. Once all corrections are made, the system should be subjected to another round of testing.

### 5.11.7 Analyzing and recording system qualification test results

Qualification test plans and results should be archived to allow the development team time to analyze between each build in order to determine what can be improved during the development of the next build.

## 5.12 Preparing for software use

### 5.12.1 Preparing the executable software

The software, once installed on the target system and has passed full system testing, should not be modified so that the team can guarantee that the system is in a working condition when the system is shipped to the client at the end of each build.

### 5.12.2 Preparing version descriptions for user sites

As the development team completes each build, a version description of the build should be created that summarizes the new features and changes in the build.

### 5.12.3 Preparing user manuals

A user manual that documents the usage and function of each feature in the build should be written and included with the system shipment to the client. The foundations of the manual should be complete and ready for delivery with Build 0, and updated with the delivery of each subsequent build.

### 5.12.4 Installation at user sites

The hardware team is responsible for documenting the various hardware connections, de-installing the system at the development site, preparing the hardware for shipment, and re-installing the system at the client site.

## 5.13 Preparing for software transition - To Be Updated

NA

## 5.14 Software configuration management - To Be Updated

### 5.14.1 Configuration Identification

Configuration management, utilizing a git repository, will utilize the git-flow method [3].

### 5.14.2 Configuration Control

The git-flow method [3] will be used to control branch usage and naming.

### 5.14.3 Configuration Status Accounting

As major features (as described in the git-flow model and identified by the development teams) are incorporated into the development branches, the configuration management team is responsible for reviewing the changes and merging the changes. Any changes from the development branch must have associated documentation submitted with the merge.

### 5.14.4 Configuration Audits

Once a month, the configuration team is responsible for checking that all development and release branches and their associated documentation are up to date and follow proper naming standards.

### 5.14.5 Packaging, Storage, Handling, and Delivery

Once each build is delivered, the code in the release branch will be made available to the client.

## 5.15 Software product evaluation - To Be Updated

### 5.15.1 In-process and final software product evaluations

As part of the monthly project status reports, the project manager should include an evaluation of the current state of the software project.

### 5.15.2 Software product evaluation records, including items to be recorded

The evaluation should include a list of currently implemented features as well as as features that are currently being worked on. In addition, relevant results from testing should be assessed and included in the evaluation if relevant.

### 5.15.3 Independence in software product evaluation

Testing results used to generate the evaluation report are generated by independently contracted teams, as described in sections 5.9.1 and 5.11.1.

## 5.16 Software quality assurance - To Be Updated

### 5.16.1 Software quality assurance evaluations

As part of the monthly project status reports, the project manager should include a record of the implementation status of the requirements of the project.

### 5.16.2 Software quality assurance records, including items to be recorded

Included in the evaluation record are the results from CSCI and system qualification testing as it pertains to the status of requirement implementation.

### 5.16.3 Independence in software quality assurance

Qualification results are generated by independently contracted teams, as described in sections 5.9.1 and 5.11.1.

## 5.17 Corrective Action

A record of all changes made during development of the Kaleidoscope Project shall be maintained as per company policy

## 5.18 Joint Technical and Management Reviews

### 5.18.1 Joint Technical Reviews

Technical reviews shall be performed during the development of the Kaleidoscope project for each development environment (PC and Sun) at least every 6 months.

### 5.18.2 Joint Management Reviews

Management reviews shall be performed during the development of the Kaleidoscope project for each development environment (PC and Sun) at least every 6 months.

## 5.19 Other Software Development Activities

### 5.19.1 Risk Management

Risk management may be carried out through risk assessment and risk mitigation. Risk assessment may be performed to quantify risk through various levels of risk using risk cube display and risk likelihood/criticality tables. Risk mitigation can be achieved through the following:

* Intensified technical and management reviews of the engineering process
* Special oversight of designated component engineering
* Special analysis and testing of critical design items
* Rapid prototyping and test feedback
* Consideration of relieving critical design requirements
* Initiation of fallback parallel developments

These risk management assessment and mitigations strategies were pulled from Systems Engineering Principles and Practice Textbook.

Some assumptions were made during the risk assessment process:

* Our company is assumed to be a new contractor for the KTS project.
* The software being developed for this project is assumed to be of moderate complexity.

The program has identified the following risks:

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Risk Factor (Pf/Cf) | Risk Factor Justification | Mitigation Steps |
| Developing the wrong user interface | 0.72 (0.63/0.23) | The KTS project represents the development of a brand new system that relies heavily on user interfaces. Should the UI fail to meet requirements, usage of the system will be heavily impacted as a large part of system utilization is reliant on the UI. In addition, the details of the UI are not documented in the SOW. While the potential rework of the UI to better address client requirements can be done quickly, the technical impact will be moderate due to the heavy reliance on the UI for performance. | A Customer Review Board, with customer representatives, should be created in order to help advise the development team on the functionality required in the UI, and to report back to the client on the status of the preliminary design. By solidifying UI design early in the design life cycle, the client will have an opportunity to review the requirements list and provide corrections where needed. |
| Unsuitable Organizational Structure | 0.86 (0.63/0.63) | The current build schedule aims for 100% personnel utilization, shifting team members around aggressively in order to finish each build as quickly as possible. There is a risk that such an aggressive schedule could result in project delays due to shifting priorities and potential rework necessary. | If possible, it may be prudent to build in some slack time between development builds in order to decrease the impact of any schedule delays. In addition, the development team should prioritize important features in order to deliver as much functionality as possible within the given time frame. |
| Excessive Requirements Volatility | 0.72 (0.63/0.23) | The SOW leaves a lot of requirements open to interpretation. We are proceeding with the assumption that the client has given us the authority to make design decisions, but there exists a risk that the vision and design produced by the development team does not match that of the client. Such mismatches of requirements can result in a small reduction in technical performance but a moderate cost and schedule increase due to rework. | The development team should assume the authority of making design decisions, but should reach out to the client to see if a customer representative is available to provide requirements feedback. In addition, the CRB assembled should help review the requirements list as necessary. |
| Shortfall in externally furnished components (software) | 0.85 (0.37/0.77) | The KTS project relies on various pieces of COTS software for critical parts of the software infrastructure such as communications. While the COTS is fairly reliable due to being widely used and most of the complexity has been abstracted away, failure of such COTS components can spell disaster for the project, both in terms of technical performance as well as the time needed to address all of the issues. The cost of such a failure would also be moderate as the team would either need to search for replacement software or provide a custom fix to the problem. | The COTS software utilized by the KTS project should be evaluated as early as possible in the development cycle to ensure the advertised functionally exists and can be used by the project. Any issues with any component of the COTS software, should they exist, can be discovered early enough to determine the appropriate remediation steps. |
| Shortfall in externally furnished components (hardware) | 0.51 (0.23/0.37) | The developers will rely upon COTS developed hardware for all hardware used in the project. However hardware used will be standard PC and server equipment. Failure to acquire these COTS devices from one vendor may be easily bought from another resulting in slight schedule slip and possible increase in price to have hardware delivered in a shorter time frame. | Hardware used by the KTS should be evaluated as early as possible in the development cycle to secure a vendor as early as possible. Possible second and third vendors may also be secured to order devices from to ensure that if one vendor does not deliver on time, other devices may still arrive. |
| Shortfall in initial LOC estimates | 0.69 (0.37/0.5) | Given that the schedule is based off of these initial LOC estimates provided by the SOW, if these estimates are off, they would have a significant impact on the schedule leading to large schedule slip and missing possible milestones. Cost may also be affected by this. However, the existing design would remain and may only result in a moderate increase of complexity. Since software components will be interacting with each other within the KTS, software components are dependent on each other. | Slack may be given to the schedule to account for such a scenario, we would want as large a schedule as possible to give the most development time possible anyway. We would have to accept that this risk may occur and track a LOC metric to see if this risk begins to manifest itself. |

Along with the identified risks are the expected risks of any project of personnel leaving the project, developers adding excess functionality to the project not within the original scope (gold plating), and the addition of added requirements or features to the KTS as the project progresses by the customer. These risks are inherent to any project and will be dealt with by adjusting cost and schedule as appropriate.

### 5.19.2 Software Management Indicators

N/A

### 5.19.3 Security and Privacy

Security of the system shall be maintained through training of software developers and management. Users of the system will receive security guidance and best practices upon delivery of the system. Enforcement of least privilege of the system will keep security breaches damage to a minimum. All security policies will be documented as per company policy. Outside connection to the system will be enforced with multi-factor authentication.

All user data generated by the system will be stored within the TMS and accessible only by qualified instructors and company personnel who work on the project. Users may access their own data but will not access other user’s data.

### 5.19.4 Subcontractor Management

N/A

### 5.19.5 Interface with Software IV&V Agents

N/A

### 5.19.6 Coordination with associate developers

N/A

### 5.19.7 Improvement of Project Processes

A record of all improvement made to procedures and processes during development of the Kaleidoscope Project shall be maintained as per company policy.

### 5.19.8 Other Activities

N/A

[2] Systems Engineering Principles and Practice

# 6. Schedule and Activity Network - To Be Updated

TBD- the following is some preliminary work to help with future development of a schedule.

The project is broken up into three deliveries. Each build is to be developed at the contractor's facility, deinstalled, shipped, and re-installed at the operational site.

To estimate the time to develop, we are assuming that the average LOC/day for the production team is 15 and 160 hours/SM. Time for labor was calculated with a 90% chance of meeting the schedule.

###### 

|  |  |  |
| --- | --- | --- |
| **Build** | **PC Environment CSCIs (LOC)** | **Sun Environment CSCIs (LOC)** |
| 0 | Total LOC: 4500  SM: 15  TL: 7.40 | Total LOC: 6000  SM: 20  TL: 8.14 |
| 1 | Total LOC: 3700  SM: 12.34  TL: 5.31 | Total LOC: 9500  SM: 31.67  TL: 9.49 |
| 2 | Total LOC: 3800  SM: 12.67  TL: 6.99 | Total LOC: 4000  SM:  TL: 7.11 |

Table 4- Build Estimates

# 7. Project Organization and Resources

## 7.1 Project Organization - To Be Updated

Due to the nature of the project, the organization will be set up for Projectized Management. A single program office will oversee the contract, and the two development teams, each responsible for one of the main subsystems. A Projectized Management structure makes sense, since there will probably be a single systems engineering team for both subsystems, as well as an independent QA team.

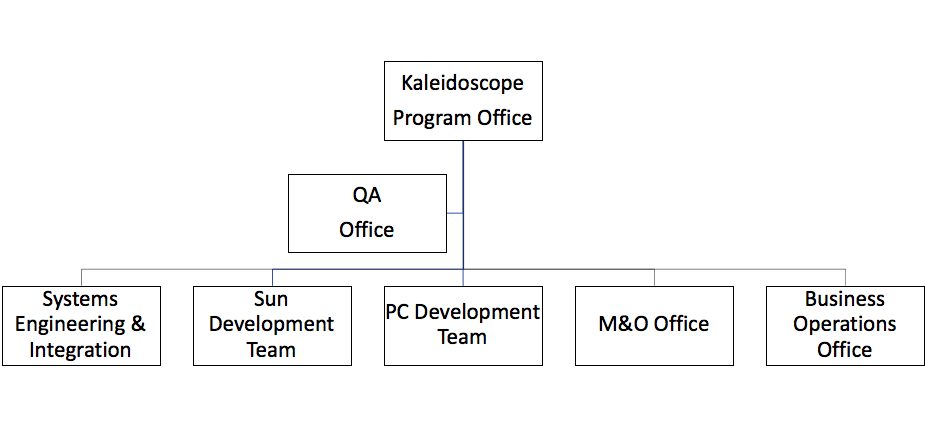


Figure 1: Project Organization

## 7.2 Project Resources - To Be Updated

The contractor has set aside $100,000.00 of capital funds for the computer aided software development environment (CASE tools.)

The following COTS software packages will be used for the Sun and PC CSCIs.

|  |  |
| --- | --- |
| **Sun** | **PC** |
| Unix | Windows XP |
| Sybase DBMS | Communications |
| Communications |  |
| Print Driver |  |
| Disk Controller |  |

Table 5: COTS Software Packages

# 8. Notes

NA

# Appendixes

NA