**SpartaGold:**

**An SJSU Cryptocurrency**

|  |
| --- |
| A Project Report Presented to  The Faculty of the Computer Engineering Department |
| San Jose State University In Partial Fulfillment Of the Requirements for the Degree Bachelor of Science in Software Engineering and  Bachelor of Science in Computer Engineering |

|  |
| --- |
| By |
| Art Tucay Jr., Benjamin Roppiyakuda,  Mojdeh Keykhanzadeh, Paul Portela, Phi Le |
| 09/2014 |

|  |
| --- |
| Copyright © 2014 |
| Art Tucay Jr., Benjamin Roppiyakuda,  Mojdeh Keykhanzadeh, Paul Portela, Phi Le |
| ALL RIGHTS RESERVED |

|  |
| --- |
| APPROVED FOR THE COLLEGE OF ENGINEERING |
|  |
| Dr. Thomas Austin, Project Advisor |
|  |
| Professor Keith Perry, Instructor |
|  |
| Dr. Xiao Su, Computer Engineering Department Chair |

**ABSTRACT**

**SpartaGold: an SJSU Cryptocurrency**

By Art Tucay Jr, Ben Roppiyakuda, Mojdeh Keykhanzadeh, Paul Portela, Phi Le

A digital currency has to establish authenticity, security, and non-repudiation in order for it to be considered a viable currency. Credit cards, debit cards, and PayPal transactions are all considered widely used centralized digital currencies. As digital currencies expanded, decentralized currencies connected through peer-to-peer networks emerged, such as Litecoin, Primecoin, and the very popular Bitcoin. These decentralized digital currencies need cryptographic user-authenticated protocols for transactions to provide reduced transaction costs in comparison to centralized digital currencies.

The objective of this project is to build a functioning decentralized digital cryptocurrency for the San Jose State University campus. This cryptocurrency, dubbed “SpartaGold”, will be tradable and spendable among SpartaGold users. SpartaGold will utilize digital signature techniques to provide high security for user transactions.

Cryptocurrencies currently in circulation conduct transactions with high end security protocols. The SpartaGold project values security first and foremost as well. The SpartaGold project has the ability to increase the public’s knowledge concerning security and peer-to-peer networking, primarily within the SJSU campus.

Table of Contents

**Chapter 1 Introduction Page 1**

* 1. Project Goals and Objectives Page 1
  2. Problem and Motivation Page 1
  3. Project Application and Impact Page 2
  4. Project Results and Deliverables Page 3
  5. Market Research Page 3
  6. Project Report Structure Page 4

**Chapter 2 Background and Related Work Page 5**

* 1. Background and Used Technologies Page 5

2.2 State-of-the-art Page 6

* 1. Literature Survey Page 8

**Chapter 3 Project Requirements Page 14**

3.1 Domain and Business Requirements Page 14

3.2 System (or Component) Functional Requirements Page 17

3.3 Non-function Requirements Page 19

3.4 Context and Interface Requirements Page 20

3.5 Technology and Resource Requirements Page 22

**Chapter 4 System Design Page 28**

4.1 Architecture Design Page 28

* 1. Interface and Component Design Page 30
  2. Structure and Logic Design Page 32
  3. Design Constraints, Problems, Trade-offs, and Solutions Page 43

**Chapter 5 Project Plan and Schedule Page 50**

5.1 Project Team Page 50

5.2 Project Tasks and Schedule Page 50

**Chapter 6 Tools and Standards Page 54**

6.1. Tools Used Page 54

6.2. Coding Standards Page 58

6.3. Coding Guidelines Page 60

**Chapter 7 Testing and Experiment Page 64**

7.1 Testing and Experiment Scope Page 64

7.2 Testing and Experiment Plan Page 65

7.3 Testing and Experiment Results and Analysis Page 69

**Chapter 8 Conclusion and Future Works Page 94**

**References**

List of Figures

Figure 1. Send Signed Message .............................................................. Page 14

Figure 2. Receive Message and Give Merchandise.................................. Page 15

Figure 3. Send Message .......................................................................... Page 16

Figure 4. Validate Transaction................................................................... Page 16

Figure 5. See Balance .............................................................................. Page 17

Figure 6. Block Chain Architecture............................................................ Page 28

Figure 7. Send Message GUI Design ....................................................... Page 30

Figure 8. Transaction GUI Design ............................................................ Page 31

Figure 9. Mine GUI Design ………............................................................ Page 31

Figure 10. Peer-to-Peer Component Diagram .......................................... Page 32

Figure 11. Send Gold Sequence Diagram ................................................ Page 35

Figure 12. Receive Gold Sequence Diagram ........................................... Page 36

Figure 13. Validate Transactions Sequence Diagram .............................. Page 36

Figure 14. Class Diagram ......................................................................... Page 37

Figure 15. State Diagram…....................................................................... Page 38

Figure 16. Package Diagram…….............................................................. Page 39

Figure 17. Control Flow Diagram …......................................................... Page 40

Figure 18. Context Diagram………............................................................ Page 41

Figure 19. MVC Data Flow Diagram ......................................................... Page 42

Figure 20. V Model ................................................................................... Page 64

Figure 21. Test Plan ................................................................................. Page 65

List of Tables

Table 1. System Functional Requirements ............................................... Page 18

Table 2. Non-functional Requirements: Usability ..................................... Page 19

Table 3. Non-functional Requirements: Reliability .................................... Page 19

Table 4. Non-functional Requirements: Performance .............................. Page 19

Table 5. Non-functional Requirements: Supportability ............................. Page 20

Table 6. Non-functional Requirements: Security ...................................... Page 20

Table 7. Hardware Requirement for SpartaGold Wallet ........................... Page 22

Table 8. Hardware Requirement for Mining .............................................. Page 23

Table 9. Software Requirement for Users Computer ............................... Page 24

Table 10. Software Requirement Traceability Matrix ................................ Page 25

Table 11. Use Case 1 ............................................................................... Page 33

Table 12. Use Case 2 ............................................................................... Page 33

Table 13. Use Case 3 ............................................................................... Page 34

Table 14. Testing Levels .......................................................................... Page 66

Table 15. Test Tools ................................................................................. Page 67

Table 16. White Box Test Cases............................................................... Page 69

Table 17. Black Box Test Cases............................................................... Page 70

Table 18. Black Box Test Scenarios.......................................................... Page 71

Table 19. Functional Requirements Checklist........................................... Page 91

# Chapter 1:  Introduction

## 1.1 Project Goals and Objectives

The goal of the SpartaGold project was to provide San Jose State University with a digital currency that allowed students and staff to conduct transactions with economic and competitive transaction fees.  By simply downloading the SpartaGold Wallet, students and staff are able to exchange SpartaGold with others whenever they need to buy, sell, or give products.  A SpartaGold transaction fee also makes small transactions convenient as the fee no longer diminishes business profits.  Once SpartaGold proves to be a convenient and secure form of currency, the SpartaGold project will shift its goal to promoting the SpartaGold network to on-campus businesses and vendors.

## 1.2 Problem and Motivation

Transactions conducted in physical cash within SJSU are widely accepted, but hold many inherent problems: the loss of physical bills and coins is constant and irreversible, cash handling is open to theft and robberies, paper and metal coins are unclean and prone to spreading disease, and their physical weight and size become a burden to carry on a daily basis.  The use of credit and debit cards conveniently solves all of these problems, but banks charge users for this central service.  Banks handling credit card purchases are within the right to charge their users transaction fees, which ultimately affect both the consumer and the vendor, as vendors can pass these costs onto the consumers.

In order to reduce these transaction fees, the central bank must be removed.  Decentralized cryptocurrencies effectively solve this problem using peer-to-peer networks secured through public and private key signatures and a proof of work.  Decentralized cryptocurrencies already in existence - Bitcoin being a prime example - are difficult to obtain, are exposed to the global community, suffer from a volatile market, and discourage local purchases and spending.

The SpartaGold project solves these economical and engineering problems by creating a decentralized digital cryptocurrency that holds no cap, which reduces the volatility of the market and eases the process of obtaining new currency.  The locality of SpartaGold will also encourage spending, causing a more circulated currency.

## 1.3 Project Application and Impact

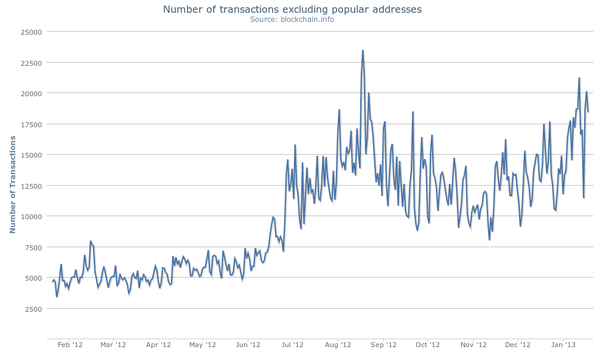
SpartaGold is a digital decentralized cryptocurrency targeted towards the SJSU community.  Students are able to sell and buy items from one another without worrying about transaction vulnerabilities and high transaction fees.  For example, students often raise money for their respective clubs by selling food and beverages on campus to the SJSU community.  With SpartaGold, the community can now sell and buy goods without cash or credit.  In addition, students would also be able to sell and buy books from each other using SpartaGold without worrying about actual cash handling during transactions. These examples, along with many other general applications, have provided SpartaGold users with a cheap and easy process for the exchange of goods.  In short, SpartaGold is applicable to any transactions between multiple peers, providing the SJSU community with low transaction costs.

## 1.4 Project results and deliverables

At the end of May of 2014, the SpartaGold project report was completed and submitted to the professor in charge of senior project planning.  The document contains the project scope, schedule, requirements, tools, and a test plan that was executed during the implementation phase.  A prototype was also delivered to the project advisor Dr. Thomas Austin during the summer to show the proof of concept.  These two deliverables guided the project towards the final steps of implementation of the SpartaGold Wallet with functional GUI, network connectivity, and cryptographic security. After strict alpha and beta testing with users, the SpartaGold project reached completion in December of 2014. Users of SpartaGold have generated their own Gold through the use of the SpartaGold Wallet, and have successfully transferred funds among each other.

## 1.5 Market Research

Many digital currencies compete for popularity and users, with Bitcoin at the top.  Bitcoin is currently the most popular digital currency with more than two million users.  Bitcoins have slowly been gaining acceptance.  Reddit, one of the largest social news networks, recently allowed Bitcoins as payment for their services and goods.  There are other digital cryptocurrencies trying to compete, each used to their own extent within their own niches.  The goal of this project was to reduce student costs and increase purchasing privacy by implementing a digital currency for SJSU student and faculty niche. The following chart shows the popularity of Bitcoin by graphing the number of transactions with Bitcoin by date:



## 1.6 Project Report Structure

This report documents the background of decentralized cryptocurrencies and how they stand in today’s economy and technological business, the needs and scope of the SpartaGold project, the architecture and implementation of the SpartaGold Wallet and network, and the tools used to complete these tasks. Each section is numerically labeled, with each subsection numerically labeled through an appendage to the super section. Tables and Figures are labeled and titled, and can be referred to through the Table of Contents. A collection of terms used throughout the document is listed at the end of Chapter 3: Project Requirements.

# Chapter 2: Background and Related Work

## 2.1 Background and Used Technologies

Cryptocurrencies are digital funds highly encrypted with various forms of information security, and are used to conduct online transactions among peers.  The concept is simple at first glance: User-A wants to buy goods from User-B and agrees to pay User-B 10 units of currency.  User-B accepts the payment and reciprocates by sending User-A the desired purchase.  This is an average transaction done with physical currencies around the world, using paper money or metal coins as the valued commodity.  However, when conducting online transactions, physical currencies are not available or easily provided.  Problems arise in sender authentication, receiver authentication, or third-party attacks.  User-A must authenticate that they are indeed sending their 10 units of currency to User-B, and User-B must authenticate that User-A is the person sending 10 units of currency.  User-B must also validate that User-A has 10 units of currency to spend, and User-A and User-B must reassure that a malicious attack from User-C cannot disrupt their transaction.  These safety protocols are implemented using cryptography, the practice of securely communicating with another entity while in the presence of other parties.

Peer-to-peer (often abbreviated as “P2P”) networks are a very important factor in cryptocurrencies that do not rely on a central authority.  Peer-to-peer file sharing is a data sharing process which connects users to other users, also known as “peers,” instead of users to a single server.  Instead, each user acts as both a client and a server, synonymously. Specific software is necessary to connect a peer to other peers.  Users send requests for downloads of specific files through this software.  Once a connection is established with other peers on the internet running similar software, the software will securely transport required files and messages from one peer’s computer to another, and then proceed with cryptographic protocols.  This is the backbone of current cryptocurrencies, as it allows users to assure funds are legitimate, and to transfer currencies without a central authority, or, in the case of online file sharing, a single server.

The development team of SpartaGold has utilized knowledge from a series of courses at SJSU to produce SpartaGold and the SpartaGold Wallet:

|  |  |
| --- | --- |
| **Course** | **Application** |
| Java Data Structures | Base code written in Java |
| Object-Oriented Design | GUI implementation |
| Information Security | Security for transactions |
| Computer Networks | Basis for the P2P network |
| Project Management | Team work allocation and timeline |

## 2.2 State-of-the-art

The most widely known and used decentralized cryptocurrency in the online market is, subjectively, Bitcoin.  Bitcoin’s boom in the virtual market began in 2009, and many users found the use of a purely digital, highly anonymous currency very alluring for online transactions.  Unlike some physical currencies, such as the US dollar or the European Euro, Bitcoin is a deflationary cryptocurrency; it gradually and naturally grows in value due to its limited supply.  Since Bitcoin has a limit of 21 million bitcoins, the supply is limited, so the price per coin rises.  While inflation is a typical problem among currencies with printable money, deflation also causes some problems within an economy. When a currency deflates, exchange rates begin to fall in relation to the deflated currency.  This causes holders of a deflated currency to hold onto their money instead of spending it.  In addition, those in debt with the current deflated currency will be hit with an even harder debt to fill, as the amount needed to pay off a debt rises with the value of the currency.

Other cryptocurrencies, such as Primecoin and Dogecoin, are still in circulation online.  The most common difference between digital currencies on the current online market is the proof-of-work, or the algorithm used to verify that a user has put the most effort in validating a transaction.  Primecoin uses prime chains as its proof-of-work system, while Dogecoin and Litecoin use a scrypt function.  These and other different digital currencies (also known as “Altcoins”), are briefly described in section 2.3 Literature Survey.

All decentralized cryptocurrencies rely on a peer-to-peer network.  Some support multiple peer-connectivity programs which store, send, and receive coins from other users. These programs come in multiple forms also known as “wallets”: a software wallet, a mobile wallet, a web wallet, or a paper wallet.  Web wallets store coins on a cloud and are accessed normally through web browsers, while paper wallets are physical print-outs of the encrypted string (the code behind each coin), generally condensed into a QR code.  Software wallets are stored on desktops or laptops and provide basic transaction functions using the peer-to-peer network, among other functionalities.  Mobile wallets are smaller versions of a software wallet used for mobile devices such as smartphones. Mobile wallets handle transactions using QR code scans and phone-to-phone transfers, likely through Bluetooth.

## 2.3 Literature Survey

Decentralized cryptocurrencies exist in many different forms. The following lists brief examples of past and currently existing cryptocurrencies and their unique traits:

**Bitcoin**

Bitcoin is the basis of many current digital currencies, and is the model of how a cryptocurrency can work without a central server.  The popularity of Bitcoin has expanded over time, and the value per coin has skyrocketed.  On October 5, 2009, one US dollar had an equivalent worth of 1,309.03 bitcoins, or BTC.  As of March 23, 2014, one US dollar has an equivalent worth of about 0.001803 BTC.  To put that in a better scale, 1 BTC equals roughly 554 USD (this price fluctuates by the second).  Bitcoin uses a peer-to-peer network as a solution to double spending by maintaining a list and creating a chain of transactions.  A bitcoin, at its core, is a chain of digital signatures created from the hash of the previous chain and the receiver’s public key, which is then digitally signed and added to the end of the chain.  The longest chain proves that a miner has spent the most amount of computing power on the hash-based proof-of-work, and thus they would be rewarded with a specified amount of bitcoin, after a SHA-256 proof-of-work function is done and broadcasted to all peers.  [6]

**Peercoin**

Peercoin, also known as PPCoin, is a cryptocurrency based on Bitcoin and its architecture.  PPCoin is currently the third largest cryptographic currency, with more than 21 million coins in circulation.  It uses many of the same components as Bitcoin, such as a peer-to-peer network, hashing functions, and a proof-of-work.  However, what makes PPCoin unique is its implementation of a newer transaction validation system titled “proof-of-stake”.  The idea behind proof-of-stake is to have miners validate transactions based on the amount of coins they are holding, instead of the amount of computational power they have, like Bitcoin’s proof-of-work.  For example, if a miner is holding 1% of the currency, then they will validate 1% of all transactions.  This has the effect of making it very expensive for dishonest miners to validate fraudulent transactions, easy for users to validate, and environmentally friendly as it does not require much power to validate.  The purpose of this system is to avoid a case where there are not enough miners connected to the network.  PPCoin also implements an unlimited and constantly growing amount of coins circulating through the system.  [5] [7]

**Primecoin**

Primecoin was created by the same creator(s) of Peercoin, under the pseudonym of “Sunny King.”  Primecoin’s proof-of-work sets it apart from Bitcoin, using prime chains as it’s proof-of-work system.  Primecoin accepts three types of prime chains as proof-of-work: Cunningham chain of the first kind, Cunningham chain of the second kind, and bi-twin chain.  Much if its architecture mimics that of its sister, Peercoin.  [5]

**Ripple**

Ripple is an online currency exchange system used to trade currencies with other currencies.  Ripple is a math-based cryptocurrency with secure and reliable transactions.  A unit of Ripple currency, dubbed “XRP”, is purchasable and tradable, much like other cryptocurrencies.  Peer-to-peer payments are also incorporated into the system, providing the same decentralized benefits of other cryptocurrencies.  Transactions are direct and distributed among all users of the Ripple protocol, and provides no counterparty risk or fees to any system administrator.  Ripple boasts zero inflation due to a set amount of XRP.  [2]

**Litecoin**

Litecoin is a peer-to-peer digital currency created by Charles Lee on October 13, 2011.  According to current trading markets, Litecoin now is the second most valuable digital currency in the world.  In general, Litecoin uses the same protocol as Bitcoin, with the exception of two improvements.  Instead of using SHA-256 to hash in the proof-of-work algorithm as Bitcoin, Litecoin uses scrypt.  This proof-of-work algorithm requires a large amount of memory allocated towards computational processing.  Litecoin also implements a shorter transaction confirmation time, so transactions complete at a faster rate. According to its development team, Litecoin’s transaction confirmation time is only 2.5 minutes (on average) in comparison to Bitcoin’s 10 minutes.  Litecoin is a limited currency, capped at a total of 84 million coins.  [9]

**Dogecoin**

Dogecoin is based on three cryptocurrencies: Bitcoin, Litecoin, and Luckycoin.  Dogecoin is named after the popular “doge” meme, and is used in a similar fashion as Bitcoin.  However, the creator, Billy Markus, strove to keep Dogecoin apart from Bitcoin’s reputation as a Silk Road currency.  Dogecoin uses the scrypt function instead of Bitcoin’s SHA-256, but shares similarities in its encryption protocol. Dogecoin uses hashes of public keys as addresses, all beginning with the letter D.  There is no maximum amount of dogecoins, with approximately 5.2 billion coins created every year, making it an inflationary currency.  It is currently often used as a tipping system for generally well received content on social media sites.  [12]

**Namecoin**

Namecoin is a good example of the Bitcoin protocol’s adaptability.  Based on Bitcoin, Namecoin is not a currency in the traditional sense.  Instead, Namecoin acts as an alternative Domain Name System (DNS), controlling the bit domain outside the realm of the Internet Corporation for Assigned Names and Numbers (ICANN), which oversees all the top-level domains.  Users simply purchase bit domains for one Namecoin apiece, and the domain itself is added to the so-called “block chain,” or public ledger that keeps track of who owns which namecoins/domains.  Ownership of a name is based on the ownership of a coin, which is, in turn, based on public key cryptography.  The Namecoin network reaches consensus every few minutes as to which names have been reserved or updated.  Because Namecoin has value outside of the currency itself (in the form of domain name ownership), it may one day prove to be stronger and more valuable than other cryptocurrencies.  [1]

**Scrypt**

Scrypt is a cryptographic function used to make hardware attacks difficult.  Password-based key derivation functions, or password-based KDFs, were designed to be computationally costly and take a significant amount of time.  However, possible breaches still exist, such as parallel computing for brute force attacks.  Scrypt is a variation of a password-based KDF, but it requires an increased use of memory, making it memory-hard as well as password-based.  Cryptocurrencies such as Litecoin and Dogecoin use scrypt as a proof-of-work, requiring all work to be done within a system’s memory instead of the GPU, where other cryptocurrencies allocate work.  [10]

**Cybercash**

Cybercash was an electronic payment service designed to support e-commerce transactions.  The company was founded in 1994 by Daniel C. Lynch, William N. Melton, Steve Crocker, and Bruce G. Wilson.  Similar to PayPal, Cybercash was designed to ensure secure online transactions; it acted as a third party from a seller to a buyer.  Cybercash is based off of the Cybercash Wallet program; buyers had to install a software on their computer before any transaction could take place.  [2]

**Disadvantage of SpartaGold**

Disadvantages of using SpartaGold exist in the literal concept of a digital currency.  Any new currency is only as valuable as it is perceived.  If the majority of users of SpartaGold believe that the worth of a single SpartaGold is 1/100th of a US Dollar, then its value will drop.  This is why the Bitcoin market is so volatile; Bitcoins can be sold for over 500 US Dollars one minute, then 200 US Dollars the next.

# Chapter 3: Project Requirements

## 3.1 Domain and Business Requirements

#### 3.1.1 Process Summary Diagram

|  |
| --- |
| **send message.png** |
| **Figure 1.** Send Signed Message |
| Receive Message.png |
| **Figure 2.** Receive Message and Give Merchandise |

#### 3.1.2 Process Decomposition Diagram

|  |
| --- |
| **send_message.png** |
| **Figure 3.** Send Message |

|  |
| --- |
| validate_transaction.png |
| **Figure 4.** Validate Transaction |

|  |
| --- |
| see_balance.png |
| **Figure 5.** See Balance |

## 3.2 System Functional Requirements

|  |  |  |
| --- | --- | --- |
| **F-01** | The user shall be able to see their pending and past transactions. | Medium |
| **F-02** | The user shall be able to see their available balance. | High |
| **F-03** | The user shall be able to receive Gold from another user. | High |
| **F-04** | The user shall have the option to validate transactions. | Medium |
| **F-05** | The user shall have a ledger of all transaction records that have occurred in the SpartaGold network. | High |
| **F-06** | The user should be able to sort their transaction record by date, address, amount, sent, received, or pending. | Low |
| **F-07** | The user shall be able to send Gold to another user by inputting the receiver’s address and the amount in a text field. | High |
| **F-08** | The user shall be charged a transaction fee in SpartaGold which will be awarded to the validator. | High |
| **F-09** | The user shall be able to generate new random addresses for transactions. | Low |
| **F-10** | The user shall be able to sign a message with their private key. | High |
| **F-11** | The user shall be able to verify the signature of a received message. | High |
| **F-12** | The user should be able to generate a QR code of an address. | Low |
| **F-13** | The system shall provide anonymity to users’ personal information. | High |
| **F-14** | The system shall implement a cryptographic protocol to ensure secure one-on-one conversations. | High |
| **F-15** | The system shall implement a proof-of-work function to deter attacks and phony validations. | High |
| **F-16** | The system’s wallet shall request password verification for access to its features. | Low |
| **F-17** | The system’s wallet shall provide a new address when requested. | Medium |
| **F-18** | The system should allow users to transport their wallet addresses or private keys through print or mobile devices. | Low |
| **F-19** | The system shall allow users to mine through a set of console commands or buttons. | Medium |
| **F-20** | The system shall broadcast pending transactions to others connected to the network. | High |
| **F-21** | The system’s wallet shall still operate with limited functionality when offline by disabling transaction sending, transaction receiving, and ledger synchronization. | Medium |
| **F-22** | The system shall provide validators with a specified amount of SpartaGold when a block chain is discovered. | Medium |
| **F-23** | The system shall provide validators with a specified amount of Gold allocated from the transaction as a transaction fee when a block chain is discovered. | Medium |

**Table 1.** System Functional Requirements

**3.3 Non-functional Requirements**

**Usability**

|  |  |  |
| --- | --- | --- |
| **N-01** | The system shall let users complete a transaction within 3 clicks. | Medium |
| **N-02** | The system wallet’s UI shall be visually appealing (tested with a focus group), efficient by strictly following Fitts’ Law, and provide fast navigation. | High |
| **N-03** | The system wallet should not provide more than 3 buttons per page (File and Settings bar, window bar not included). | Low |
| **N-04** | The wallet shall support standard English QWERTY keyboard. | High |

**Table 2.** Non-functional Requirement: Usability

**Reliability**

|  |  |  |
| --- | --- | --- |
| **N-05** | The system shall update users’ ledger automatically when connected to the network. | High |
| **N-06** | The system shall connect users using a peer-to-peer network during the wallet’s startup in less than 10 seconds. | High |
| **N-07** | The system should check for internet connectivity every 10 seconds when offline. | Low |
| **N-08** | The system should not complete a transaction process and cancel a broadcast to the network when an error (ex: lost connection, power failure) occurs mid-transaction. | High |

**Table 3.** Non-functional Requirement: Reliability

**Performance**

|  |  |  |
| --- | --- | --- |
| **N-09** | The system shall broadcast transaction information to the network in less than 5 seconds after a user initiates a transaction. | Medium |
| **N-10** | The SpartaGold network should be accessible as long as an internet connection is live. | Low |
| **N-11** | The system shall send accurate and unaltered information across the network when connected to the internet. | High |
| **N-12** | The system shall listen for updates to the ledger at most every 10 minutes. | High |

**Table 4.** Non-functional Requirement: Performance

**Supportability**

|  |  |  |
| --- | --- | --- |
| **N-13** | The system should support updates without introducing new bugs (tested with regression testing). | Medium |
| **N-14** | The system shall provide the same support and functionality of previous updates when using the most recent updated system. | High |

**Table 5.** Non-functional Requirement: Supportability

**Security**

|  |  |  |
| --- | --- | --- |
| **N-15** | The system shall not provide private key information to anyone who is not the private key’s original owner. | High |
| **N-16** | The system shall be protected from Man-in-the-Middle attacks at all times. | High |

**Table 6.** Non-functional Requirement: Security

## 3.4 Context and Interface Requirements

#### 3.4.1 Development

The system was designed to support users’ needs.  SpartaGold was developed strictly based on functional requirements.  The primary focus was to provide users safe, secure, and efficient transactions.  In addition, the system was developed with simplicity in mind, as SpartaGold’s target user base spans all branches of education.  Developing this system became a multi-stage process. First, the main focus involved implementing the requirements and using them as guidelines in prototyping a system.  This was done by developing simple components via functions to carry out individual tasks such as identifying users, carrying out a transaction between two parties, signing a transaction, and more. Stage two in the development process focused on testing each component to verify they functioned as intended.  After each component was tested, the top level design was assembled and made available for university students.

#### 3.4.2 Testing

Initial prototypes of the system were tested by the developers.  The developers recruited beta users and users then scouted for changes and made it such that the system was usable to the public.  All testing was conducted by the development team, SJSU Students, and SJSU faculty. Suggestions were submitted via email or communicated directly to the team for further improvement.

#### 3.4.3 Deployment

Since the system operates on a peer-to-peer network, all users were required to download the SpartaGold Wallet in order to perform transactions.  The SpartaGold Wallet, after being rigorously tested by the developing team, was deployed throughout San Jose State University’s campus, for students and faculty members to use.  Once popularity grows, implementation and marketing will proceed for on-campus vendor trade and exchange.

#### 3.4.4 Interface

An initial prototype interface was implemented alongside the system.  Basic GUI design principles were taken into account in order to design an interface that was efficient for users.  This prototype was tested for usability; once finished, it became an initial version.  Once the entire initial version of the system was made available for the users, we obtained feedback and made changes accordingly.

## 3.5 Technology and Resource Requirements

#### 3.5.1 Hardware Requirements

**SpartaGold Wallet**

The SpartaGold Wallet does not require high level and expensive hardware.  Most desktop systems have enough processing power to operate the wallet and conduct transactions, assuming an internet connect is established. The minimum requirements to run the SpartaGold Wallet are as follows:

|  |  |
| --- | --- |
| **Component** | **Minimum Requirement** |
| CPU | 1GHz 32-bit (x84) or 64-bit (x64) |
| RAM | 1GB (DDR, DDR2, or DDR3) |
| HDD/SSD | 20GB (ATA or SATA) |

**Table 7.** Hardware Requirement for SpartaGold Wallet

**Mining**

CPUs and GPUs are used to solve a proof-of-work process in order to verify the amount of computational power used during validation.  However, CPU's are not used for mining anymore since they have relatively slow computational speeds compared to GPU, FPGA, and ASIC devices.  GPU's have the ability to perform fast computations with their multiple ALU's but come with a high cost.  The minimum requirements to mine for SpartaGold are as follows:

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum Requirement** | **Notes** |
| CPU | 1 GHz 32-bit (x84) or 64-bit (x64) | Needed to run the OS, not mining. |
| RAM | 1 GB (DDR, DDR2, or DDR3) | Needed to run the OS, not mining. |
| HDD/SSD | 20 GB (ATA or SATA) | Needed to store data of the OS, wallet software, and the block chain. |
| Motherboard | At least 1 PCI-E slot (4X, 8X or 16X) | More slots are recommended to increase the mining speed by using more than one card. |
| GPU | 1 GB RAM, 667 MHz clock speed | Clock speed is more important than the amount of memory. |
| PSU | ATX. Input 110/220VAC. Output +3.3, +5, ±12VDC. Power 750W | The system runs 24/7, so it is important to have a reliable PSU that meets all graphic card requirements. |
| Internet connection | 1.5Mbps | An internet connection must be stable. |

**Table 8.** Hardware Requirement for Mining

The introduction of FPGAs and ASIC devices, which offer much faster mining speeds in comparison to CPUs/GPUs, has put an end to CPU/GPU mining.  These devices consist of integrated circuits that can be configured by the customer.  They are programmed to do only one job, which can be allocated to mining for Gold, and thus provide a very fast mining speed.

#### 3.5.2 Software Requirements

|  |  |
| --- | --- |
| **Software** | **Required for** |
| Windows OS, Mac OS, Linux OS | Environment for development |
| Netbeans IDE, Eclipse IDE | Development of SpartaGold software |
| Java 7 | Creation, compilation, and execution of SpartaGold code |
| Internet Explorer, Firefox, Chrome, Safari | Browser view and testing of an optional website for SpartaGold |
| SpartaGold-developed wallet | Execution of transactions and ledger updates |
| Android SDK, XCode | Mobile development and testing |
| TCP/IP , HTTP ,HTTPS , FTP | Network System |

**Table 9.** Software Requirement for Users Computer

#### 3.5.3 Dependencies

Dependencies determine the order in which each task is performed and they expose bottlenecks in the entire development process.  The initial dependency for implementation was the design of the cryptographic protocol. Development started by creating different versions of the protocol until one was decidedly secure and efficient.  The next dependency was the design of the proof-of-work.  Different types of proof-of-work functions were analyzed from established cryptocurrencies to help determine a properly scoped proof-of-work for SpartaGold.  Another dependency was the need for a powerful computer. This computer helped during tests of SpartaGold’s proof-of-work function and the overall network.  The last dependency was the need for users.  Users were a critical necessity to the P2P network, as they provided feedback and overall interest and participation.  Users were needed to help test the UI and the network, and were recruited from students of San Jose State University.  These dependencies were dealt with at the beginning of the design process.

#### 3.5.4 Traceability Matrix

|  |  |  |
| --- | --- | --- |
| **Functional Requirements** | **Test Case ID** | **Design Element**  **(Figure ID)** |
| **F-01** | BT-04, BT-05 | 4.3 |
| **F-02** | BT-03 | 3.5, 4.3, 4.4, 4.5 |
| **F-03** | BT-01, BT-02 | 3.2, 4.8 |
| **F-04** | BT-08, WT-05 | 4.9 |
| **F-05** | BT-04, WT-02 | 4.1, 4.6, 4.9 |
| **F-06** | BT-05 | 4.4 |
| **F-07** | BT-01, WT-02, WT-04 | 3.1, 4.7, 4.11 |
| **F-08** | BT-08 | 4.9 |
| **F-09** | BT-06, BT-07 | 4.4 |
| **F-10** | BT-01, WT-06 | 4.1, 4.6, 4.7, 4.8 |
| **F-11** | BT-02, WT-06 | 4.1, 4.6, 4.7, 4.8 |
| **F-12** | BT-06 | 4.4 |
| **F-13** | BT-01, BT-06, BT-07, WT-05 | 4.1 |
| **F-14** | BT-01, BT-02, WT-01 | 4.1 |
| **F-15** | BT-01, BT-08, WT-01 | 4.9 |
| **F-16** | BT-01, BT-02, BT-04, BT-06, WT-03 | 4.4, 4.5, 4.6 |
| **F-17** | BT-06, BT-07 | 4.1, 4.4 |
| **F-18** | BT-06, BT-07 | none |
| **F-19** | BT-07, BT-08 | 4.9 |
| **F-20** | BT-01, BT-02, BT-08, WT-03 | 4.6 |
| **F-21** | BT-01, BT-02, BT-03, BT-04, BT-05, BT-06, BT-07, BT-08 | 4.1 |
| **F-22** | BT-03, BT-08, WT-06 | 4.6 |
| **F-23** | BT-03, BT-08, WT-06 | 4.6 |

#### Table 10. Functional Requirements Traceability Matrix

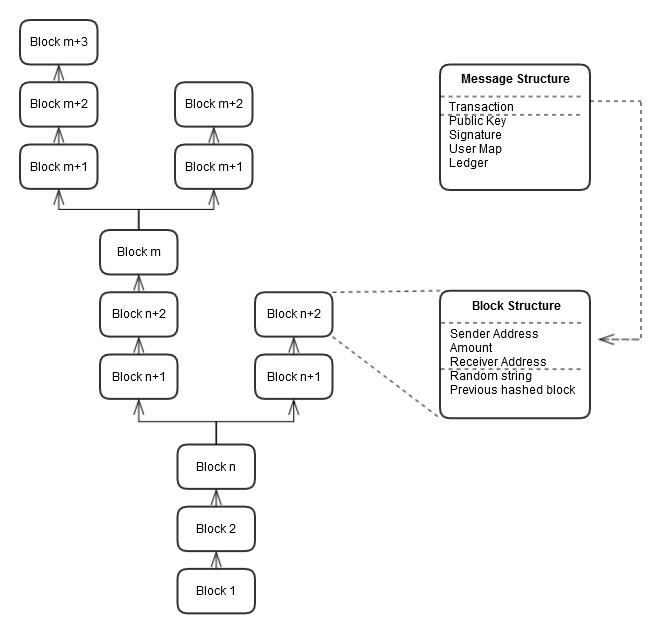
#### 3.5.5 Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| *address* | A randomly generated string of characters used to locate a specific connection in a peer-to-peer network |
| *Android SDK* | The Android SDK(software development kit) is a set of development tools used to develop application for Android |
| *ALU* | Arithmetic Logic Unit |
| *ASIC* | Application Specific Integrated Circuits |
| *ATA* | Advanced Technology Attachment |
| *ATX* | Advanced Technology eXtended |
| *block chain* | A long string of characters continuously edited and appended after each validation is finished |
| *Gold* | A unit of currency for SpartaGold |
| *CPU* | Central Processor Unit |
| *DDR* | Double Data Rate |
| *FPGA* | Field Programmable Gate Array |
| *GPU* | Graphics Processing Unit |
| *GUI* | Graphical User Interface |
| *HDD* | Hard Disk Drive |
| *IDE* | An integrated development environment |
| *Ledger* | A document housing all previous transactions and block chains |
| *Message* | Communication between users in a network |
| *Mbps* | Mega-Bit Per Second |
| *mining/mine* | The process of validating transactions using a proof-of-work which awards validators with a specified amount of SpartaGold |
| *OS* | Operating System |
| *PCI-E* | Peripheral Component Interconnect Express |
| *private key* | A secret string of characters not shared with others, used to sign messages to prove authenticity |
| *PSU* | Power Supply Unit |
| *public key* | A public string of characters shared with other users, used to encrypt data and unlockable only through the use of the user’s complimentary private key |
| *RAM* | Random Access Memory |
| *SATA* | Serial ATA |
| *SpartaGold* | A digital cryptocurrency |
| *SSD* | Solid State Drive |
| *Transaction* | An exchange of SpartaGold over the peer-to-peer network |
| *Validator* | A user who commits processing time to complete a proof-of-work and validate a transaction |
| *Wallet* | Software used to send, receive, store, and log Gold transmitted in relation to the user |

# Chapter 4: System Design

## 4.1 Architecture design

SpartaGold transactions are built around the block chain architecture. This block chain grows by one every time another transaction is completed. The longest chain is fundamentally shared among all validators, or “miners”, as a mining winner is decided based on the length of their chain.



**Figure 6.** The SpartaGold block chain architecture

A **block** contains string of five key parts: a sender’s address, an amount, a receiver’s address, a randomly generated string, and the previous hashed block in the chain. These parts are hashed together using SHA-1 and concatenated to the first 20 bytes in length.

The **block chain**, also known as the **ledger**, is a collection of sequential blocks. As branches of this chain emerge, discrepancies among who validated the transaction first arise. To relieve this contest, the validator with the longest block chain is awarded.

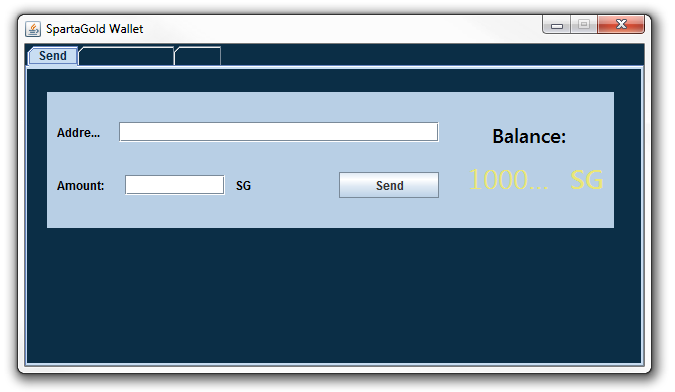
A **message** is a batch of files containing key parts for block chain processing, proof-of-work, and user balance updates. The files included in the message are the transaction file, a public key file, a signature file, a user map file, and a ledger file. The transaction file holds the sender’s address, the amount, and the receiver’s address in readable text. The public key and signature files are used in conjunction with the transaction file to validate the sender and receiver. The user map contains a map of all users in the system and their balances. Any tampering of this map can easily be overwritten by the copies stored on every other node in the network. The ledger file holds the block chain.

The **SpartaGold Wallet** automatically generates a user’s address based on their IP address. The Wallet is also the hub of all transaction activity and processing. Validation and proof-of-work is done within the “Mine” page, and Gold sending is done within the “Send” page.

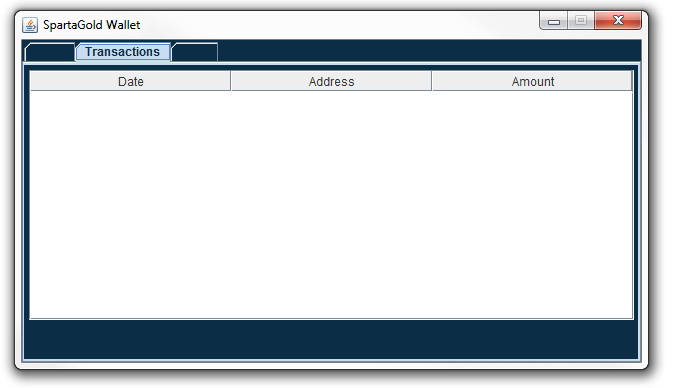
## 4.2 Interface and component design

#### 4.2.1 Interface Mock Ups

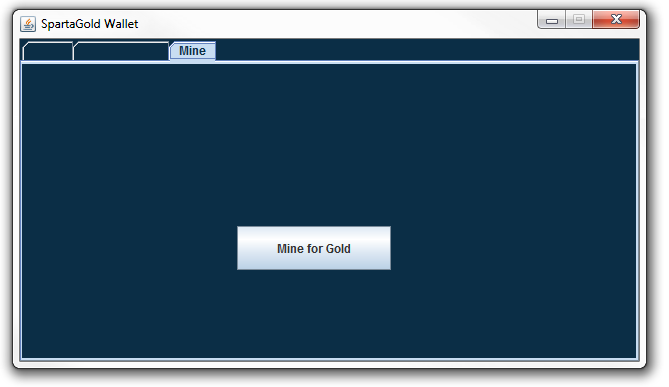
The SpartaGold Wallet has three main pages: “Send”, “Transactions”, and “Mine”. The “Send” page allows users to send Gold to specific users. The “Transactions” page allows users to view all past transactions. The “Mine” page contains the “Mine for Gold” button. The design is simple and concise, with minimal buttons. Each button is labeled with their function. The total balance of the user is displayed on the “Send” page of the Wallet.



**Figure 7.** “Send” page of the SpartaGold Wallet. This Send page includes a form with fields for the receiver’s address and the amount to send.



**Figure 8.** “Transactions” page of the SpartaGold Wallet. The Transactions page displays all past transactions as a list by date, address, and amount.



**Figure 9.** “Mine” page of the SpartaGold Wallet. The “Mine for Gold” button begins the validation process by listening for broadcasted messages and performing the proof-of-work. This process ends once the Wallet is closed.

#### 4.2.2 Peer-to-Peer Component Diagram

|  |
| --- |
|  |

**Figure 10.** A component diagram of the peer-to-peer network. Users are all interconnected, as there is no central server to designate transactions.

#### 4.3 Structure and logic design

The structure and logic design of the SpartaGold system can be conveyed by the UML diagrams: use cases, sequence diagram, class diagram, state diagram, package diagram and control flow diagram.

#### 4.3.1 Use Cases

The use cases below walks us through the main steps of the three main processes of the SpartaGold system. Each use case consists of participating actors, flow of events, entry conditions, exit conditions, and quality requirements. These use cases help us define and validate our requirements, which is a great first step towards a successful project.

|  |  |
| --- | --- |
| **Use case name** | **SendGold** |
| **Participating actors** | Initiated by Bob  Communicates with Alice |
| **Flow of events** | |  | | --- | | 1. Bob gets the address from Alice. | | 2. Bob states in the message the amount of Gold he has, the change he wants to receive, and the transaction fee. | | 3. Bob hashes the message, appends a timestamp to the resulting hash, and signs with his private key. | | 4. Bob gets the original message, the results from step 3, the timestamp, and sends everything as a single message encrypted with Alice’s public key. | | 5. Alice receives the message and decrypts the message with her private key. | |
| **Entry condition** | * Bob and Alice must have public and private keys |
| **Exit conditions** | * Bob has received an acknowledgement and his change from the transaction OR * Bob has received an explanation indicating why the transaction could not be passed. |
| **Quality requirements** | * Bob’s message is acknowledged within 10 minutes. * The selected message arrives no later than 5 seconds after it is sent Bob. |

**Table 11.** Use Case 1

|  |  |
| --- | --- |
| **Use case name** | **ReceiveGold** |
| **Participating actors** | Initiated by Alice  Communications with Bob and Miner |
| **Flow of events** | |  | | --- | | 1. Alice gives Bob her address. | | 2. Bob prepares the message and sends it to Alice. | | 3. Alice receives the message, decrypts it, and broadcasts it to the peers in the SpartaGold network for validation. | | 4. The Miner receives the message of Alice, validates it, updates the public ledger, and notifies everybody by broadcasting it. | | 5. Alice’s total amount of Gold gets updated and she then completes the transaction with Bob. | |
| **Entry condition** | * Alice must have an address * SpartaGold network must have people available to validate each transactions. |
| **Exit conditions** | * The public ledger has been updated. * Alice has received an acknowledgement regarding the validation. |
| **Quality requirements** | * Alice’s broadcasted message is acknowledged within 10 minutes. |

**Table 12.** Use Case 2

|  |  |
| --- | --- |
| **Use case name** | **ValidateTransaction** |
| **Participating actors** | Initiated by a Miner  Communications with Alice and the SpartaGold network |
| **Flow of events** | |  | | --- | | 1. The Miner is told about the broadcasted pending transaction of Bob and Alice. | | 2. The Miner begins the validation process by trying to solve a proof-of-work puzzle. | | 3. Around 10 minutes later, the Miner finds the answer to the puzzle and broadcasts it to the SpartaGold network along with the transaction message. | | 4. All peers in the SpartaGold network update and record a copy of the ledger. | | 5. The Miner is awarded the transaction fee and newly generated Gold from the SpartaGold network. | |
| **Entry condition** | * The Miner has a copy of the pending transactions and ledger. |
| **Exit conditions** | * The Miner’s total amount of Gold is updated to include the awarded Gold and transaction fee. |
| **Quality requirements** | * Proof-of-work puzzle must be computationally difficult to solve. |

**Table 13.**  Use Case 3

#### 4.3.2 Sequence Diagrams

The sequence diagrams below tie our use cases with our objects. They explain how the behavior of a use case is assigned among its participating objects. The first sequence diagram in figure 11 shows how the user will send an amount of SpartaGold to someone in the SpartaGold network. The second sequence diagram in figure 12 shows how the user will receive SpartaGold from someone in the SpartaGold network. Last but not least, in figure 13, the sequence diagram shows how a user will mine Gold in the SpartaGold network.

|  |
| --- |
| Send Bucks.jpg |

**Figure 11.** Send Gold through the network by entering information into the wallet.

|  |
| --- |
| Receive Payment1.jpg |

**Figure 12.** Receive Gold from another user by creating a personal address.

|  |
| --- |
| validate.PNG |

**Figure 13.** Validate Transactions

#### 4.3.3 Class Diagram

In figure 14, the class diagram shows the essential classes and the relationships among them. For example, a user is associated with a ledger because he/she reads from it, and also, the user is associated with messages because he/she writes them. The same idea goes for the miner. The miner reads and writes to the ledger. The miner also creates the blocks of transactions that make up the ledger by validating them before writing them to the ledger.

|  |
| --- |
| class diagram.png |

**Figure 14.** Class Diagram

## 

#### 4.3.4 State Machine Diagram

The figure below shows the state diagram of the SpartaGold node as it listens for a connection. At first, the node is listening for any nodes trying to connect to it. Once it notices that there is a node in the network trying to connect, it creates a connection and it is ready to handle incoming messages. When a message is received, it calls the appropriate methods and then it closes the connection.

|  |
| --- |
| uml_node_listening.png |

**Figure 15.** State Diagram - SpartaGoldNode Object

#### 4.3.5 Package Diagram

The classes used to build SpartaGold are organized by using packages. The figure below shows the hierarchy of the six packages.

|  |
| --- |
| uml_packages.png |

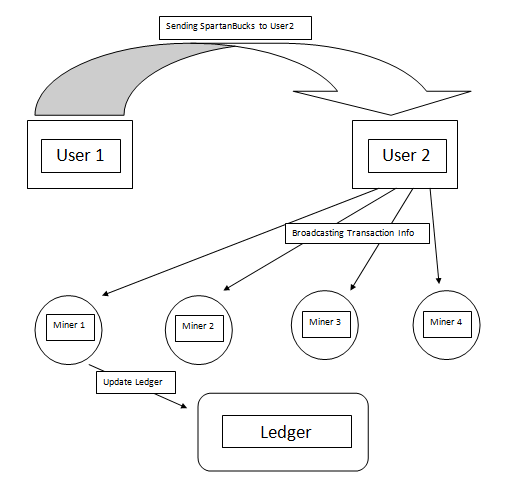
**Figure 16.** Package Diagram - Hierarchy of Class Packages

#### 4.3.6 Control Flow Diagram: Sending SpartaGold

|  |
| --- |
|  |

**Figure 17.** Control flow diagram of the process of sending SpartaGold.

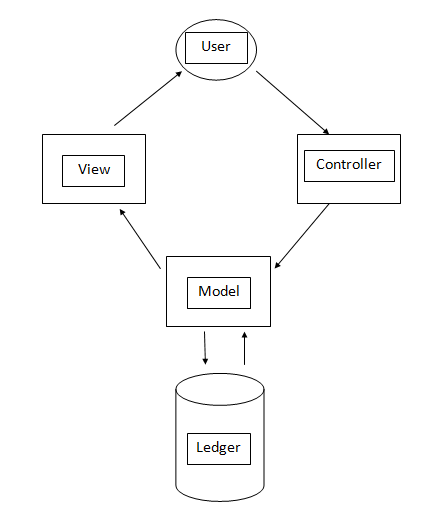
#### 4.3.7 Context Diagram



**Figure 18.** Context diagram for SpartaGold.

#### 4.3.8 Data Flow Diagram

For the SpartaGold wallet we will use the Model View Controller (MVC) pattern for our implementation. Below is the data flow diagram of SpartaGold wallet.



**Figure 19.** MVC data flow diagram for SpartaGold Wallet.

## 4.4 Design Constraints, Problems, and Trade-Offs and Solutions

#### 4.4.1 Design Constraints

**Society**

The system is intended for San Jose State University Students. Therefore, the system is available for such individuals for the purpose of this project. However, since the system is completely online and dependent on internet access, privileged users are able access their wallet and perform transactions anywhere.

**Environment**

As this system is purely software oriented, there are no environmental constraints. Privileged users are able to access their account and perform transactions anywhere around the world as long as they have an internet connection.

**Economic**

From a financial perspective, it was important to reduce the amount of overhead to implement the system. Overhead depends on the amount of computing power, as well as the size of each transaction. For example, assuming each transaction takes up 1KB as opposed to a transaction taking up 2KB, users’ hard drive could possibly fill up twice as fast, requiring more frequent hard drive purchases. In the case of the system, the cost will only increase as users will require an increasing amount of space to store transaction data.

Economically, the P2P network reduces the need for a central authority, which greatly reduces the costs of conducting transactions. Central authorities collect transaction fees to mitigate costs for services, insurances, and interests. With a decentralized network, the service and insurance is provided by the users for the users. Transaction fees are implemented only as incentive for users to validate quickly and efficiently.

**Hardware**

As the program itself is relatively small, each user of the network is required to keep track of all transactions. As transactions are rapidly accumulating, there is no definite requirement of space to hold all transactions. The amount of space will always be increasing.

* No environmental constraints as the system is implemented via online.
* Hardware and software constraints (the amount of data to be stored is only going to increase).

**Software**

Software-wise, SpartaGold is compatible with Linux, Windows, and MAC operating systems. Most users have one of the three operating systems as these are the most common to the public. The program is composed of Java code, so the latest JDK or JRE is highly recommended.

#### 4.4.2 Design Problems and Challenges

**Economic**

Deflation or inflation was one of the economic design problems. Cryptocurrencies are subject to the law of supply and demand. Putting a cap on the number of SpartaGold circulating causes deflation by increasing the value since the users keep hoarding the Gold instead of spending them. Placing a cap also puts another challenge of exhausting money one day in our way. Thereby, SpartaGold designers decided to not put a cap. However, this approach is vulnerable to inflation since every time more money is pumped into the system, the value of a SpartaGold decreases. Considering all factors, SpartaGold does not put a limit on the amount of Gold that may be in circulation; this is to provide a more economically secure system since the supplies never end and it is more appealing to the users.

Also, as money supply growth is deterministic, changes in money demand are reflected in the exchange value of the currency, raising or lowering the cost of producing the next coinbase as the protocol adjusts the difficulty up or down in response to the entry or exit of hashing power. So the exchange value of the mining determines the marginal costs rather than the other way round.

**Resources**

The SpartaGold system runs across a large network of computers and requires a considerable amount of electricity for miners to validate transactions. Therefore, users and miners need computing resources to help keep the system up and running. The miner is paying for electricity today for an award he will get at some point in future.

**Society and Environment**

The development of SpartaGold, like other digital currencies, will be an ongoing tug of war with public power and the domination system, driven by users' needs and the changing times. Leading resistance to digital currencies are banks with concerns that one day it might eliminate the existence of central banks.

**Hardware/Software**

For software, designing a secure protocol that is not vulnerable to double spending, DOS attacks and man-in-the-middle attacks was a very challenging problem. Also, using a cryptographic hash function for a digital wallet and the proof-of-work was another design constraint. An alternative implementation was to store coins in hardware wallets, but that was too challenging since it involved hardware which costs money.

**Safety**

The first challenge of safety was how to leverage the deflation and inflation of SpartaGold to assure the user that their money is secure. The first consideration was how much money the miners pump into the system each time they validate to assure the stability of the system. Also, designers considered how to prevent coordination of large pools of miners who were collaborating together to mine because it can be a security issue if the pool is too large.

**Reliability**

It is important for SpartaGold to be reliable so users can feel confident that their money will not go to waste and thus increase the success of the system. One important characteristic is making sure that the system is up for most of the time. This is crucial since we want users to feel confident that their transactions will be sent across the network and validated as well. Also, having the system cryptographically secure will increase its reliability. The system was implemented with cryptographic standards to assure components are not vulnerable to attacks.

#### 4.4.3 Design Solutions and Trade-Offs

**De-centralized Authority**

Instead of implementing a system based around a centralized authority, a peer-to-peer network was integrated. With a central authority, transactions would require a bank to conduct its transactions. In addition, the user does not need to keep track of their history of transactions as it would be stored through the bank. However, a major dilemma to this approach is non-anonymity; the bank will always be monitoring each customer’s funds - especially when they use a debit or credit card for a purchase. Also, it puts more overhead and responsibility on the people in charge as they will be responsible for keeping track of all the transactions. This increases the amount of space and computing power needed in order to keep everything running efficient for the rest of the users on the network. There is a much larger risk to a centralized authority as well. If the central server goes down in a worst case scenario, the entire network falls with it, becoming unavailable for users. As this approach is traditional, the problematic aspects revealed a decentralized network to be the optimal choice for SpartaGold. In a decentralized network, all data and validation of transactions are processed and saved by every user in the system. In order to achieve this, SpartaGold was designed to be a peer-to-peer network. A peer-to-peer network will allow for high anonymity for every user by hashing transactions and establishing a public/private key for any encryption needed. This removes responsibility from what would be a central source and distributes the responsibility to everyone in the network.

**Processor Speed**

As mentioned earlier, mining for SpartaGold efficiently requires a powerful computer with extreme computing ability. A user has multiple options, but this comes down to the user’s budget. Assuming that money and computing speed are directly proportional, more money invested in a faster computer will yield more efficient mining due to faster processing speed which directly correlates to more SpartaGold in a shorter period of time.

**Economic Solution**

There were two solution options: either put a cap on the amount of SpartaGold that can be found/generated in the economy, or allow for an infinite amount of Gold to be in circulation. A cap would gradually cause deflation on the currency, due to the limited availability of Gold. Just as real-world gold goes up in value due to a finite amount, SpartaGold will also increase in value. On the other end, creating a cap-less currency would consistently inflate the value of each Gold.

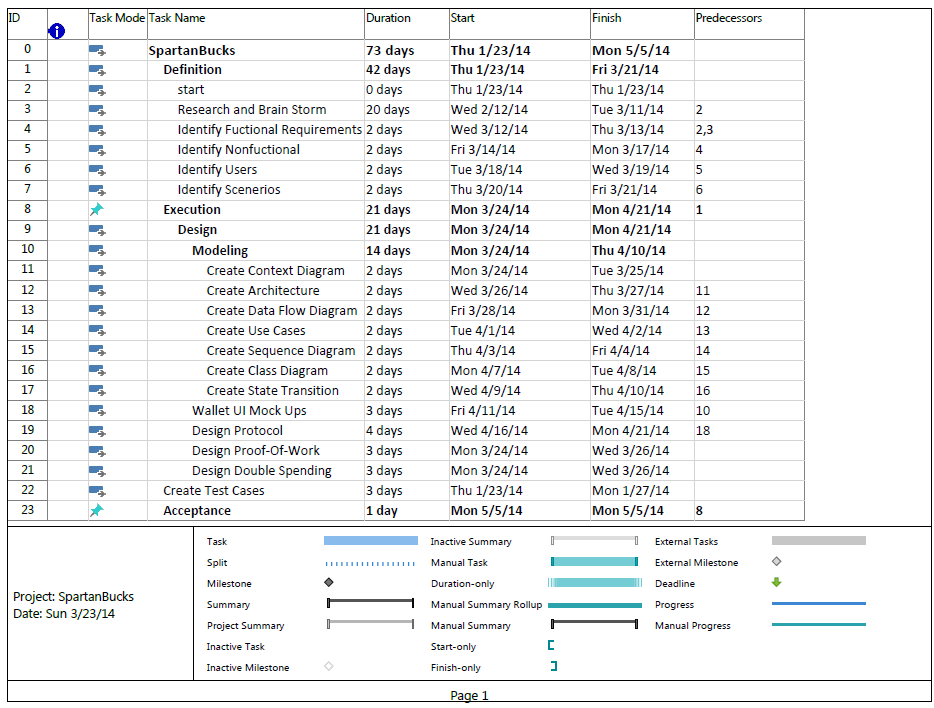
After some deliberation, the latter solution was implemented into the system. No cap was integrated, and the currency is expected to inflate over time. However, this is not necessarily a problem, as the amount of currency allowed in the system can be controlled to ease any difficulties with inflationary currency. In addition, avoiding the inherent problems with a capped currency was in our best interest. Capped currencies grow in value per unit, but this causes many holders of this currency to spend less, causing stagnation in the economy. Volatility arises from this limitation, and each unit of a capped currency explodes and diminishes in frightening spikes.

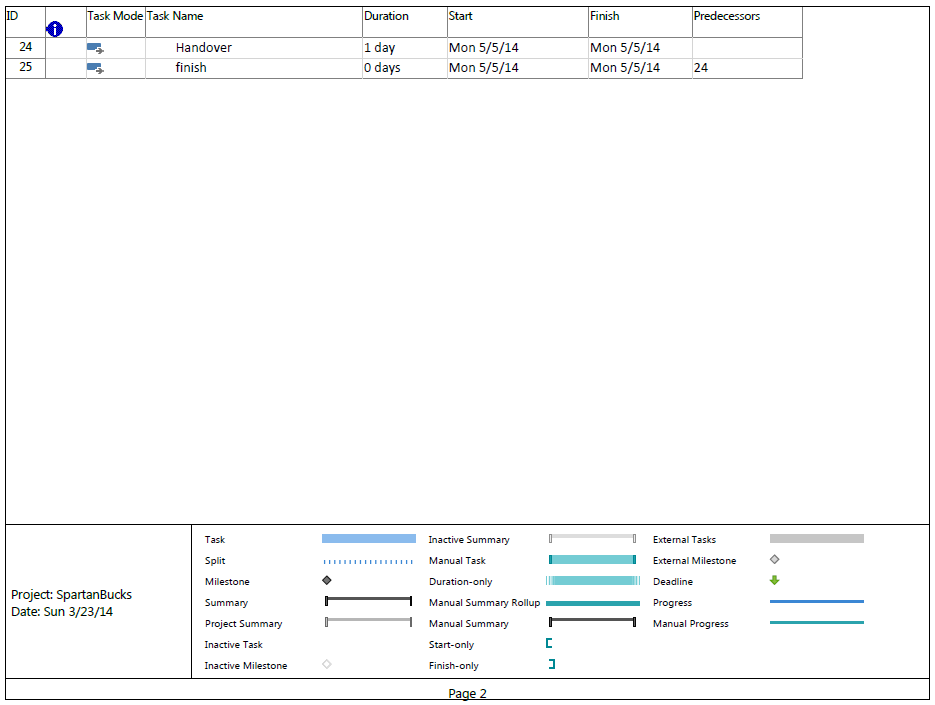
# Chapter 5: Project Plan and Schedule

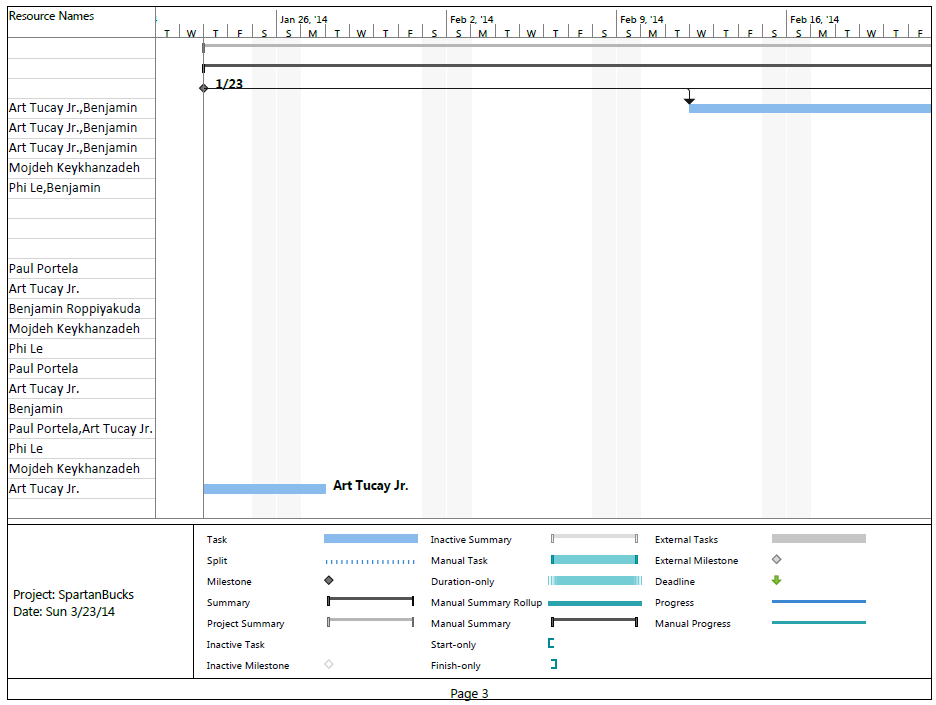
## 5.1    Project team

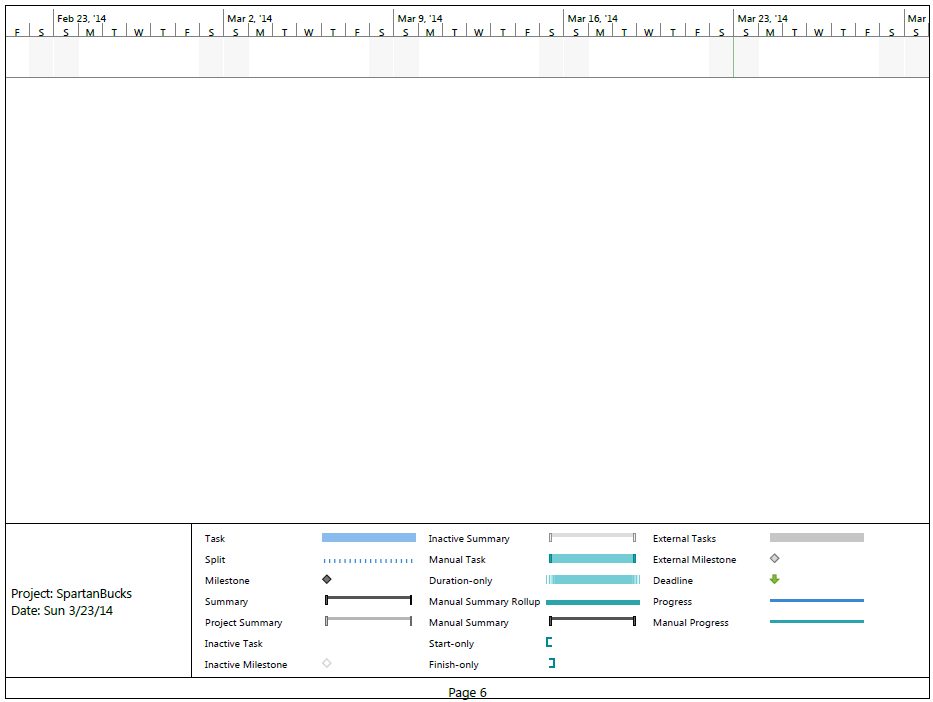
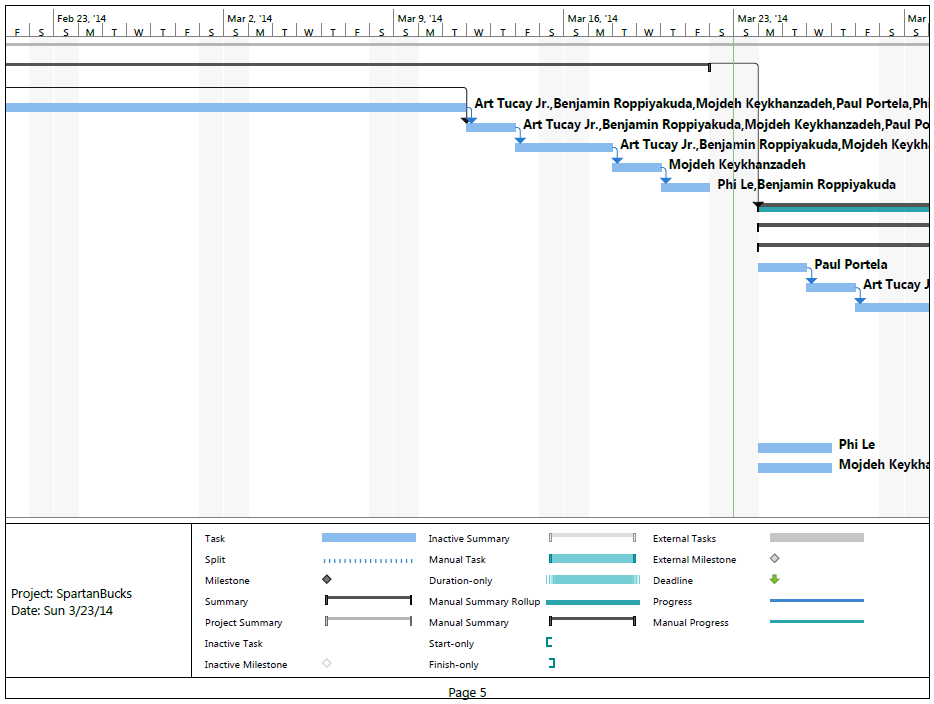
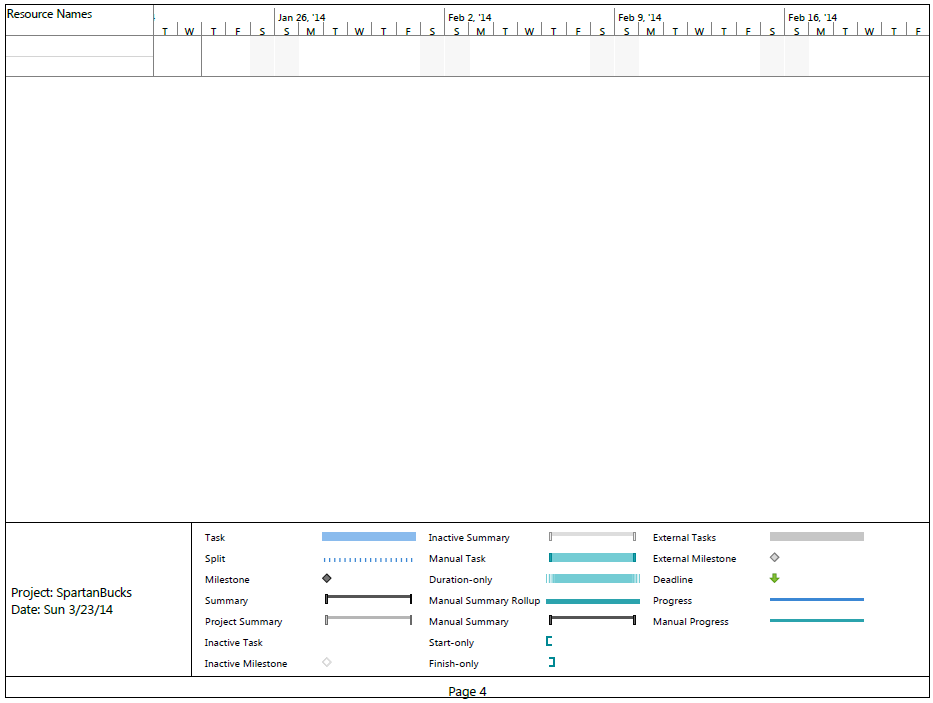
Our team consists of five San Jose State University students. Art Tucay Jr., Paul Portela, and Mojdeh Keykhanzadeh are working towards their degree in software engineering, and Benjamin Roppiyakuda and Phi Le are working towards their degree in computer engineering. Our project adviser is Dr. Thomas Austin, and our project manager is rotated among the entire team.

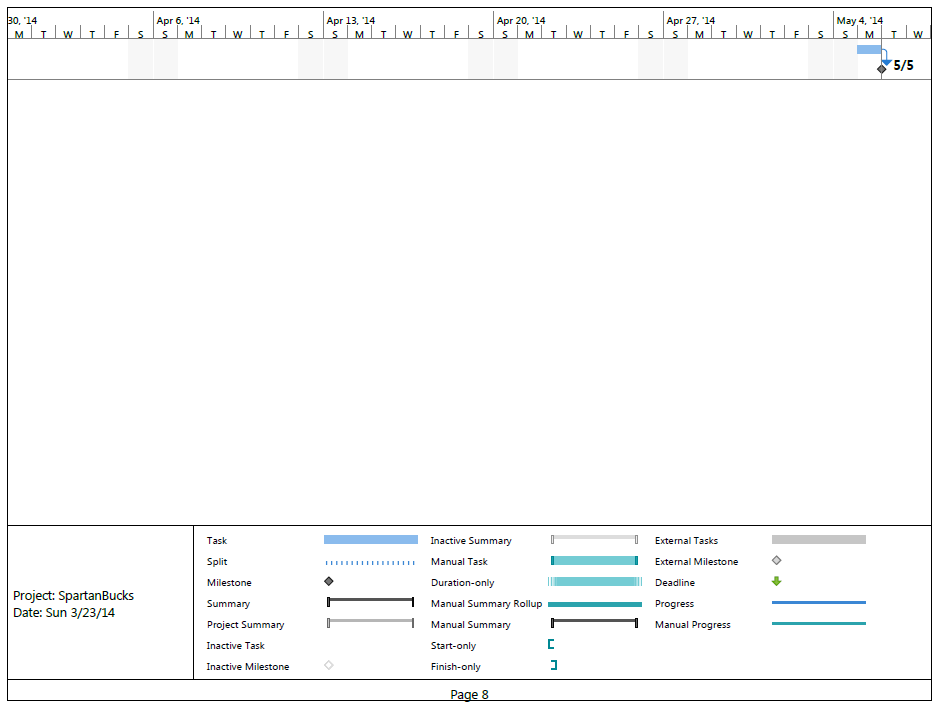
## 5.2    Project tasks and schedule











# **Chapter 6: Tools and Standards**

## 6.1.   Tools Used

#### 6.1.1    Modern Engineering Tools

|  |  |
| --- | --- |
| Java using Eclipse IDE | Java was used as the main developing language to implement SpartaGold’s backend and frontend. |
| GUI Programming | Java’s Swing library was used to implement the buttons, text boxes, and panels for SpartaGold’s wallet GUI. |
| Network Programming | Java Sockets and Socket Servers were implemented into SpartaGold’s peer-to-peer network to allow nodes to  send and receive messages. |
| JUnit | Java’s JUnit library was used for unit testing. |
| Microsoft Project | This management tool allowed for a comprehensible list of tasks and times which made work more manageable.  It also provided a detailed Gantt chart. |
| Google Docs | The group was able to work on documents as team and share them too. |
| Git | Version control was great for backing up code into the local machines. |
| GitHub | A remote repository allowed developers to backup and share code instantly. |
| Canvas | Turning in documents was never made any easier because Turnitin is built in and it constantly sends us notifications. |

#### 6.1.2    Techniques & Skills

* Solved the peer-to-peer development problem by learning and utilizing a framework that had the needed functionality.  Other solutions were also considered such as JGroups, TomP2P, JXTA, and BitTorrent, but they proved to be too comprehensive and had a steep learning curve.
* Considered different methods to deter spam on our peer-to-peer network.  Proof-of-stake is one of them and it works by using the users’ coins, but it’s relatively new and it hasn’t been fully certified.  Proof-of-work proved to be the best one because it is currently used in other systems like email and it works by having computers solve difficult puzzles.
* Implemented a SpartaGold website to help define what it is.  Other methods were also included like a “README.txt” and a tab in the GUI wallet called “About Us” that explains SpartaGold in detail.
* Implemented a command line interface (CLI) as an alternative to a graphical user interface (GUI) since CLI uses less memory than the GUI version, so it allows the system to run more efficiently.  Using CLI won’t make the system load additional libraries which are needed to load a GUI application.
* Made mining coins more flexible by providing users different ways of mining.  For example, a miner will only mine if there is a fee that he/she will receive.  Other methods were created by applying a Java interface for the mining scripts.

#### 6.1.3    Communication Variety

**E-mail**

Due to varying time constraints and schedules, e-mail became the most reliable and most often used form of communication among the team.  Email was used to schedule group meetings with the project advisor.  SpartaGold team members outlined specific tasks, goals, and notifications through e-mails among each other, and with the advisor.

**Biweekly face-to-face group meetings with advisor**

Verbal communication was the best way to communicate ideas among the team.  Every other week, SpartaGold team members met with the project advisor, Dr. Thomas Austin, to discuss achievements, objectives and goals, and task division.  Meeting notes were written down, and supplementary e-mails were sent before and after the meeting.

**Live document group editing**

Google Docs was used as a live document editing tool.  Assignments as well as the project report were created by one team member and then shared with others.  Everyone worked on their parts in their convenient time and collaborated together at the same time.  Everyone can see what others are working on simultaneously.  In addition, team members can make comments or talk to each other instantly through a chat window integrated in Google Docs.  When all the jobs were done, documents can be published to the web or downloaded as different formats such as MS Word or PDF documents.

**Instant messaging**

Communication was made easier by using Google’s chatting application.  Each person created a Google account and had access to the chatting application.  Team sessions were created where all team members chatted with each other and talked about the current duties and assignments.

**Text Messaging**

Sometimes it took long time for emails to be sent and received, so text messaging through phones was a faster way to reach other members.  Text messages were used occasionally when we wanted fast responses from others.

**Peer programming**

It was a good practice that two members worked together at the same workstation.  The driver was the one who wrote code, and the pointer was the one who reviewed the code, pointed out errors, or gave ideas/suggestions to improve the program.  The role between these two was switched frequently.

**GitHub**

GitHub was used to share current versions of code.  Each person would create their own branch to work on, and then at the end, someone would merge everything together to the master branch.

**Dropbox**

It is a file hosting service where one member uploaded files and shared with other members.  After installing, a folder called Dropbox was created, and users started saving their files there.  These files can be accessed through any computers, Dropbox website or Dropbox mobile application.  Anytime the user changes the folder, Dropbox will synchronize it automatically.  Other members who were shared the folder can access it as well.

**Team-created P2P file sharing system to test**

Using a remote repository made development of the P2P network more manageable.  For example, developers were able to create a P2P function and push it to the GitHub repository.  Then, testers were able to see the new update and start their testing.  A special package called testing was made to contain all the tests done.  This has been the method used to build other components of SpartaGold.

**6.2.   Coding Standards**

Coding standards focused specifically on how developers wrote code.  The following standards listed were used for the scope of this project and were important in assuring an accurate and desirable product.

**6.2.1   Coding Indentation**

Indentation is essential to code organization.  It drastically improves readability toward the developers.  The purpose of using indentation is to:

* Distinguish one or more control statements (i.e. any kind of loop or recursion)
* Distinguish one or more condition statements (i.e. if statement or )
* Distinguish a new scope (function/class/struct, etc.)

A standard indentation used was a tab.  Multiple tabs were used when nesting control statements or conditions.  As a tab is a consistent amount of space, it simplified implementing the code’s organization as it will not require pressing the spacebar multiple times.

**6.2.2   Inline Commentation**

Inline comments explain a particular section of code.  Comments could be used for explaining function definitions and how they work, particular algorithms, or any part of the code that may not appear clear and concise.

**6.2.3   Structured Programming**

Developers used structured programming.  The system was organized into packages and modules.  Each package and module is a top level view of major functionalities.  All modules were further divided into sub-functionalities. Each sub-functionality contained classes and functions pertaining to its operation.

**6.2.4   Classes, Functions, and Methods**

Programming classes, functions, and methods were reasonably sized to avoid complexity. They are reasonably sized in the sense that Java functions and methods are not long to make it less complex and easier to debug.  It is the developers’ advantage to have small functions and split up methods to smaller parts since it enhances readability and modularity.

**6.3   Coding Guidelines**

SpartaGold Developers adhere to java coding standards.  The aim was to complete a line of source code from reaching more than one line.  Java code was implemented with proper spacing within the code itself.  When a line of code exceeded the width of a single line of coding space, proper wrapping line techniques were used.

**6.3.1   Line Length**

Understanding simplicity as an essential factor in undertaking this project, developers made it a point to always aim to complete a line of source code from reaching more than one line.  In some situations developers were not be able to meet the requirement, so some lines were dealt with accordingly (see section 6.3.3).

**6.3.2   Spacing**

Developers implemented proper spacing within the code itself.  An example is as follows:

import java.io.\*;

public void example() {

    int a, b, c;

   a = 1;

   b = 2;

   System.out.println(“Hello world!”);

   c = a + b;

}

Multiple variable declarations on a single line were separated by a space.  Variable assignments on each side of the assignment operator were separated by a space.  Any operation involving variables, functions, and/or classes needed to have spaces in between each operator.

**6.3.3   Wrapping Lines**

When a line of code exceeded the width of a single line of coding space, the following was done accordingly:

-    Begin a new line after a comma

-    Begin a new line after an operator symbol

After starting a new line, it was indented to the opening of a parenthesis, an assignment operator, or a comment.  An example is as follows:

import java.io.\*;

public void example() {

int a, b, c, d, e, f, g;

// There are unused variable declarations but this

// example should demonstrate our standards

    a = 5;

    b = 10;

    c = a + b + a + b + a + b + a + b + a +

       b + a + b + a;

}

**6.3.4   Variable Declarations**

Variable declarations occupying multiple lines shall always be preceded by a data type.  The end of a line was defined as a standard page width (8.5 inches).

**6.3.5   Program Statements**

Individual programing statements were limited to one per line.  This includes anything from variable declarations to function / class definitions.

**6.3.6   Use of Parenthesis**

In order to ensure user friendly readability for the team and for others looking into the code, parentheses were included.  The parentheses were used in any arithmetic operations to distinguish the order of execution, even if the operations would execute correctly without them.

**6.3.7   Inline Commentation**

As mentioned earlier, inline comments improved program readability.  This would be particularly beneficial for someone who is not up to date with function definitions or sections of the project.  The comments provided an overview of a particular section of code, when required.  The inline comments shall take up no more than 25% of the entire code; any higher would make finding the actual code difficult.

Javadoc guidelines were used for inline commenting.  In general, the Javadoc standard for any multi-line comments is (“\\*....\*\”).  For single-line comment, “\\” is used.

**6.3.8   Coding for Efficiency and Coding for Readability**

Given the purpose of the project and what it entailed, the project was coded so that it would run as efficiently as possible.  In addition, although many updates and bugs were addressed as we proceeded, the code was relatively simple and easy to maintain.  This included minimizing the amount of objects needed to be changed when a minor modification was made.

**6.3.9   Reasonably Sized Functions and Methods**

The code did not contain an excessively large number of lines of code. Instead, functions and methods were organized by groups and separated into individual files.  The code was written in the way to perform specific tasks.  If they became too long, they would be broken down into subtasks which can be handled by new routines or methods.  This reduced large functions and enhanced readability.

# Chapter 7:  Testing and Experiment

## 7.1 Testing and Experiment Scope

During the software development cycle, testing was performed at numerous points as SpartaGold is similar to the V model below. Testing was also conducted at different levels to ensure the product was thoroughly tested and a quality product was delivered to the users. In addition, the different levels of testing were done systematically according to corresponding test plans. Since this project incorporated a five person team, resources were limited so tradeoffs had to be accepted. Coordination was accomplished by using a test management tool that kept track of test cases and bugs.

|  |
| --- |
| v model.PNG |

**Figure 20.** V Model

## 7.2 Testing and Experiment Plan

The testing for SpartaGold consisted of unit, integration, security, performance, acceptance, and beta test levels. Testing was done by an independent tester of the team who was in charge of creating test cases with the intent of breaking the system. However, the SpartaGold project was constrained to only five people so the development team helped with white box testing.

|  |
| --- |
| test plan.PNG |

**Figure 21.** Test Plan

#### 7.2.1 Test Levels

|  |  |
| --- | --- |
| **Unit Testing** | The goal of unit testing was to isolate each part of the program and show that individual parts were correct in terms of requirements and functionality.  These tests were done by the developers by writing test cases and documenting the outputs.  Then, the team leader reviewed the tests cases and outputs before accepting it as done.  The tester also analyzed the results and code coverage reports. |
| **Integration Testing** | The goal of integration testing was to determine if combined parts of the program functioned correctly together.  These tests were done by the tester and with the help of some developers.  Any errors were reported and tracked by the tester.  The tester reported immediately if any errors were preventing tests to run. |
| **Security Testing** | The goal of security testing was to determine any security issues.  The whole team tested the system, and a security specialist examined the system again to ensure the system met security requirements. |
| **Performance Testing** | The goal of performance testing was to determine any bottlenecks or performance issues.  The tester used a special software tools to simulate different loads in order to perform load testing.  The tester then reported on the system’s behavior resulted from the different loads. |
| **Acceptance Testing** | The goal of acceptance testing was to determine if the system met the intended specifications and satisfied the client’s requirements.  This testing was be done by the users with the help of the team leader.  The actual users were San Jose State University students. |
| **Beta Testing** | Beta testing is also known as “pre-release” testing.  SpartaGold held an open beta test version and distributed the software to anyone who wished to try out the system.  If errors did occur, the system automatically reported all errors to the development team. |

## Table 14. Testing Levels

#### 7.2.2 Test Tools

|  |  |  |
| --- | --- | --- |
| **Performance Testing** | JMeter | Used to run loads on the SpartaGold website |
| **Bug Tracking** | Google Spreadsheet | Tracked all bugs by creating a spreadsheet |
| **Java Unit Testing** | JUnit | Used to create unit tests for all Java classes |
| **Equivalence Class Partitioning** | ACTS | Created test cases for various input values of SpartaGold Java methods |
| **Model Based Testing** | Selenium | Used to create automated tests for the SpartaGold website |

**Table 15.** Test Tools

#### 7.2.3 Pass / Fail Criteria

The testing process shall be completed when all test cases meet system requirements. It will be the job of the tester to track the bugs and make sure they are assigned to a developer for fixing. The tester shall follow up with the developer for any progress until the bug gets fixed. After making sure all bugs have been taken care of, the tester shall report to the team leader for a final evaluation.

#### 7.2.4 User Stories

**US-01:** As a miner, I can selectively choose to validate specific transactions based on the amount offered by the sender of the transaction.

**US-02:** As a SpartaGold user, I am able to send a specific amount of my own SpartaGold to another user by inputting the receiver’s address and the amount I aim to send.

**US-03:** As a miner, I can allocate computational power towards validating a transaction in an attempt to earn SpartaGold.

**US-04:** As a SpartaGold Wallet user, I am able to view past transactions in the Transactions page, and sort this list based on date, label, amount, or address.

**US-05:** As a SpartaGold user, I can generate new addresses by clicking the “New Address” button, which returns a new and unique address for other users on the network to use to connect to me.

**US-06:** As a SpartaGold Wallet user, I can verify messages received from others to assure I am receiving Gold from the proper source.

**US-07:** As a SpartaGold Wallet user, I can sign messages to authenticate myself and prove to the receiver that I am who I say I am.

**US-08:** As a SpartaGold Wallet user, I am able to award a validation reward at a value of my choosing by specifying it within the Wallet settings.

## 7.3 Testing and Experiment Results and Analysis

#### 7.3.1 White Box Test Cases (Unit & Integration Test Cases)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Test Objectives** | **Pre-conditions** | **Priority** | **Expected Result** | **Actual Results** | **Defects** |
| **WT-01** | Verify the “SpartaGoldNode” constructor in SpartaGoldNode class. | Need maxPeers integer and PeeInfor object | High | The initial host and initial port should be in the hashtable. | Pass | None |
| **WT-02** | Verify the “buildPeers” method in the SpartaGoldNode class | Need host integer, port integer, and hops integer | High | The hash table peers should contain peers’ information. | Pass | None |
| **WT-03** | Verify the “JoinHandler” private class in the SpartaGoldNode class | Need PeerConnection object and PeerMessage object. | High | The class adds peer info and sends a reply message | Pass | None |
| **WT-04** | Verify the “ListHandler” private class in the SpartaGoldNode class | Need PeerConnection object and PeerMessage object. | High | The class the list of known peers to the peer connected | Pass | None |
| **WT-05** | Verify the “NameHandler” private class in the SpartaGoldNode class | Need PeerConnection object and PeerMessage object. | High | The class sends the its peer information as a String | Pass | None |
| **WT-06** | Verify the “SolutionFoundHandler” private class in the SpartaGoldNode class | Need PeerConnection object and PeerMessage object. | High | The class verifies the solution by hashing it with the message | Fail | None |

**Table 16.** White Box Test Cases

#### 7.3.2 Black Box Test Cases (Performance & Acceptance Test Cases)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Test Objectives** | **Pre-conditions** | **Priority** | **Expected Result** | **Actual Results** | **Defects** |
| **BT-01** | Verify the “Send” feature of the SpartaGold Wallet | Users should have enough SpartaGold in the wallet. | High | The desired amount of SpartaGold is sent. | Meets expected. See analysis for details. | None |
| **BT-02** | Verify the “Receive” feature of the SpartaGold Wallet | Users should have an address. | High | The desired amount of SpartaGold is received. | Meets expected. See analysis for details. | None |
| **BT-03** | Verify the “Balance” feature of the SpartaGold Wallet | The SpartaGold wallet is installed. | High | Balance is ready to view | Meets expected. See analysis for details. | None |
| **BT-04** | Verify the “View Transactions” feature of the SpartaGold Wallet | Users have made transactions before. | Medium | Transactions history is ready to view | Meets expected. See analysis for details. | None |
| **BT-05** | Verify the “Sort Transactions” feature of the SpartaGold Wallet | Users have made at least two transactions beforehand. | Low | Transactions are sorted by desired categories. | Does not meet expected. See analysis for details. | Sorting of table not implemented. |
| **BT-06** | Verify the “New Address” feature of the SpartaGold Wallet | Users should have at least one address. | Low | New address is successfully generated. | Does not meet expected. See analysis for details. | Feature not fully implemented. |
| **BT-07** | Verify that the SpartaGold Wallet executes from a runnable .JAR | Users should have the executable file in Windows. | High | SpartaGold Wallet runs. | Meets expected. See analysis for details. | None |
| **BT-08** | Verify the “Mine for Gold” feature | Transactions are pending, and users should have at least one address. | High | Transactions are successfully validated. | Meets expected. See analysis for details. | None |
| **BT-09** | Verify the “Exit” feature of the SpartaGold Wallet | The SpartaGold wallet software is running | High | The SpartaGold wallet is closed. | Meets expected. See analysis for details. | None |

**Table 17.** Black Box Test Cases

#### 7.3.3 Black Box Test Scenarios

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-01:** Verify the “Send” feature of the SpartaGold Wallet. |
| ***Test Location*** | SpartaGold Wallet “Send” page |
| ***Feature to be tested*** | Allow user to send Gold if users have enough Gold in their balance |
| ***Feature Pass/Fail Criteria*** | The test passes if the user can send SpartaGold successfully and the transaction information is placed in the ledger. |
| ***Means of Control*** | The test case is started by a test driver that sets up an environment with only two users in the network. |
| ***Data*** | An address is used to send the SpartaGold to the right location.  A list of transactions is needed to keep track of transactions happening in the network. |
| ***Test Procedure*** | The tester opens up the wallet and goes to the send page.  The tester then inputs the address and a specific amount of SpartaGold.  The tester clicks send and the ledger should be updated. |
| ***Special Requirements*** | The tester needs an amount of SpartaGold and a valid address to send the SpartaGold. |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-02:** Verify the “Receive” feature of the SpartaGold wallet |
| ***Test Location*** | SpartaGold Wallet “Receive” page . |
| ***Feature to be tested*** | Allow user to receive SpartaGold if the user is the addressee and the ledger should be updated with the transaction information. |
| ***Feature Pass/Fail Criteria*** | The test passes if the user can receive SpartaGold successfully and if the ledger is updated with correct transaction information. |
| ***Means of Control*** | The test case is started by a test driver call “receivedMessage”. |
| ***Data*** | The ledger that holds the list of transactions and the address. |
| ***Test Procedure*** | The test opens up two wallets.  The first one is in the “Send” page and the other in the “Receive” page.  The tester sends SpartaGold from the “Send” page.  The other wallet should the received amount. |
| ***Special Requirements*** | Two wallets need to be installed and a valid address. |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-03:** Verify the “Check Balance” feature of the SpartaGold wallet |
| ***Test Location*** | SpartaGold wallet “Check Balance” page |
| ***Feature to be tested*** | The user are able to see the balance and balance displays correctly. |
| ***Feature Pass/Fail Criteria*** | The test passes if the balance is correct and can be seen by the user. |
| ***Means of Control*** | The method CheckBalance() will be called within the test driver to verify the balance. |
| ***Data*** | The wallet will read balances by CheckBalance()method and verified against account. |
| ***Test Procedure*** | The test begins by selecting the “Check Balance” button on the Balance page.  This test will run until an balance is displayed or if the pass/fail criteria is met |
| ***Special Requirements*** | The balance must be ready to view. |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-04**: Verify the “View Transactions” feature of the SpartaGold wallet |
| ***Test Location*** | SpartaGold Wallet “Send” or “Receive” page |
| ***Feature to be tested*** | View a list of all previous transactions conducted with the wallet’s owner. |
| ***Feature Pass/Fail Criteria*** | The test passes if the page completely lists and prints all transactions done by the wallet’s owner into a window. |
| ***Means of Control*** | The method TransactionsPage() will be called within the test driver to switch to the Transactions Page. |
| ***Data*** | All previous transactions are stored in a text file which will be read in by the retrieveTransactions() method. |
| ***Test Procedure*** | The test begins by selecting the “Transactions” button at the top of the wallet.  The test will run until the page is displayed or the program is terminated. |
| ***Special Requirements*** | The wallet owner must have at least one transaction to view. |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-05**: Verify the “Sort Transactions” feature of the SpartaGold wallet |
| ***Test Location*** | SpartaGold Wallet “Transactions” page |
| ***Feature to be tested*** | Sort the list of transactions by label, date, amount, address, or sent/received. |
| ***Feature Pass/Fail Criteria*** | The test passes if the list is successfully sorted by the selected attribute. |
| ***Means of Control*** | Attributes within the list of transactions will be selected in order from left to right.  This will call their respective sorting methods (ex: sortByLabel() ). |
| ***Data*** | All previous transactions are stored in a text file which will be read in by the retrieveTransactions() method. |
| ***Test Procedure*** | The test begins by selecting the “Transactions” button.  After the list is returned, the attributes will be selected in order from left to right.  Each return will be recorded for pass or fail criteria. |
| ***Special Requirements*** | The wallet owner must have at least two transactions with different labels, dates, amounts, addreses, and send/receives before testing. |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-06**: Verify the “New Address” feature of the SpartaGold wallet |
| ***Test Location*** | SpartaGold Wallet “Receive” page |
| ***Feature to be tested*** | Generate a completely new address for other users to refer to when sending SpartaGold. |
| ***Feature Pass/Fail Criteria*** | The test passes if a completely new address is generated and continues to operate through the same wallet. |
| ***Means of Control*** | The button labeled “New Address” will call the genNewAddress() method which will print the return results in a text field. |
| ***Data*** | The wallet will read a text file containing all wallet addresses previously used, then generate a new address and compare it with those within the list for originality. |
| ***Test Procedure*** | The test begins by selecting the “New Address” button on the “Transactions” page.  This test will run until an address is displayed or if the pass/fail criteria are met. |
| ***Special Requirements*** | None |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-07**: Verify that the SpartaGold Wallet executes from a runnable .JAR |
| ***Test Location*** | Desktop/directory execution |
| ***Feature to be tested*** | This test case tests if the Wallet and executable through a runnable .JAR file. |
| ***Feature Pass/Fail Criteria*** | The test cases passes if the Wallet successfully connects to the P2P network and the Wallet window appears on screen. |
| ***Means of Control*** | Based on the environment, the user will execute the file (In Windows, the user double clicks the file). |
| ***Data*** | A window appears and a connection is established. |
| ***Test Procedure*** | The tester simply runs the JAR file through command lines or through other GUI means (i.e. double clicking). |
| ***Special Requirements*** | The wallet needs to be saved locally. |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-08**: Verify the “Mine for Gold” feature |
| ***Test Location*** | SpartaGold Wallet “Mine” page |
| ***Feature to be tested*** | This test case tests if miners are able to mine Gold from the SpartaGold system and the transaction information is added to the ledger. |
| ***Feature Pass/Fail Criteria*** | The test case passes if the miner receives the right amount of Gold after it discovers a proof during the proof-of-work process. |
| ***Means of Control*** | The validation process is started by test driver which has a puzzle and an answer. |
| ***Data*** | The ledger will have an additional entry after the validation. |
| ***Test Procedure*** | The test case is stated by opening the terminal and then entering “mine -c” to start the mining process.  The terminal starts to validate pending transactions and finds the answer.  The tester should see a message saying the amount of coins awarded. |
| ***Special Requirements*** | The tester must have a SpartaGold address. |

|  |  |
| --- | --- |
| ***Test Case Identifier*** | **BT-09:** Verify the “Exit” feature of the SpartaGold wallet |
| ***Test Location*** | The wallet window. |
| ***Feature to be tested*** | This test case tests if a person can exit the wallet after the user is finished with it. |
| ***Feature Pass/Fail Criteria*** | The test case passes if the wallet closes after it closes the running services. |
| ***Means of Control*** | The wallet is started by a test driver called TestWallet. |
| ***Data*** | A set of times and system notifications will be used. |
| ***Test Procedure*** | The test is started by opening the SpartaGold wallet.  The tester will then click on the “Exit” button.  The wallet closes and system notifications are displayed to the tester. |
| ***Special Requirements*** | The tester must have wallet synchronized the SpartaGold network. |

**Table 18.** Black Box Test Scenarios

#### 7.3.4 Validation based on documentation

## Identification of the software

Aim: To identify the software under examination

|  |  |  |  |
| --- | --- | --- | --- |
| **Identification of the software** | **Yes** | **No** | **Comments** |
| Are operating systems used? (If yes, note name and version of the OS.) | X |  | Windows 7 or Ubuntu |
| Is it possible to download new versions of the software? |  | X |  |
| Is there a unique identification of the software version? |  | X |  |
| Is the software version possible to read on the display of the device? | X |  |  |
| Is the software version possible to read through a web link? | X |  |  |
| Has the version of the software development tools (compiler, linker etc.) been identified and documented? | X |  |  |

## Completeness of the documentation

Aim: To check the completeness of the documentation describing the measuring instrument and its software.

|  |  |  |  |
| --- | --- | --- | --- |
| **Completeness of the documentation** | **Yes** | **No** | **Comments** |
| Is there an operating manual? | X |  |  |
| Are there manuals for advanced users,  e.g. installation manuals, service manuals and manuals describing communication? |  | X |  |
| Are there design descriptions for the hardware and the software? | X |  |  |
| Are there no functions other than the documented ones? | X |  |  |
| Is there a description of the menus and dialogues? | X |  |  |
| Is there an overview of the security aspects of the operating system, e.g. protection, user accounts, privileges etc. | X |  |  |

## Examination of the operating manual and technical documentation

Aim: To reveal faults in the legally relevant functions described in the operators manual and in the technical documentation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Examination of the operating manual**  **and the technical documentation** | **Yes** | **No** | **Comments** |
| Have all the legally relevant functions been described in the operating manual? | X |  |  |
| Have all the legally relevant functions been described in the technical documentation of the functionality? | X |  |  |
| Is there a detailed description of all legally relevant software functions? | X |  |  |
| Is there a detailed description of all legally relevant parameters that determine the functionality of the software? | X |  |  |

## Inspection of the specification

Aim: To examine the design objectives for the legally relevant functions, and to make sure that all relevant requirements have been considered.

|  |  |  |  |
| --- | --- | --- | --- |
| **Inspection of the Specification** | **Yes** | **No** | **Comments** |
| Is the specification complete that all its parts are present and fully developed? | X |  |  |
| Is the specification consistent that its provisions do not conflict with each other? | X |  |  |
| Is the specification feasible that the life-cycle benefits of the system specified will exceed the life-cycle costs? | X |  |  |
| Is the specification testable that one can identify an economically feasible technique for determining whether or not the developed software will satisfy the specification? | X |  |  |

#### 7.3.5 Test Analysis Report

## Purpose

The purpose of this Test Analysis Report was to provide very detailed examinations of SpartaGold processes.  This report detailed the types of testing conducted by the development and testing teams, and inspected reasons behind defects as well as improvement advice.

## Testing Scope

A variety of tests were conducted on the SpartaGold Wallet and network to provide detailed results.  These results have helped the SpartaGold project coordinators detect and inspect defects, limitations, and constraints within the functionality and performance of the application.  Validation testing methods were incorporated in the testing process to produce a proper product as intended by the consumer (i.e. the development team and their advisor).  White box testing, which encompasses the project’s unit and integration testing, was conducted by the development team to ensure individual components operated as expected, and each vital method within the code works as intended.  Black box testing, which encompasses the project’s performance and acceptance testing, was conducted by the testing team to provide feedback on end user experiences with functionality.  Results are outlined under the Test Analysis section.  Beta testing and security testing have not been conducted, as the final build of the project must be completed before any tests can take place.

## Test Analysis

### *White Box Testing (Unit and Integration Testing)*

**WT-01:** Verify the “SpartaGoldNode” constructor in SpartaGoldNode class

**System Function:**

The SpartaGoldNode constructor creates an object that initializes instance variable and adds a known peer to its known peer list.  The constructors takes as input a PeerInfo object and an integer for the number of peers.

**Functional Capability:**

WT-01 completed successfully.  The constructor was able to create an object by allocating the right amount of memory and locations.  It initialized the required instance variables by using the parameters and other local variables.  In the test, there was assertion equal statements that used the getter methods of the object to check for equality.

**WT-02:** Verify the “buildPeers” method in the SpartaGoldNode class

**System Function:**

The buildPeers method connects with different computer in the network and saves their IP address and port number in a hash table.  It requests a list of peers from a node and it does a depth first search connecting with the peers in that list.

**Functional Capability:**

WT-02 completed successfully.  The buildPeers was able to successfully able to add IP addresses and port numbers to the node’s hash table.  In the test, there was assertion equal statements used that accessed the peers hash table for the key values.

**WT-03:** Verify the “JoinHandler” private class in the SpartaGoldNode class.

**System Function:**

The JoingHandler private class handles messages of the type “INSERTPEER” by archiving the requestors’s IP address and port number.

**Functional Capability:**

WT-03 completed successfully.  The JoingHandler was able to handle a message request of the type “INSERTPEER”.  In the test, the node received a message that had a node’s IP address and port number.  An assertion true statement was used by checking that the peers table contained specific information.

**WT-04:** Verify the “ListHandler” private class in the SpartaGoldNode class.

**System Function:**

The ListHandler private class handles any request of the type “LISTPEER” by sending the node’s list of known peers to the requestor.

**Functional Capability:**

WT-04 completed successfully.  The ListHandler was able to handle a message of the type “LISTPEER”.  In the test, an assertion equals and assertion null statements were used to check for when it receive a list and when it did not.  Both test passed and more will be done.

**WT-05:** Verify the “NameHandler” private class in the SpartaGoldNode class

**System Function:**

The NameHandler private class handles any request of the type “PEERNAME” by sending the node’s IP address and port number to the requestor.

**Functional Capability:**

WT-05 completed successfully. The NameHandler was able to handle a message of the type “PEERNAME” by sending accesing the node’s information and sending it to the requestor.  In the test, the getter for the information and assertion equal statement were used to check for equality.

**WT-06:** Verify the “SolutionFoundHandler” private class in the SpartaGoldNode class

**System Function:**

The SolutionFoundHandler private class handles any requests of the type “FOUNDSOLUTION” by hashing the receive message and solution.  If the solution is correct, the node updates its chain block.

**Functional Capability:**

WT-06 failed when executed.  The SolutionFoundHandler needed to validate a received solution by hashing a solution and message.  The hashing function seems to be the culprit because it is not returning a binary stream of numbers.  This shall be fixed by the developer.

### *Black Box Testing (Performance and Acceptance Testing)*

**BT-01:** Verify the “Send” feature of the SpartaGold Wallet

**System Function:**

The “Send” tab of the SpartaGold Wallet takes the address of the receiver and an amount of SpartaGold (SG).  These messages, along with a public key file, a signature file, a block chain file, and a user map file, are broadcasted using an integrated peer-to-peer network library to every user connected to the SpartaGold network.

**Functional Capability:**

BT-01 completed successfully, first on separate unit tests (GUI and file conversion, and simple message sharing through a peer-to-peer network), and then as one process.  The specified files were sent from one user to another a total of 10 times.  A connection was established between two users after both parties specified a port and had either disabled any existing firewalls or port forwarded the specified port.

**Performance Capability:**

The “Send” feature successfully reached its intended goal.  However, concerns lie among “loose” files being sent in batches.  File packaging is advised.  Broadcast times gradually grew as both the block chain file and the user map file expanded.

**BT-02:** Verify the “Receive” feature of the SpartaGold Wallet

**System Function:**

After a user has generated an address and submitted it to another user through external means (text message, written, etc.), the receiver simply awaits validation of the transaction as funds are automatically added to the receiver and deducted from the sender.  This transaction is conducted and validated over the SpartaGold P2P network.

**Functional Capability:**

BT-02 completed successfully.  A P2P connection was established in order to process the transaction.  Initially, prototypes successfully received text messages.  This evolved into sending a file over the P2P network, which then changed into sending multiple files at once.  Like BT-01, a port was specified among both parties and port forwarded to ensure no interruptions.

**Performance Capability:**

Receiving a full broadcast took less than a second long on all 10 attempts, but this could grow once the block chain becomes much larger.  In addition, the validation process created an unpredictable amount of time, due to the nature of the proof-of-work concept (in short, this unpredictable amount of time was expected and intended).  Validation can take significantly less time when performed on a high-end machine or when multiple users are validating the same transaction.

**BT-03:** Verify the “Balance” feature of the SpartaGold Wallet

**System Function:**

The “Balance” feature is automated and displayed on the “Send” tab of the SpartaGold Wallet.  After conducting a transaction with another user, and depending on whether the user is sending, receiving, or validating, the “Balance” feature will display the user’s current balance saved on the user map file.

**Functional Capability:**

BT-03 completed successfully.  The wallet owner’s/user’s balance displayed with the correct balance after processing 3 transactions in each situation: sending Gold, receiving Gold, and receiving a validation reward.  A port was established between the sender and receiver in order to send and receive, but this test also passed in its unit test during development.

**Performance Capability:**

As the user map file grows, balance fetching increases in time.  However, all 9 tests returned under 1/10th of a second each.  Users and their balances are mapped using a key-value pair, so data retrieval will not require more than O(n) time complexity in its worst case scenario and have an average of O(1).

**BT-04:** Verify the “View Transactions” feature of the SpartaGold Wallet

**System Function:**

The “Transactions” tab displays a table of all previous transactions the current wallet has processed.  The list has multiple columns for multiple sections of information per transaction.  This table is purely local, so a connection to the SpartaGold P2P network is not necessary.  This table functions as a receipt and logs all transactions done by the user.

**Functional Capability:**

BT-04 completed successfully.  The table does not grow very fast and the information stored within is not complex (saved in plaintext).  In addition, the local file storing these transactions does not need to be shared with those on the SpartaGold network, and therefore does not require a connection to the P2P network.  Newly processed transactions were added to the list, as expected.

**Performance Capability:**

Data retrieval is very fast, under 1/10th of a second consistently for 5 trials.  Local file storage and maintenance provided fast and efficient results.

**BT-05:** Verify the “Sort Transactions” feature of the SpartaGold Wallet

**System Function:**

While under the “Transactions” tab, each column of the table provides the option to sort the list based on the column’s attribute.  The attributes are listed in order as follows: Date, Address, and Amount.  Clicking each of these headers will sort the subsequent list to that attribute.

**Functional Capability:**

BT-05 did not complete successfully.  Clicking each header did not sort the list, and the list just continued to populate based on the time it was added to the receipt.  This is an implementation defect, as the developing team did not focus on completing this task due to its low priority rating.

**Performance Capability:**

As the test failed, performance could not be measured.  These tests were run under the assumption that users have at least two previous transactions processed.

**BT-06:** Verify the “New Address” feature of the SpartaGold Wallet

**System Function:**

A “New Address” button located on the “Transactions” tab allows users to generate a new public key and wallet address.  This process is automated and will return the generated address in a text field located next to the “New Address” button.

**Functional Capability:**

BT-06 completed, but with concerns regarding usage with the user map file.  Generating a new address works as intended, but the Wallet saves this new address as a separate user.  This is unacceptable, as users intend to generate new addresses linked to the same Wallet.

**Performance Capability:**

Generating new addresses is fast, as the process is simple.  No difficulties were found in this process.

**BT-07:** Verify that the SpartaGold Wallet executes from a runnable .JAR

**System Function:**

Upon executing the .JAR file (within a Windows environment), the SpartaGold Wallet boots and automatically connects to the network if a network connection is established.  The Wallet window appears and shows the initial “Send” tab.

**Functional Capability:**

BT-07 completed successfully.  The execution file boots the window with the initial “Send” tab.  The wallet automatically produces the user’s first ID.  The Wallet automatically connects to the P2P network and waits for transaction commands from the user.

**Performance Capability:**

Depending on the internet connection of a user, the Wallet’s startup time can vary.  On all 10 tests conducted on the Wallet, the startup time took 4 seconds on average.  Connection to the network is the primary reason behind this amount of time, as the Wallet must establish a connection with other peers.

**BT-08:** Verify the “Mine for Gold” feature

**System Function:**

The “Mine” tab contains a single button labeled “Mine for Gold”.  This button begins the validation process by listening for any transaction broadcasts awaiting validation.  Once a transaction is found, the Wallet automatically processes the proof-of-work algorithm.  If a proof is found, the block chain process finishes and broadcasts to everyone among the network.  The winning validator is then awarded a validation reward.

**Functional Capability:**

BT-08 completed, but with limited success.  Transaction verification and validation are autonomous, using the broadcasted public key, signature, and transaction information.  The proof-of-work does not stop processing when another user has discovered a proof, causing work for absolutely no value.  A winner’s broadcast listener is advised.

**Performance Capability:**

Measurement of performance is difficult with BT-08.  The concept of proof-of-work relies on the amount of computations a system can perform within an uncertain time frame.  This process was tested with only one validator, so the proof-of-work’s complexity was diminished in order to perform relatively fast proofs.  As the amount of validators increases, the proof-of-work complexity must increase.

**BT-09:** Verify the “Exit” feature of the SpartaGold Wallet

**System Function:**

The SpartaGold Wallet window contains an “X” button at the top right, which closes the window and disconnects the user from the SpartaGold network.  Validation tasks are interrupted, but transactions do not require a connection to the P2P network after a broadcast has been sent to at least one peer.

**Functional Capability:**

BT-09 completed successfully.  The Wallet shuts down as expected, and all connections to the P2P network end.  Mining stops, as the process runs within the Wallet’s functionality.

**Performance Capability:**

Ending all processes is not very computationally extensive, and all 10 tests of shutting down the Wallet worked as expected.

## Functional Requirements Checklist

|  |  |  |  |
| --- | --- | --- | --- |
| **F-01** | The user shall be able to see their pending and past transactions. | Medium | Pass |
| **F-02** | The user shall be able to see their available balance. | High | Pass |
| **F-03** | The user shall be able to receive Gold from another user. | High | Pass |
| **F-04** | The user shall have the option to validate transactions. | Medium | Pass |
| **F-05** | The user shall have a ledger of all transaction records that have occurred in the SpartaGold network. | High | Pass |
| **F-06** | The user should be able to sort their transaction record by date, address, amount, sent, received, or pending. | Low | Fail |
| **F-07** | The user shall be able to send Gold to another user by inputting the receiver’s address and the amount in a text field. | High | Pass |
| **F-08** | The user shall be charged a transaction fee in SpartaGold which will be awarded to the validator. | High | Pass |
| **F-09** | The user shall be able to generate new random addresses for transactions. | Low | Pass\* |
| **F-10** | The user shall be able to sign a message with their private key. | High | Pass |
| **F-11** | The user shall be able to verify the signature of a received message. | High | Pass |
| **F-12** | The user should be able to generate a QR code of an address. | Low | Fail |
| **F-13** | The system shall provide anonymity to users’ personal information. | High | Pass |
| **F-14** | The system shall implement a cryptographic protocol to ensure secure one-on-one conversations. | High | Pass |
| **F-15** | The system shall implement a proof-of-work function to deter attacks and phony validations. | High | Pass |
| **F-16** | The system’s wallet shall request password verification for access to its features. | Low | Fail |
| **F-17** | The system’s wallet shall provide a new address when requested. | Medium | Pass |
| **F-18** | The system should allow users to transport their wallet addresses or private keys through print or mobile devices. | Low | Fail |
| **F-19** | The system shall allow users to mine through a set of console commands or buttons. | Medium | Pass\* |
| **F-20** | The system shall broadcast pending transactions to others connected to the network. | High | Pass |
| **F-21** | The system’s wallet shall still operate with limited functionality when offline by disabling transaction sending, transaction receiving, and ledger synchronization. | Medium | Pass |
| **F-22** | The system shall provide validators with a specified amount of SpartaGold when a block chain is discovered. | Medium | Pass |
| **F-23** | The system shall provide validators with a specified amount of Gold allocated from the transaction as a transaction fee when a block chain is discovered. | Medium | Pass |

\*Passed with concerns or complications.

**Table 19.** Functional Requirements Checklist

## Conclusion

### *System Deficiencies*

Within the white box testing experiments, errors were discovered within the code which caused some tests to fail in WT-05.  The priority of the failed test is high; therefore, a fix to this bug must be taken into consideration as soon as possible.  Black box testing revealed a few deficiencies in the project.  The “Sort transactions” function used in BT-05 was not implemented, and did not perform as expected.  It is important to note that this functionality was prioritized as “low”, so implementation was scrapped due to time constraints.  The “New Address” function used in BT-06 provided the functionality it was asked to perform, yet the combined use of this new address with the balance of the user was not planned, and therefore did not perform to its full functionality.  As well, the “Mine for Gold” function used in BT-08 worked as expected, but did not stop its own process when needed, causing a large amount of computational work for no discernible reason.  Four functional requirements did not pass due to time constraints.  Three of these requirements (F-06, F-12, and F-18) have a “low” priority, so their implementation will not have much consideration.

### *System Refinements*

Integration testing is still needed to solve WT-05 and BT-06.  WT-05 needs specific attention among parts within the P2P code, while BT-06 needs attention among the ledger and cryptography code.  BT-08 requires modular work in order to pass.  The development team will continue to fix these bugs, then conduct tests for WT-05.  The testing team will wait for confirmation from the development team before conducting tests for BT-06 and BT-08.  F-06, F-12, F-16, and F-18 should not be considered in implementation as they do not hold a “high” priority.

### *Recommendations and Estimates*

Integration work is very necessary, so high priority items must be completed before lower priority items.  If time does not allow for all bugs to be fixed, lower priority items such as transaction sorting must be removed.  Beta testing and Security testing will be conducted after black box and white box testing has passed with a 100% pass rate.

# Chapter 8

## Conclusion and Future Works

SpartaGold is an improvement on an already-established service within the San Jose State University community.  Through rigorous implementation sessions and tests, SpartaGold can now flow through the SpartaGold Wallet from users directly to other users.  SpartaGold exchanges will keep transaction fees low, removing the necessity of a centralized authority.  These low transaction fees are given to other users who put the effort into validating other users’ transactions, creating a strong motivational force to drive the currency into a safe low-inflationary state.  Testing phases of the SpartaGold Wallet have showed the development team that the currency will continuously sustain itself, on the basis that users continue to mine and spend.

 Future work into SpartaGold will revolve around a more complex proof-of-work system.  In order for this change to come into reality, the use of SpartaGold must first grow.  The implementation of the concept of “proof-of-stake” feature by the SpartaGold development team is a very possible option, as well as the integration of mobile devices.  The implementation of SpartaGold within on-campus vendors is also a very likely possibility in the future, as transactions are most common in these establishments.  These future plans require a stable user base, which will come to fruition once SpartaGold becomes the go-to currency of students and faculty of SJSU.

# References

[1] Couts, A. (2013, December 14). Bitcoin alternatives explained: Litecoin, Namecoin,  Ripple, and more | Digital Trends. Retrieved from<http://www.digitaltrends.com/web/missed-bitcoin-boat-6-cryptocurrencies-might-make-rich/#!AYYw9>

[2] CyberCash (n.d.). Retrived from<http://www.ksofttech.com/help/cybercash/>

CyberCash -- Useful information for Internet merchant. Retrieved from<http://www.vivtek.com/cybercash/info.html>

[3] Jackson, B. (2014, January 21). Beyond Bitcoin: Top 5 cryptocurrencies by market cap | IT Business. Retrieved from [http://www.itbusiness.ca/slideshows/beyond-bitcoin-top-5-cryptocurrencies-by-](%20http://www.itbusiness.ca/slideshows/beyond-bitcoin-top-5-cryptocurrencies-by-)m[arket-cap](http://www.itbusiness.ca/slideshows/beyond-bitcoin-top-5-cryptocurrencies-by-market-cap)

[4] King, S., & Nadal, S. (2012, August 19). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. Retrieved from<http://primecoin.org/static/ppcoin-paper.pdf>

[5] King, S. (2013, July 07). Primecoin: Cryptocurrency with prime number proof-of-work.  Retrieved from<http://primecoin.org/static/primecoin-paper.pdf>

[6]Nakamoto, S. (2009, January 03). Bitcoin: a peer-to-peer electronic cash system. Retrieved from<https://bitcoin.org/bitcoin.pdf>

[7] Steadman, I. (2013, May 11). Wary of bitcoin? a guide to some other cryptocurrencies. Retrieved from<http://arstechnica.com/business/2013/05/wary-of-bitcoin-a-guide-to-some-other-cryptocurrencies/2/>

[8] NOAA National Weather Service (2007, June 30). General Software Development Standards and Guidelines. Retrieved from http://www.nws.noaa.gov/oh/hrl/developers\_docs/General\_Software\_Standards.pdf

[9] Stevenson, J. (2013). All you need to know about Litecoin (after Bitcoin). Retrieved from <http://books.google.com/books?id=9ydzAgAAQBAJ>

[10] Percival, C. (2012). Stronger key derivation via sequential memory-hard functions. Retrieved from <http://www.tarsnap.com/scrypt/scrypt.pdf>

[11] Retrieved from arxiv.org/abs/1311.0243

[12] Retrieved from <http://dogecoin.com/>

[13] Retrieved from https://bitcoin.org/en/vocabulary