# -What is CS 2450.pptx Notes-

Each sprint has:

* Rubric
* Project Document
* Developed(Checklist)

Programming:

* Python
* Thonny
* HCGUI built with Tkinter

Office hours:

* Syllabus
* Emailed APPT. for team or individuals

Misc.

* Self learning
* Habit of discipline
* Information will be there but don’t expect help
* TONS OF INFORMATION IS PROVDIED BUT YOU HAVE TO READ IT

# -Course Guidelines-

## Overview:

### Course is about

* + Management
  + Programming
  + Development techniques

### In this course you will

* + Develop a design for solving simulated real-world project
  + Use CS1410 project as a beginning prototype
  + Develop a prototype to satisfy the course project requirements
  + Once Prototype is approved
    - Make required revisions
    - Add more functionality
  + Create a solution with documentation and project management that demonstrates to the stakeholder
    - Exactly what your solution will provide
    - How your solution will work when you obtain approval for the final baseline

### Project

* + Divided into 6 sprints

### Professor

* + Will act as professor in class
    - Ask technical questions about the project in class
  + Will act as customer in team meetings
    - Ask questions about what your software should do in team meetings
  + Instructions on software product
    - Not technically
    - Not complete
    - Not well founded
    - Requires research on the team’s part(Requirements Engineering)

## Preparing Sprint Documentation

### Before simply giving an opinion, you must include references from

* + Book
  + Other sources that support your opinion

### Clearly and succinctly(brief and clear) state your

* + Reasons
  + Opinions
  + Research
  + Etc.

### All answers in exercises should have

* + Grammar
  + Spelling
  + Clear formatting
  + Name
  + Date
  + Assignment
  + Course information in header

## Team Project:

### The Team Number must be included in all team project documentation

### Random membership assignment will be developed by the instructor

### Teams must have a team leader

### Refer to **Term Project Docs** for description of product

### Refer to **Term Project example** for a partially successful project solution

### Refer to **Team Procedure document** for additional information

## Detailed File Management:

### Team assignments must have the team folder submitted individually by all team members

### Team document submission

* + <Team Number>\_<assignment name>\_CS2450-<section>\_<session period>.pdf

### Individual team member file

* + <Team Number>\_CS2450-<section>\_<Team member Initials>\_<assignment name>\_pdf

### Any exercise/assignment requiring written submission

* + <Submitter’s name>\_CS2450\_<section>\_<assignment name>.pdf

### Take home exams if essay questions are required

* + <Submitter’s name>\_CS2450\_<section>\_<exam name>.pdf

### Functioning code

* + Above team submission format in a Zip file

### Required Software

* + Latest version of Thonny
  + Latest version of Python/Tkinter

### Must support pdf output

* + Latest version of Lucidchart
  + Latest version of Backlog
  + Latest version of Work Breakdown Structure software
  + Word processing document

# -Project Management 101: The complete guide to Agile, Kanban, Scrum and beyond-

## Gantt chart:

### Gantt chart

* + Lists a project schedule based on start and finish dates
  + List how long a task takes
  + List if any other tasks have to be completed before you the task can start
  + Calculate the critical path of the activities that must be completed by certain dates
  + Estimate how long the total project will take

## Common Project Management Terms

### Agile

* + An iterative form of project management where tasks are completed through specific phases

### Critical path

* + The list of critical tasks that must be completed before a project is finished
  + Together, they show the total estimated project time

### Event Chain Diagram

* + A bar graph of events in a project and the order they’ll be completed based on resource availability

### Float

* + The amount of time a task can be delayed without causing a delay to subsequent tasks or the entire project

### Gantt chart

* + A bar graph and calendar fusion that shows the time each task in a project will require

### Milestone

* + The time when important tasks in a project are completed

### Project Manager

* + The team member whose top responsibility is to place, carry out and close a project

### Resources

* + Elements required to complete a project

### Scope

* + The definition of what the project will cover

### Scope creep

* + When the scope of a project increases

### Sprint/iteration

* + Period of time in which a certain part of a project is created and shipped

### Traditional Project Management

* + Basic project management where tasks are completed one after another

## Traditional Project Management:

### Most obvious way to break up your projects into a workflow

### Often referred to as waterfall project management

### Handles tasks in a linear order

### Stresses on-time delivery

### Stringent budget

### Useful when

* + Emphasize planning
  + Tasks need to be completed one after another

### Six specific stages

* + Initiation phase
    - Determine product requirements
    - Requirement’s determination
    - Everyone participates in a brain dump
  + Planning and design phase
    - Team makes sure the proposed design meets the product requirements
  + Execution and testing phase
    - Construction
    - Integration
    - Team builds product
  + Monitoring and completion phase
    - Work that never ends
    - Dedicated to keeping customers and users happy
    - Improving product
    - Maintaining and supporting product

## Agile:

### Split project into smaller projects and ship each one as steps towards reaching the full goal

### Plan broad ideas of the project and divide it up, then

* + Plan
  + Design
  + Build
  + Test

### Strengths

* + Flexibility
  + Responding to change over following a plan

### Weaknesses

* + Difficult to focus and push your projects to completion
  + Less set in stone
  + No process to make sure the project is continuing smoothly

## Scrum:

### One of the most structured frameworks of the Agile methods.

### Teams work as a unit to reach a common goal

### Breaks projects up into tasks that are completable on their own

### Assigns each sprint (2-4 weeks) to ship that phase of the project

### Focus on time

### Requires a reassessment and project changes at the end of each sprint

### Five meetings

* + Backlog Refinement Meeting
    - Held day one of each sprint
    - Look over tasks left in the project and decide what to focus on
    - PO makes the call on how to prioritize tasks
  + Sprint Planning Meeting
    - This meeting helps the team understand what they’ll be building and why
    - Share user stories, describing features from the customer’s point of view
    - Divide tasks for each team to work on during the sprint
  + Daily Scrum Meetings
    - Simple daily meetings that should only last about 15 minutes
    - A way for team members to update each other on progress
  + Sprint Review
    - Team members will present what they’ve completed to all stakeholders
  + Sprint Retrospective
    - Collaborative feedback
    - Looking at success and hold ups
    - Everyone decides what is working and what isn’t working

## Lean:

### Adds workflow processes to Agile

### Break up your project into smaller pieces of work

### Define workflow for each task

## Kanban

### Do some work towards a project, then ship that item on down the line to the next station where something else is done

### Leave tasks at various stages until they’re needed

### Define stages of your workflow

### Setup a way to move each task from one stage to the other

### Four pillars of the Kanban philosophy

* + Cards
    - Each task has a card that includes all relevant info about it
  + Cap on work in progress
    - Limit how many cards are in play at once
  + Continuous Flow
    - Move down the list of backlogs in order of importance
    - Make sure something is always being worked on
  + Constant improvement
    - Analyze the flow to determine how efficiently you’re working
    - Always strive to improve efficiency

## Six Sigma

### More structured version of Lean

### Make customers happy with a quality product

### Heavily reliant on data analysis

### Ship parts of your project while addressing project pitfalls that arise

### Five steps

* + Define
    - Everyone determines the scope of the project
    - Gets information from all sides
    - Determines what the business goals are
  + Measure
    - Establishes the nature in which the team will calculate progress and overall goals
  + Explore
    - Project manages figures out the way in which the team can meet and exceed product requirements
  + Develop
    - Detailed strategic plan put in place
    - Most of the project’s momentum occurs here
  + Control
    - Documented review full of lessons learned and applied

## Prince2

### Approaches a project as one big sprint

### Stresses quality of delivery

### Three interests at play

* + Business interest
    - Is it going to make money?
  + User interest
    - Will customers find this valuable
  + Supplier interest
    - Do we have what we need to make this happen?

### Each team member has specific roles

### Seven stages

* + Startup
    - Leadership chooses a project manager and clearly relays what they expect the product to be
    - Project manager
      * Focuses on fine details
      * Reports to project board
  + Initiation
    - Project manager writes the initiation document
    - Project is divided into phases
    - Phase must be completed before moving on to the next one
  + Direction
    - Sets overall management structure for the project
    - Outlines how each stage should progress
    - Outlines what should happen if something changes along the way
  + Control
    - Roadmap for each phase is determined by the review of the previous one
    - General plan can be manipulated
  + Boundary management
    - Looks at product delivery
      * What is going out?
      * How it’s going out?
      * Is the product that is going out exactly what the business wanted
  + Delivery
    - Project manager is in charge of making sure that everyone is doing a job that aligns with the project’s goals
  + Closing
    - In-depth analysis report of how the project faired

# -UML 2 Sequence Diagrams: An Agile Introduction-

## UML sequence Diagrams:

### Model the flow of logic within your system in a visual manner

### Enable both documentation and validation of logic

### Used for analysis and design purposes

### Used to model

* + Usage scenarios
    - Description of a potential way your system is used
    - May be part of a use case
    - May be one entire pass through a use case
  + The logic of methods
    - Explore the logic of complex operation, function, or procedure
    - Visual object code
  + The logic of services
    - A high-level method
    - Often one that can be invoked by a wide variety of clients

## How to draw sequence diagrams

### Identify the scope of what you are trying to mode

### Tackle small usage scenarios at the system level or a single method/service at the detailed object level

### Lay out classifiers across the top as needed

### Automatically add the object lifelines

### Heart of diagram is in the messages

* + Add to diagram one at a time as you work through the logic

### Give messages intelligent names which make it clear what is being returned

### As you diagram you will identify new responsibilities for

* + Classes
  + Objects
  + New classes

### Update class model appropriately

### Each message sent to a class invokes a static method/operation on that class each message sent to an object invokes an operation on that object

### Draw messages going left-to-right

### Draw return values from right-to-left

### Justify label on messages and return values

### Label sequence diagrams from left-to-right

### Indicate the

* + Actors
  + Controller class(es)
  + User interface class(es)
  + Business class(es)

### During design

* + Add system classes
  + Add persistence classes

## Reference

### <http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>

# -CS 2450 software links-

[www.backlog.com (Links to an external site.)](http://www.backlog.com/) - make backlog records  
[www.workbreakdownstructure.com (Links to an external site.)](http://www.workbreakdownstructure.com/) - diagram work breakdown structures  
[www.lucidchart.com (Links to an external site.)](http://www.lucidchart.com/) - diagram UML and many others.  
Office 365 no longer includes visio for free  
[www.smartsheet.com (Links to an external site.)](http://www.smartsheet.com/) - supports team and project management  
[www.smartdraw.com (Links to an external site.)](http://www.smartdraw.com/) - build charts  
how to get lucid art for free: [current link on Banner (Links to an external site.)](https://uvu.edu/software/).  
[www.tutorialspoint.com (Links to an external site.)](http://www.tutorialspoint.com/) - PyQt tutorial book.

# -The Delphi Method-

## Delphi process:

### Step 1

* + Every team member lists their own suggested functional requirements/non-functional requirements
  + Discussion is not allowed between team members
  + This is the first iteration, and needs to be in the sprint submission

### Step 2

* + Every team member sends their functional requirements/non-functional requirements to the team leader anonymously

### Step 3

* + Team leader assembles the four documents into two, eliminating duplications
  + Send each document to a sub-group of two students
  + Each sub-group discusses, revies and modifies their document

### Step 4

* + Once the sub-group has agreed upon their functional requirements/non-functional requirements, submit it to the leader
  + This is the second iteration
  + Don’t write names on documents
  + Include these documents in your submission as well

### Step 5

* + Using a Group meeting, assemble the documents not on functional requirements / non-functional requirements chart without duplication
  + Discuss, modify, and revise to reach the final version
  + Include the final version in your submission as possible additions to the baseline

## Functional requirement:

### A description of the service that the software must include

### It describes a software system or its component

### A function is nothing but its

* + Inputs to the software system
  + Behavior
  + Outputs

### A function can be a

* + Calculation
  + Data manipulation
  + Business process
  + User interaction
  + Any specific functionality which defines what function a system is likely to preform

## Functional Requirements of a system should include the following things:

### Details of operations conducted in every screen

### Data handling logic should be entered into the system

### It should have descriptions of system reports or other outputs

### Complete information about the workflows performed by the system

### It should clearly define who will be allowed to create/modify/delete the data in the system

### How the system will fulfill applicable regulatory and compliance needs should be captured in the functional document

## Non-functional requirements:

### If you think of functional requirements as those that define a system is supposed to do, non-functional requirements define constraints which affect how the system should do it

### A system can work if the non-functional requirements are not met

### Keep functional requirements in line

## Types of non-functional requirements:

### Security

* + Does your product store or transmit sensitive information?
  + Does your IT department require adherence to specific standards?

### Capacity

* + What are your system’s storage requirements, today and in the future?
  + How will your system scale up for increasing volume demands

### Compatibility

* + What are the minimum hardware requirements?
  + What operating systems and their version must be supported?

### Reliability and Availability

* + What is the critical failure time under normal usage?
  + Does a user need access to this all hours of every day?

### Maintainability + Manageability

* + How much time does it take to fix components
  + How easily can an administrator manage the system

### Scalability

* + The Black Friday Test
  + What are the highest workloads under which the system will still perform as expected

### Useability

* + How easy is it to use the product?
  + What defines the experience of using the product?

## Best practices for writing non-functional requirements:

### Be consistent in terminology and format

### Quantify requirements

## Non-Functional vs. Functional Requirements:

| **Parameters** | **Functional Requirement** | **Non-Functional Requirement** |
| --- | --- | --- |
| What it is | Verb | Attributes |
| Requirement | It is mandatory | It is non-mandatory |
| Capturing type | It is captured in use case. | It is captured as a quality attribute. |
| End result | Product feature | Product properties |
| Capturing | Easy to capture | Hard to capture |
| Objective | Helps you verify the functionality of the software. | Helps you to verify the performance of the software. |
| Area of focus | Focus on user requirement | Concentrates on the user’s expectation. |
| Documentation | Describe what the product does | Describes how the product works |
| Type of Testing | Functional Testing like System, Integration, End to End, API testing, etc. | Non-Functional Testing like Performance, Stress, Usability, Security testing, etc. |
| Test Execution | Test Execution is done before non-functional testing. | After the functional testing |
| Product Info | Product Features | Product Properties |

# -Submissions.pdf-

## Submissions for grades:

### 1 document required for early sprints

### 2 documents required for later sprints(when coding starts)

### Sprint document contains

* + All requested content that is not complete code
  + Code snippets(sometimes)
  + Code in Plain English(sometimes)
  + Pseudo code(sometimes)
  + All work completed by all teammates

### Sprint document(no-code)

* + Submitted in one pdf file
  + Is to explain your project experiences
  + Title shall be
    - Team Number
    - Sprint number
    - Course designation.pdf

### Working code

* + Shall be submitted in a zip file
  + Shall be named
    - Team Number
    - Sprint number
    - Course designation.zip

### Best solution to submitting assignments is using a team google Drive

* + Share link to the root project folder to “anyone with the link”
  + Allow them to edit
  + Use this with team members and shareholder
  + Within the root project folder place the one/two submission files in a submission file folder named after the team project sprint
    - T#-SPR#\_CS2450-XXX\_F2

### Quality Control effort would also include a complete and thorough review of

* + Your submission when reviewed using the course requirements
  + The project requirements
  + Rubric content
  + Can develop QA submission checklist

### All team members need to submit the same team Sprint link at the end of the current Sprint

### Connect google drive link to the root folder

* + Rename the link text T#\_SPR#

### Create another folder in the root project folder that is called ToReviewFiles

* + Notify sharpcs2450@gmail.com that there are unreviewed files that need to be looked over

# -12 Principles Behind the Agile Manifesto-

## 12 principles:

### Principle 1

* + Our highest priority is to satisfy the customer through early an continuous delivery of valuable software

### Principle 2

* + Welcome changing requirements, even late in development
  + Agile processes harness change for the customer’s competitive advantage

### Principle 3

* + Deliver working software frequently from a couple of weeks to a couple of months
  + Preference to the shorter timescale

### Principle 4

* + Businesspeople and developers must work together daily throughout the project

### Principle 5

* + Build projects around motivated individuals
  + Give them the environment and support the need
  + Trust them to get the job done

### Principle 6

* + The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

### Principle 7

* + Working software is the primary measure of progress

### Principle 8

* + Agile processes promote sustainable development
  + The sponsors, developers, and users should be able to maintain a constant pace indefinitely

### Principle 9

* + Continuous attention to technical excellence and good design enhances agility

### Principle 10

* + Simplicity-the art of maximizing the amount of work not done-is essential

### Principle 11

* + The best architectures, requirements, and designs emerge from self-organizing teams

### Principle 12

* + At regular intervals, the team reflects on how to become more effective, then tunes ands adjusts its behavior accordingly

# -Get the Best from Scrum Meetings-

## What are scrum meetings:

### Short and crisp meetings

### Members usually stand as the discomfort of standing makes sure that the meeting does not stretch too long

### Meetings are held daily or short intervals

### Topics

* + What was done yesterday?
  + What is to be done today?
  + Are there any impediments?

## Tips for successful stand-up meeting:

### Stick to the rule: Time is money

* + Meetings should start at a fixed time every morning whether all members are there or not
  + Could offer incentives for those who arrive on time
  + Could impose penalties on those who are late
  + Keep meeting no longer than 15 minutes

### Predefine the standard of the meeting

* + Only scrum leader and the team members should be allowed to report in the daily scrum meeting
  + Only one person should be allowed to speak without interruption at a given time

### Do not lose focus

* + Prepare a list of topics and priorities which need to be discussed during a stand-up meeting
  + Only those items which are currently in process or those who due date is approaching should be dealt with
  + Topics should be of universal interest

### Keep the interest level high by raising the fun quotient

* + Start meeting with a joke, motivating quote, etc.
  + Lottery system to decide who speaks next

### No laptops or phones

* + Ensure the stand-up meetings are tech-free

### Stand up

* + Stand up during the meeting

### Do not attend the meeting if

* + If your time is worth more than the meeting

### Use Parking Lot for unresolved issues

* + Have a separate meeting for specific issues with specific people

### Be prepared ahead of time

* + Team members should know exactly what they would speak at the meeting
  + Speech should be strictly limited to what they did the day before, what they plan to do today, and what are the difficulties they are facing
  + Scrum leader should make it clear to the team members what he expects to hear from them at the meeting
  + Anything off the topic should be interrupted

### Use scrum meetings to infuse team spirit

* + Make sure teammates are communicating

# -The ultimate guide to Scrum boards-

## What is a Scrum board:

### A Scrum board is a project management tool used by teams to make items in their Sprint Backlog visible

### The board is constantly being updated by team members

### Show all items that need to be completed for the current Sprint

### Tool to help you manage and run your projects

### Provides you with a way of seeing what’s going on and keeping on top of it

### Scrum board is specifically designed to help you tackle Scrum projects

### A Scrum board can

* + Visually organize a Sprint backlog and the individual use stories
  + Assign ownership of user stories and workflows to team members
  + Give everyone context so they can identify potential bottlenecks in the process

## How to use a Scrum board:

### Adopt a template

* + https://monday.com/templates

### Create your task board as a team

* + Populate your template with
    - Tasks
    - User stories
    - Features
    - Requirements

### Flesh out each candidate product backlog item

### Create logical work items

### Estimate how long each project and activity should take

### Use Story Points rather than hard deadlines

### Use Scrum Sprint Planning template

* + <https://monday.com/templates/scrum-planning>

## How to stop the Scrum board from getting overwhelming:

### Use separate boards for separate parts of the project

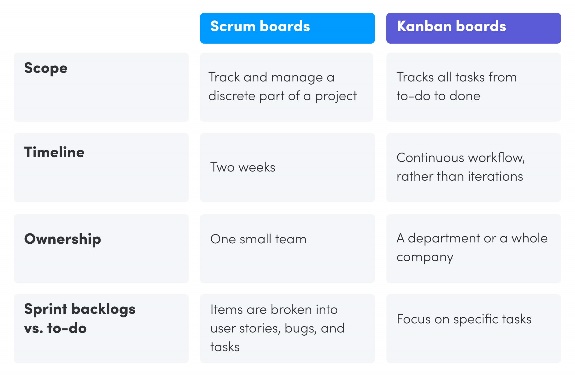
### Don’t become too ambitious when planning your Sprints

### You don’t want that are too long(4 days+) or too short(1-2 hours)

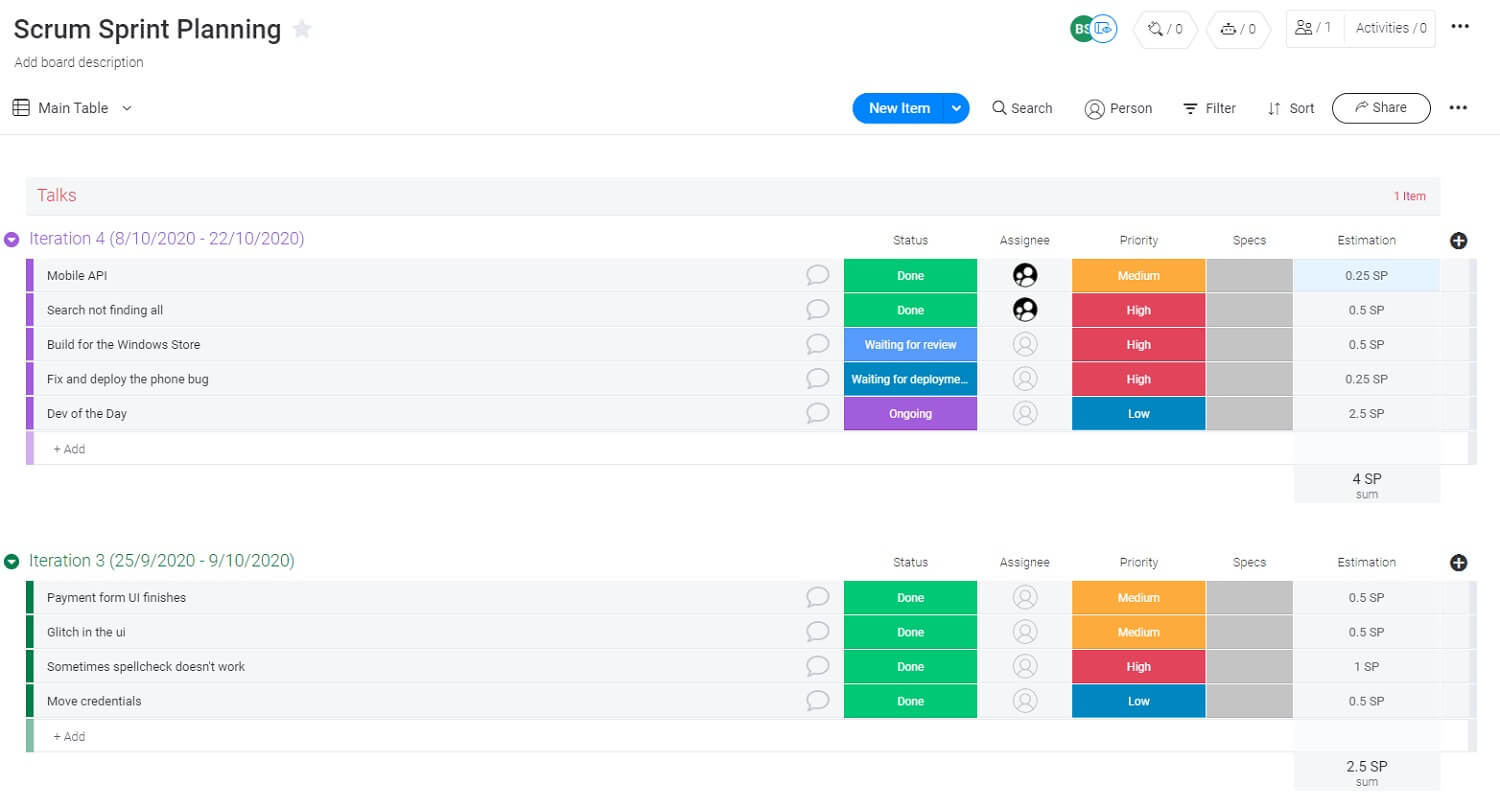
### Only import relevant user stories from your main product backlog

### Don’t include too much text/information(link or attach a doc)

## Difference between Scrum and Kanban board:



## Example of Scrum board:



# -Team Procedures-

## Required Team Procedures

### All team members participate

### The final Meeting Log MUST include

* + a signature by the customer
  + Must include an accurate team presented % proportion of total assigned effort graded for each team member
  + Signed by each member

### Any team document must begin with team name in the file name, site name, etc.

### All individual’s submission must be accurate copies that are the same for all team members

* + include the student’s name in submission file
  + include the team’s name in submission file

### All team Sprint Project Document submissions must be in .pdf file format

### The customer(shareholder) must be an administrator of the project site which contains

* + Files at all stages of project available for view and comment by the stakeholder

### You are required to

* + NOT write complete functioning code *early* in this project
  + Test your ideas using code snippets
  + Use a prototype solution so that all aspects of your design can be reviewed and approved by the customer prior to extensive development or when a new feature is added
  + Following must be simulated clearly and functionally
    - Data representation
    - Events in GUI
    - Product interaction
    - Etc.
  + Evolutions are presented in ever increasing detail from the freehand sketch to the final working copy

## Sprints:

### Sprint 1

* + Link to document pdf will be uploaded by each team member to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.

### Sprint 2

* + Link to document pdf will be uploaded by each team member to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.

### Sprint 3

* + Link to document pdf will be uploaded by each team member to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.

### Sprint 4

* + Link to document pdf will be uploaded by each team member to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.
  + Link to code development file(s) to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.
  + If more than on file is required, they must be grouped in a zip

### Sprint 5

* + Link to document pdf will be uploaded by each team member to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.
  + Link to code development file(s) to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.
  + If more than on file is required, they must be grouped in a zip

### Sprint 6

* + Link to document pdf will be uploaded by each team member to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.
  + Link to code development file(s) to their team submission space as a .txt file or direct entry of an active root folder Team GDrive link.
  + If more than on file is required, they must be grouped in a zip

## General Requirements:

### All team members need to read the entire text on schedule with the course

### Make sure that they read the chapters of the book very early in each Sprint

### Conduct regularly scheduled team meetings

### Meet with customer to answer questions that arise from team meetings

### Make a request by assigning a task to the customer requesting a review

### Record all team meetings **team meeting form**

### Assign and track ANY AND ALL tasks required for each team member for each sprint

* + Include estimates of the number of hours required to complete the task
  + Track accuracy of time estimates
  + Adjust time estimates accordingly as time goes on

### Manage all aspect of the project

### Determine very early on the project approach(sub-version of agile development)

### Follow the Agile Manifesto

## Additional Suggestions

### Watch this video <https://www.pbs.org/video/living-on-auto-pilot-5p5jct/>

### Survey each team member as to their

* + Strengths
  + Interests
  + Weaknesses

### Trade members if to many members have the same skill set

### Ideas for Team roles

* + Scribe and Milestone document builder
  + Information Manager/Researcher
  + Team Manager
  + Coder
  + Milestone Document Contributor
  + Schedule Manager
  + GUI Developer
  + Q&A Manager

# -Project Intro Document-

## The Project:

### Start with the Employee Class project that you completed in CS1410

* + Will serve as a beginning prototype that you will develop further to solve some of the major features that the Employee class did not address

### Features for the project

* + Simple and effective user interface using GUI
  + More data that is required as part of a useful employee record
  + The ability to edit the employee’s data
  + The ability to hide certain aspects of the employee’s data from those who shouldn’t see it
  + The ability to add new employees
  + Ask customer for further details

## Submission Requirements:

### Any submission can receive a preliminary review and comment

* + By creating a review task for the Stakeholder and notifying them accordingly

### Each member MUST read Chapters 1, 2, & 3 by the listed due dates

### Sprints will be submitted in canvas by each team member as a link to complete pdf document, on time and with exactly the same content as the others. There must be a title page that lists the name of the project, the course, the date, the team number, and the team members with their roles listed.

test

# -SPR-1 Document-

## Strongly recommended focus of team efforts for sprint 1:

### Number 1

* + Determine overview roles for each team member
  + Familiarize each other with team members
  + Assess whether a trade is needed

### Number 2

* + Develop Sprint products development site location
  + Add the professor to the approved file access list/use

### Number 3

* + Select one or two candidates for the prototype from CS1410
  + Files are also in the course Project Files folder

### Number 4

* + Everyone needs to be up to speed on the slides and chapters 1, 2, 3

### Number 5

* + Develop key focus from the book, the slides, and brainstorming thoughts developed from research on the problem and course readings
  + Use the book’s table of contents to help outline the course efforts

### Number 6

* + Set up team meetings site in MS Teams
  + Set up team file location
  + Setup dispersal online site cloud drive

### Number 7

* + Make sure everyone is properly set up in Canvas and is aware of Canvas contents

### Number 8

* + Get copies of course materials

### Number 9

* + Develop roles for team members and determine if any trades need to happen
  + Sketch up and/or build templates for
    - Diagrams
    - Illustrations
    - Plans
    - Reports

### Number 11

* + Set up a template available to all team members that allows them to record time estimated and time actual for each task

### Number 12

* + Assemble the above content in a professional Project Document and submit for Sprint 1
  + Assign the monitoring, progress and completion of each of the above to the team members.

# -SPR-1 Thoughts-

## General observations:

### #1Some teams often overdo SPR-1

* + You do not need to design the entire project at this point in time
  + This is part of the design stage
  + You are designers
  + Present preliminary ideas and possible choices that explain what your team is facing
  + Beginning of the development of an outline that you will hang your course development on

### #2Code comments should be developed

### #3Assign team members the role of Chapter Expert for each Chapter

* + They should be responsible ahead of the actual course schedule for
    - Reading
    - Highlighting
    - Explaining overview of contents of chapters
  + Chapter expert should be outlining the possible approaches that you might use and commenting upon the chapters in the book that they are responsible for
  + Include in every milestone
  + Have outlines of the content for the major components of Software Engineering that you will be Including in your milestone reports
    - Brief explanation of why you are considering a particular approach
    - How you intend to implement those approaches
    - Anything you are thinking about using that is under further investigation
    - Type of discussion about the design or designs that you are presenting as options for the customer to consider
    - Use chapter slides and the study of chapters to develop your startup approaches

### #4If you meet with stakeholders as a research effort select a couple of people from your team to go to the meeting

* + Try to plan your meeting with other teams so you can go as a group
  + Come prepared with questions and show a general idea of what you want to do

### #5Chapter 3 of the book provides an overview of the entire effort for this course

* + Begin filling in questions and thoughts using the book table of contents right away

### #6All team members must turn in a link to the PDF of the assignment submission page in your Canvas Sprint Modules

test

# -SPR-1 Rubric-

## Basic setup:

### Determine that team roles and overview foci are established

* + Listed on the cover pages (roles)
  + Main topic in a table of contents

### File storage organization and location are established including a share to the shareholder

### Prototype candidates are established

### Team specific communication tool(s) are established with a share to the shareholder

## Listing of individual procedures, directions, concerns, and considerations

### Slide highlights that *may* apply are

* + Recognized
  + Included
  + Commented

### A team task list of work highlights has been initialized and developed demonstrated as a beginning Work Breakdown Structure

### Procedures for developing the sprint have been surveyed and included where applicable

### Chosen agile alternative listed and discussed as to pros and cons applicable to the team’s approach and skills when compared to a pure Scrum approach

testing

# Beginning team and product requirements specs started from research:

### Developed from prototype, brainstorming, and research for necessary fields

### Beginning MoSCoW chart for template

### Distribute one to each team member developed individually

### Listing and approach to key Sprint Documents contents and principle features

## Developed a commented and organized SPR-1 layout of the project for use in developing the upcoming Sprint Documents:

### Project management

* + Developed a beginning Work Breakdown Structure with overview roles
  + Developed a beginning Pert Chart and beginning Gantt Chart templates
  + At a minimum sketched of each of the above charts
  + Determined if a team trade board effort would be required
  + Developed a time tracking method that documents estimated and used manhours that will be part of each of the Sprint
  + Developed meeting minutes form and record all meetings using the form developed from the template

## Had at least one meeting with the Stakeholder

### In team communication space and documents are available from the shared document warehouse for the shareholder to review and comment on

## Developed a consistent format with exemplary form that is completed to the proper detail level for the sprint document

# -Lesson1.pptx-

## Overviewing:

### Requirements Gathering

### High-Level Design

### Low-Level Design

### Development

### Testing

### Deployment

### Maintenance

## Using course slides:

### Expected team efforts are in gold

### Suggested considerations of effort and/or content are in blue

### Requirements Gathering

* + Identify customers
    - What is it they do and what will they need
    - What do they currently use
    - What will they recognize as an improvement
  + Write down requirements
    - What can you pre-determine through advanced knowledge
    - Interview the customer(s)
    - Research standard methods
    - Review other software examples
  + Refine requirements so they’re precise enough for developers
    - You should be to describe what you are
      * Gathering
      * Assuming
      * Expecting
      * Planning
    - Develop key communication tools
  + Develop content that communicate with both developers and customers
  + Track requirements
  + Refine occurs over time and is never finished
  + Document everything
  + Track revisions

### High-Level Design

* + Determine major subsystems
    - Accommodate the work environment
    - Address accessibility
    - Address user standards
    - Consider general costs
    - Determine compatibility
  + Classes
    - Define overall class needs to manage the use case scenarios
    - Identify potential class methods for the purposes of the classes
    - Research UML diagrams that you will need and start working on each type
  + User interface concept
    - Develop an interface approach that
      * is most likely to deliver the breadth of the project’s needs
      * is self-explanatory
      * is expandable and flexible
      * works for the widest range of hardware you might encounter
      * the use of which is likely to be familiar but accommodates the software
  + External interfaces
    - Determine what the user will expect to interact with
      * Social media
      * Website(s)
      * Hardware
      * Other companies
    - Security
    - Required flexibility and connectors
  + Team Management
    - Roles of team members
    - Project data location(s)
      * Versioning
      * Format
    - Communication
    - Requirements management and documentation
    - Work Breakdown Structure
    - Agile Project Development Research
    - Project Development tools
      * IDE
      * Charts
      * Forms
      * GDrive
  + Test everything
  + Track bug status
  + Learn how to develop plain English coding
    - Natural language user interface-wiki

### Low-Level Design

* + Refine high-level design until pieces can start to be implemented
  + Assign sprint components to team members so that low level design can begin
  + Includes information about how each part of the product might work
  + Manage revisions by confirming the reactions between parts of the project
  + Update requirements spec to address the current state of the project
  + Review the design with the user(s)
  + Update the documents and diagrams to match the current state of the project
  + Verify and validate current design
  + Attach time estimates to work breakdown structure
    - Develop start of a Pert chart
    - Develop start of Gantt chart
  + Develop start of an Agile plan
  + Develop the start of a Testing plan
  + Resources
    - [Pert Chart](https://www.investopedia.com/terms/p/pert-chart.asp) definition
    - [Pert Chart](https://app.lucidchart.com/documents) template
    - [Gantt Chart](https://en.wikipedia.org/wiki/Gantt_chart) definition
    - [Gantt Chart](https://app.lucidchart.com/documents) template
    - [CPM/Gantt Chart](https://www.inloox.com/company/blog/articles/the-importance-of-the-gantt-chart-and-the-critical-path-for-project-management/) definition

### Development

* + Outline the code
  + Extend the UML class Diagrams
  + Extend the use case diagrams
  + Develop a set of sequence diagrams
  + Refine the requirements specs
  + Refine the high- and low-level designs until it makes no more sense to not being writing code
  + Refine the estimates
  + Test and design of the UML diagrams
  + Develop a beginning test plan
  + Test the code concepts and logic
    - Where would the errors probably arise
    - Test the extremes and oddities
    - Be mean and evil with your what-ifs
    - Record all of the results
  + Track bug evolution in detail
    - Record them as they are found
    - Record their status
      * Dead
      * Open
      * Under repair

### Testing in steps

* + Developers test their own code
  + Testers who didn’t write the code test the code
  + After testing merge with project and see if anything broke
    - Unit testing
    - System testing
    - Regression testing
  + Bug tracking
    - Maintain a bug tracking database
  + More testing
    - [Free bug tracking software](https://backlog.com/lp/bug-tracking/?utm_source=adwords&utm_medium=cpc&utm_campaign=bug_tracking_e&utm_term=free%20bug%20tracking%20software&gclid=CjwKCAjwltH3BRB6EiwAhj0IUPqiUx6yVzG1WNkoqnltY5OmsqD2ta3Qqi71XukNqDRHt9WA6nlr7RoCS54QAvD_BwE) + backlogs
  + Develop a testing plan

### Deployment

* + The application is installed for the users
  + Requirements may include
    - Database servers
    - Network
    - Computers
    - Training
    - Support
    - Parallel operations
    - Data conversion
    - Maintenance
    - Bug fixes
  + Might involve
    - User training
    - Maintenance of old database
    - User discovered bugs
    - Unpredictable stuff
  + Develop an outline plan that touches on the relevant tasks and possibilities that evolve with the project

### Maintenance

* + Develop an outline maintenance plan that evolves as the project progresses
    - Bug fixes
    - Changes
    - Additions
    - Enhancements
  + Wrap-up plan from documents
    - Gather information about the project – continually record your project documentation
      * What went right
      * What went wrong

# -Lesson2.pptx-

## Document Management:

### Historical documents

### E-mail

### Code

### Code documentation (docstrings/inline comments)

### Application documentation

### Meeting minutes/videos

### Contract documents

### Minutes/Memos/Correspondence

### Quality control, Testing, etc. plans

### Project management artifacts

### User manual

### Collection of decision makers

### Sharable to all team members

### Editing control

### Get most recent

### Dates and version numbers

### Tags

### Comparison of documents for changes

### Standardize tools, format

### Identify author, sources

## Historical documents:

### Change management/change orders

### Cost estimates

### Time and effort

### Expenses

### Meeting minutes

### Design process

### Bug tracking

### Scope management

## Code:

### Track bugs and task progress

### Must contain comments and docstrings

### Must be preserved during evolution

### Naming formats

### Python Style Guide

### Viability, verifiability, and validation commentary

### Continuity between high-level and low-level design

### Consistent evaluation criteria

### Describe summary concept and track revisions

## Application documentation:

### Must contain features in an evolutionary progression

### Naming formats (table of contents)

## Classification organization category ideas:

### Admin

### Rqts

### HL-Design

### LL\_Design

### Dvt

### Test

### Deploy

### Train\_App

### Maint

### Wrap

### Choose consistency not creativity focused

## General comments:

### Can assign someone different to write comments

### Use Google Docs

* + Turn on change tracking and comparison tools
  + Immediately name or rename the file to a new version number if it isn’t automatic
  + Choose a product that allows the addition of comments without modifying the principal content

# -Lesson3.pptx-

## Project Management:

### Help define requirements

### Track tasks

### Respond to unexpected problems

### Manage risk

### Keep stakeholders up to date on progress

### Interface between customers and team members

### Manage resources

* + Time
  + People
  + Budget
  + Etc.

### Manage delivery

### Requirements Spec

### Work Breakdown Structure

### Pert Chart

### Gantt Chart with Slack time

## Work breakdown structure:

### Assign roles

### Describe groups of responsibilities for the project tasks

### Start with a roundtable discussion and sticky notes on a wall or MS teams screen

* + [Free apps](https://asoftclick.com/best-sticky-notes-for-windows-10/)

### First consider major areas of requirements

### Devise major categories for requirements

Diagram

Description automatically generated

## Pert charts:

### Diagrams a project so that viewers can grasp the effective order of events

### Build a continually evolving chart from the Requirements Spec and WBS

* + Chapter 3 in book

### Building Pert charts

* + Place the start task in a ready pile
  + Other tasks in a pending pile
  + Move Ready tasks to the right
  + Look through the pending pile and cross out predecessors that you just positioned
  + Repeat from step 2

### Critical Path Methods

* + A longest path through the Pert chart network that becomes the schedule determining path
  + Label tasks with their distance from the start
  + This distance will be used to lay out the Gantt chartDiagram

    Description automatically generated

## Gantt charts:

### Show time relationships among tasks

### Displays the schedule of the entire project or each of the major components of the entire project

### Use a calendar!

### Predicting times

* + Experience
  + Break up large tasks
  + Look for similarities and differences
  + Allow for the unexpected
  + Track progress and take action if necessary
  + Break unknown tasks into simpler pieces
  + Research the tasks with experienced sources
  + Capital is time not money

A picture containing table

Description automatically generated

## Risk Management Plan

### For each task determine

* + Likelihood
  + Severity
  + Consequences
  + Workarounds
  + Include
  + Slack time in your considerations(LIES!!!)
  + WORK FILLS THE TIME ALLOTTED!
  + Convert the Gantt chart info into a trackable context

# -Lesson4.pptx-

Requirement Gathering:

* Requirement Characteristics
* Requirement Categories
* Common Requirements
* Gathering Requirements
* Refining Requirements
* Recording Requirements
* Requirements
  + A description of features you must provide
  + Requirements are gathered from numerous sources
    - Research
    - Interviews
    - Brainstorming
    - User testing
    - Governing Standards
    - A Delphi/MoSCoW process
* Requirements must be
  + Clear
  + Unambiguous
  + Consistent
  + Prioritized
  + Verifiable and Validated
* MoSCoW Method
  + Must
    - Required features
  + Should
    - Important features that will be included in a release
  + Could
    - Desirable but discard-able
  + Won’t
    - Will not be included
  + Build a voting sheet of suggested requirements
  + All team members complete a form on their own
  + Return a beginning Requirement Spec
  + Don’t add public judgement/opinion at this stage
    - Focus on quantity
    - Withhold criticism
    - Encourage left field thinking
    - Combine and improve ideas
    - Use a variety of brainstorming techniques
* Requirement Categories
  + Audience-Oriented
    - Business
      * Research comparable within and outside of the customer’s environment
    - User
      * Must have user tests
    - Functional vs. Nonfunctional
      * Desired capabilities vs. specific quality requirements
    - Verifiable and Validated
      * Requirement must be self-contained and precisely defined and must be confirmed as included at each step of development
* FURPS+
  + Functionality
  + Usability Reliability
  + Performance
  + Scalability
  + Design constraints
  + Implementation requirements
  + Interface requirements
  + Physical requirements
* Common Requirements
  + Screens
  + Menus
  + Workflow
  + Login
  + User types
  + Audit tracking
  + Archiving
  + Configuration
* Gathering Requirements
  + Listen to customers
    - Record their expectations and adapt to their concerns
    - Five Ws and one H
      * Who
      * What
      * When
      * Where
      * Why
      * How
    - Study users
    - Interview model users that are not customers
* Refining Requirements
  + Copy existing systems
  + Clairvoyance
  + Brainstorm
  + Process of moving from goals to requirements
* Recording Requirements
  + UML Diagramming
  + User stories
    - How will it be used and what for
  + Use cases
    - Actors interacting with product features
  + Prototypes
    - Quick and dirty representations of the solution ideas
  + Requirements specification
    - Follow the requirements for a workable requirement spec
  + Validation and Verification
* Validation and Verification
  + Validation
    - Are we doing the right things?
  + Verification
    - Are we doing the things right?

# -Lesson6.pptx-

## OO Design:

* Specific classes
* Identify classes
* Inheritance Hierarchies

### Refinement

* + Breaking a parent class into multiple subclasses
  + avoid over refinement

### Generalization

* + The opposite of refinement
  + develop collections of classes into one group with the same parent

## Database Design:

### Types of Databases

* + Text(delimited)
    - comma delimited file(csv)-delimited with a consistent separator
    - Not very flexible or powerful
    - Simple list processing for short projects
  + Relational
    - Stores related data in related tables (by using keys) which are made up of records
    - Columns are fields, Rows are records
    - Database design must avoid problems
    - Duplication of data should be avoided
    - Links between data records can limit removing an unused record
    - Multiple values may be limited even when necessary
    - All known as anomalies
    - Refer to chapter 6 in book
    - Tables contain records
    - Records contain fields
    - Relations connect records
    - Keys
    - Lookup tables
  + Hierarchical
    - Records are stored as a tree structure with links that establish connections
    - Each child only one parent
  + Object store
    - NoSQL type database (no tables) that is made of objects
  + Document
    - A semi-structured of document oriented info
  + Graph/network
    - Similar to a hierarchy database except the database is viewed as a graph using nodes to present objects and lacks the use of a hierarchy storage scheme
    - Records have on parent and many children but the network model allows each record to have multiple parent and child records
  + Key/value
    - A data storage design that uses a hash table (dictionary) to collect objects(records)
    - The key/value record can have multiple formats and data types
  + Database poplarity

### Database normalization

* + formatting data to avoid anomalies
    - 1st Normal Form
    - 2nd Normal Form
    - 3rd Normal Form
    - view chapter6 content for details

# -Lesson8.pptx-

## Goals:

* Reduce the effect of bugs on users
  + typical 15-20 bugs per KLOC (Thousand lines of code
* At a minimum catch the most irritating bugs

## Reasons Bugs Never Die:

* Diminishing returns
  + The ‘fresher’ the development is the easier it is to find bugs. As time and development go on, finding the bugs vecomes more difficult
  + Fixing bugs can destroy the profitability of the software
* Deadlines
  + Bean counters are focused solely on the cost of getting to a deadline. Deadlines become immovable objects except in the worst of cases
* Too soon
  + Development staff rarely is in a position to determine the deadline – the role of Software Engineering is to manage this condition
    - Security flaw – patch immediately
    - Cussing at the computer more than once a day – patch monthly
    - Occasional smirks – patch semi-annually
    - A New Feature problem – At the next release
* Usefulness
* Not a bug
  + is it simply a user perception problem?
* It never ends
* Fixing bugs is dangerous
  + Someimes a bug fix breaks more than it fixes
  + Requires a thorough understanding of what the program is to accomplish and how it accomplishes it
  + One of the primary reasons for good documentation of the code
  + The bug may make outher features work correctly

## Bug Priorities – When to fix:

* Analyze on a cost/benefit basis using categories such as:
  + Severity – the cost to users
  + Workarounds – are there any?
  + Frequency – measure the user tolerance
  + Difficulty – How problematic and thus costly
  + Riskiness – How independent is the source of the problem from other parts of the program

## Levels of Testing:

* The sooner bugs are found the easier they are to fix since more functionality often gets attached to the problem code
  + Unit test – Verifies the correctness of the code – are we building the correct thing and does it work correctly? A key concept that can be tested incrementally
    - V&V component
    - Requires test code
    - Test user actions and UI interaction, test instructions to the user
    - Allow testing to remain in the program and turn on and off
  + Integration test
    - Verifies that a new method works well with others
    - A new method can be properly called and if necessary can call existing code
    - included is regression testing which tests the program’s entire functionality to see if anything is working differencly after adding each new code componenet or bug fix
    - Integration testing follows a unit test
  + Automated testing
    - Can automate load tests
    - Uses pre-developed or COTS testing software and can run tests on a schedule
    - Can provide warnings before more code is developed
  + Component interface testing
    - Studies the interaction between components – A state chart is helpful in this process
    - Contributes to the design process – what messages aren’t being passed hthat should be
  + System testing
    - Tests the total of all program functions
    - Requires a selection of sequences that make use of every activity that the program performs
  + Acceptance testing
    - One of the key components of V&V
    - Should be confirmed at many of the development stages

## Other Testing Categories

* Accessibility test
  + Management of disabled a user’s access to the program’s purpose
* Alpha test
  + Early round testing – makes use of “friends of the developers” to review the program in the beginning stages
* Beta test
  + Semi-release to a test group
  + Can be considered a marketing technique
  + Cheap bug testing
  + Has helpful activites for users
* Compatibility test
  + focuses on environment applicability
* Destructive test
  + Forces application failure to test worst case error handling
* Functional & Nonfunctional tests
  + Tests functional and nonfunctional requirements spec items
* Installation test
  + Tests installation on new computers
* Internationalization test
  + Confirms global applicability

## Testing Categories

* Performance test
  + Runs a performance test under pre-defined conditions (work load, low memory, legacy systems, etc)
* Security test
* Usability test
  + Tests the acceptability and intuitive responsiveness of the user interface
  + Suggested Basic testing plan and results minimums
    - V&V
    - Assert
    - User Acceptance Testing
    - Unit Testing
    - Functional/Non-Functional
    - Peer testing
    - Usability Testing
    - QA Testing

## Testing Techniques

* Exhaustive Testing
  + Within reason test the range of possible inputs – from extreme values to every possible input
* Black-Box testing
  + Throw any possible input you can think of to a program that you don’t understand (simulated ignorance)
  + Develop random inputs and run a script
* White-Box Testing
  + Use code knowledge to try to devise inputs that will fail
* Gray-Box Testing
  + A combination of Black and White box testing

## Testing Habits

* Test and debug when alert
* Test your own code before you release it
* Have someone else test your code
* Fix your own bugs
* Think before you change
* Don’t believe in magic
* See what changed
* Fix bugs not symproms
* Test your tests
* Treat all testers as your best friend – you cannot be perfect so take their efforts as being helpful not a criticism

## Egoless Programming:

* Mistakes happen
* You are not your code
* Someone always knows more karate
* Don’t rewrite without consultation
* Treat people with respect, deference, patience
* Change is the only constant
* Authority stems from knowledge not position
* Fight for your beliefs but accept defeat gracefully
* Don’t hibernate
* Critique code not people

## Estimating Bugs:

* Tracking
  + Designate bug fixes as tasks to be completed – assign id’s and track progress of updated tests
  + Use a testing tool that measures code coverage
    - number of “visits” to pieces of code
* Seeding
  + The practice of purposely scattering bugs throughout the code
  + It’s easiest to create simple bugs but complex bugs are just as important
  + Lincoln Index
    - Graphical user interface, text, application

      Description automatically generated
    - E1…# of tester discovered errors
    - S - # of errors in common
    - Where E = probability \* bugs found for each

Lincoln Index

* + -E1,E2, - Total bugs found by testers 1 and 2
  + S = Bugs found in common
  + Graphical user interface, text, application

    Description automatically generated

# -Lesson9.pptx-

## The Deoployment Plan

### Scope

* + # of users
  + Data content
  + External systems reliance
  + LOC – Inhouse, Legacy, Open Source, COTS

### Develop a deployment plan that encompasses the scope

* + How will you recover from an emergency
    - Depends upon thorough planning
    - Requires similar experiences
    - A bad plan or bad scope can stop deployment in its stracks
    - Requires a proper backup of all project content

## Implementation, installation, and release

## Rollback

* + Lets you undo a venture down the wrong street

## Cutover Strategies

* + Moving users over to the new system

## Common Tasks

* + The normal tasks for deployment

## Common Mistakes

* + A disagreement on what done means

## Sized to fit the project

## List the deployment steps

## For every task:

* + What could go wrong
  + How would you handle that
  + Where’s the point of no return

## Rollback

* + What’s the rollback strategy
  + How painful will it be
  + Where’s the point of no return?
  + Where’s the point of defeat?

## Cutover Strategies

* + Staged Deployment
    - Build a fully functioning staging setup
    - Test the setup on different environments
    - Turn power users loose on the setup
  + Gradual cutover
    - A gradual cutover Gantt chart
      * A picture containing diagram

        Description automatically generated
    - use select groups to receive the new setup
    - Add one user at a time until everyone is using the new setup
    - Can create confusion between old system and new system users and what they are production – especially in information format conflicts

## Incremental Deployment

* + Requires a program that is compatible with existing conditions and that offers features with not all features in effect

## Parallel Testing

* + If your program supports multiple concurrent users you would have a selected group using your program along with the existing data next to a previous working program so that features(new) would work well with (old).

## Common Deployment Plans Tasks

* + Physical Environment
    - Are the existing featureds compatible in terms of environment support
  + Hardware
    - Confirms, with a checklist possibly, that the supporting hardware is available to each installation
  + Documentation
    - Does there need to be different versions installed for the users that provide the proper information for different versions
  + Training
    - Will this include training for the users
  + Database
    - Database is compatible, needs to be revised or added to, or requires a database conversion application
  + Other Software
    - What if other software in use causes your program to fail
  + Your Software
    - What all needs to be included with the deployment

## Common Mistakes

* + Assume everything will work
  + No rollback plan
  + Insufficient time
  + Unspecified point of defeat
  + Skip staging
  + Install many updates all at once
  + Unstable environment
  + Early point of no return

# -Chapter4-

## Requirements defined:

### Requirements

* + Features your application must provide
  + Guide development
  + Verify the finished application does what it’s supposed to

### Clear

* + Good requirements are
    - Clear
    - Concise
    - Easy to understand
  + Cannot be vague or ill-defined

### Consistent

### Unambiguous

### Prioritized

### Verifiable

* + Requirements must be limited and precisely defined

## MoSCoW:

### M-Must

* + Required features that must be included

### S-Should

* + Important features that should be included if possible
  + May need to schedule for release 2

### C-Could

* + Desirable features that can be omitted if they can’t fit in the schedule
  + Can be pushed back till release 2 or 3

### W-Won’t

* + Completely optional
  + Customers have agreed will not be included in the current release
  + May be included in a future release if time permits

## Requirement Categories:

### Audience-Oriented requirements

### Business requirements

* + Lay out project’s high-level goals
  + Explain what the customer hopes to achieve with the project

### User requirements

* + Describe how project will be used by eventual end users
  + Include sketches or forms
  + Scripts that show the steps users will perform to accomplish specific tasks
  + Explain what needs to be done not how it will be done

### Functional requirements

* + Detailed statements of the project’s desired capabilities

### Non-functional requirements

* + Are statements about the quality of the applications behavior or constraints on how it produces a desired result

### Implementation requirements

* + Temporary features that are needed to transition to using the new system but that will be later discarded

## FURPS:

### Functionality

* + What the application should do
  + Describe the system’s general features including
    - What it does
    - Interfaces with other systems
    - Security
    - So forth

### Usability

* + What the program should look like
  + Describe user-oriented features such as the
    - applications general appearance
    - ease of use
    - navigation methods
    - responsiveness

### Reliability

* + How reliable the system should be
  + Indicate such things as
    - when the system should be available
    - how often it can fail
    - how accurate the system is

### Performance

* + How efficient the system should be
  + Describe such things as
    - The application’s speed
    - Memory usage
    - Disk usage
    - Database capacity

### Supportability

* + How easy it is to support the application
  + Include such things as
    - How easy it will be to maintain the application
    - How easy it is to test the code
    - How flexible the application is

## FURPS+:

### Design constraints

* + These are constraints on the design that are driven by other factors such as
    - Hardware platform
    - Software platform
    - Network characteristics
    - Database

### Implementation requirements

* + These are constraints on the way the software is build

### Interface requirements

* + These are constraints on the system’s interfaces with other systems
  + They tell what other systems will exchange data with the one that you are building
  + They describe things like
    - the kinds of interactions that will take place
    - when they will occur
    - the format of the data that will be exchanged

### Physical requirements

* + These are constraints on the hardware and physical devices that the system will use

## Common requirements:

### Screens

* + What screens are needed?

### Menus

* + What menus will the screen have?

### Navigation

* + How will the users navigate through different parts of the system?

### Workflow

* + How does data mover through the system?

### Login

* + How is login information stored and validated?

### User types

* + Are there different kinds of users?

### Audit tracking and history

* + Does the system need to keep track of who made changes to the data?

### Archiving

* + Does the system need to archive older data to free up space in the database

### Configuration

* + Should application provide configuration screens that let the system administrators change the way the program works?

## Gathering requirements:

### Listen to customers and users

* + Focus on problem not on customer suggestions
  + If customer asks for something odd or unimportant ask them why
  + Take lost of notes while you’re listening to the customers
  + If a customer requirement seems odd, dig a bit deeper to find out why

### Use the Five Ws (and One H)

* + Who
    - Who will be using the software
      * Get to know as much as you can about those people
      * Learn as much about the users as possible
  + What
    - Figure out what the customers need the application to do
    - Focus on the goals as much as possible rather than the customers’ ideas about how the solution should work
  + When
    - Find out when the application is needed
    - If the application will be rolled out in phases, find out which features are needed when
    - Use Gantt charts and other techniques from the book to figure out how much time is actually needed
    - Compare your timeline with the customers’ desired timeline. If the two don’t match talk to the customer about some stuff
  + Where
    - Find out where the application will be used
  + Why
    - Ask why the customers need the application
    - Clarify the customers needs and see if it is real
  + How
    - Pay attention to the way that things are currently being done

## Refining requirements:

### Copy existing systems

* + Often you can use many of the behaviors of the existing system as requirements for the new one

### Brainstorm

* + Gather as many ideas as possible, not worrying about their quality or practicality. After you assemble a large list of idea, you examine them more closely to see which deserve further work
  + Get a diverse group of participants for different ideas
  + No critiquing anyone’s’ ideas
  + Focus on quantity
    - Do everything think you can to keep the ideas flowing
  + Withhold criticism
    - Criticism can make people stop contributing
  + Encourage unusual ideas
    - You can always “tone down a wild idea” but you may need to think way outside the box to find really creative solutions
  + Combine and improve ideas
    - Form new ideas by combining other ideas or using one idea to modify another
  + Brainstorming techniques
    - Popcorn
      * People just speak out ideas as they occur to them.
    - Subgroups
      * Break the group into smaller subgroups

# -Chapter5-

## High-Level Design:

### Provides a view of the system at an abstract level

### Shows how the major pieces of a finished application will fit together and interact with one another

### Specify assumptions about the environment in which the finished application will run

### Describes the hardware and software you will use to develop the application

### Describes the hardware that will eventually run the program

### Does not focus on details of how the pieces of the application will work

## The Big Picture:

### The goal is to divide the system into chucks that are self-contained enough that you could give them to separate individuals/teams to implement

## What To Specify:

### Can have overlaps with requirements but with more detail

### Security

* + Operating system security
    - Type of login procedures
    - Password expiration policies
    - And password standards
  + Application security
    - Providing the right level of access to different users
  + Data security
    - You need to make sure your customer’s credit card information doesn’t fall into the hands of Eastern European hackers
  + Network security
    - Cyber banditos might steal your data from your network
  + DON’T OVERDUE SECURITY WHERE PEOPLE START TO UNSAFE THINGS IN THE NAME OF CONVIENCE

### Hardware

* + You can build systems to run on
    - Mainframes
    - Desktops
    - Laptops
    - Tablets
    - Phone
  + Hardware to specify
    - Printers
    - Network components
      * Cables
      * Modems
      * Gateways
      * Routers
    - Servers
      * Database servers
      * Web servers
      * Application servers
    - Specialized Instruments
      * Scales
      * Microscopes
      * Programmable signs
      * GPS units
    - Audio and video hardware
      * Webcams
      * Headsets
      * VOIP

### User interface

* + Indicate main methods for navigating through the application
  + Describe special features such as
    - Clickable maps
    - Important tables
    - Methods for specifying system settings
  + Address general appearance issues such as
    - Color schemes
    - Company logo placement
    - Form skins

### Internal interfaces

* + When you chop the program into pieces you should specify how the pieces will interact
  + The teams assigned to pieces can work separately without needing constant coordination
  + Specify internal interfaces clearly and unambiguously
  + Try to write internal interface before you start coding

### External interfaces

* + Many applications must interact with external systems

### Architecture

* + Describes how its pieces fit together at a high level
  + Monolithic architecture
    - A single program does everything
      * It displays the user interfaces
      * Accesses data
      * Processes customer orders
      * Print invoices
      * Launches missiles
      * Does whatever the application needs to do
    - Requires you to understand how all the pieces of the system fit together from the beginning of the project
    - Don’t need to worry about network security
  + Client/Server architecture
    - Separates pieces of the system that need to use a particular function (clients) from parts of the system that provide those functions (servers).
    - Decouples the client and server pieces of the system so that developers can work on them separately
    - The clients and server communicate through some network such as a local area network, wide area network, or the internet
    - The client is separate from the server

### Reports

* + Reports that deal with customers
    - Who’s buying
    - Who has unpaid bills
    - Where customers live
  + Reports that deal with products
    - Inventory
    - Pricing
    - What’s selling well
  + Users
    - Which employees are selling a lot
    - Employee work schedules
  + Reports don’t need to be detailed, figure out what reports you want and get the details in the low-level design

### Other Outputs

* + In addition to normal reports you should consider other kind of outputs that the application might create
    - Generate printouts
    - Web pages
    - Data files
    - Image files
    - Audio
    - Video
    - Output to special devices
    - E-mail
    - Text messages

### Database

* + Decide what kind of database the program will need
  + Specify whether the application will store data in
    - Text files
    - XML files
    - Full-fledged relational database
    - Or something else
  + Even a program without a database needs to store data in
    - Arrays
    - Lists
    - Or some other data structure
  + If you decide to use an external database you should specify the database product that you will use
  + If you use a relational database, you can sketch out the tables it contains and their relationships during high-level design

### Audit Trails

* + Keeps track of each user who modifies a specific record
  + Auditing can be as simple as creating a history table that records a
    - User’s name
    - A link to the record that was modified
    - The date when the change occurred
  + Fancier version of auditing can
    - Store copies of the original data in each table when its data is modified
  + You may not need auditing for
    - Programs written for internal company use
    - Programs that don’t involve money
    - Programs that don’t involve Confidential records
    - Programs that don’t involve other data that might be tempting

### User Access

* + Many applications also nee to provide different levels of access to different kinds of data
  + Build a table listing the users and the privileges they should be given

### Database Maintenance

* + Overtime the database can become cluttered with outdated records
  + Move older data to long-term storage
  + You may need to periodically re-index key tables or run database tuning software to restore peak performance
  + Design a database backup and recovery scheme

### Configuration Data

* + Let users define their own ad hoc queries
  + Provide configuration screens so that users can fine-tune the application without making you write new code

### Data flows and States

* + A data flow diagram shows how data flows through various processes
  + Describe the system
  + Describes the way processes interact with the data

### Training

### UML

* + Defines several kinds of diagrams that you can use to represent different pieces of the system
  + Defines 13 diagram types divided into three categories (and one subcategory)
    - Structure diagram
      * Class Diagram
      * Composite Structure Diagram
      * Component Diagram
      * Deployment Diagram
      * Object Diagram
      * Package Diagram
      * Profile Diagram
    - Behavior Diagram
    - Activity Diagram
    - Use Case Diagram
    - State Machine Diagram
    - Interaction Diagram
      * Sequence Diagram
      * Communication Diagram
      * Interaction Overview Diagram
      * Timing Diagram
  + Structure Diagrams
    - Describes things that will be in the system you are designing
      * Class Diagram
        + Shows relationships among the classes that will represent objects in the system such as

Inventory

Items

Vehicles

Expense reports

Coffee requisition forms

* + - * + Describes the classes that make up the system, their

Properties

Methods

Relationships

* + - * + Is represented by a rectangle
        + Class’s name goes at the top, is centered, and is in bold
        + Two sections below the name give the class’s properties and methods
        + Include the data types of properties and parameters passed into methods
        + Show relationships among classes
        + Lines connect classes that are related to each other
        + A variety of \_\_\_\_\_ give more information about the kinds of relationships

Line styles

Symbols

Arrowheads

Annotations

* + - * Object Diagram
        + Focuses on a particular set of objects and their relationships at a specific time
      * Component Diagram
        + Shows how components are combined to form larger parts of the system
      * Composite Structure Diagram
        + Shows a class’s internal structure and the collaborations that the class allows
      * Package Diagram
        + Describes relationships among the packages that make up a system
      * Deployment Diagram
        + Describes the deployment of artifacts on nodes
        + Artifacts

Files

Scripts

Executables

And the like

* + - * + Nodes

Hardware devices

Execution environments that can execute artifacts

* + Behavior Diagrams
    - UML defines three kinds of basic behavior diagrams
      * Activity diagrams
        + Represents work-flows for activities
        + Include several kinds of symbols connected with arrows to show the direction of the work-flow
      * Use Case diagrams
        + Represents a user’s interaction with the system
        + Show stick figures representing actors connected to tasks represented by ellipses
      * State Machine diagrams
        + Shows the states through which an object passes in response to various events
        + States are represented by rounded rectangles
        + Arrows indicate transitions from one state to another
        + A black circle represents the starting state and a circled black circle indicates the stopping state
  + Interaction Diagrams
    - Are a subset of activity diagrams
    - Include
      * Sequence diagrams
        + Shows how objects collaborate in a particular scenario
        + Represents the collaboration as a sequence of messages
      * Communication diagrams
        + Shows communication among objects during some sort of collaboration
        + The communication diagram focusses more on the objects involved in the collaboration
      * Timing diagrams
        + Shows one or more objects’ changes in state over time
      * Interaction overview diagrams
        + Is basically an activity diagram where the nodes can be frames that contain other kinds of diagrams

# -Chapter6-

Relational Databases

* stores related data in tables
  + each table holds records that contain pieces of data that are related
* The pieces of data in each record are called fields
  + each field has a name and a data type
  + all values in different records for a particular field have that data type
* common problems(anomalies)
  + Duplicate data can waste space and make updating values slow
  + you may be unable to delete on piece of data without also deleting another unrelated piece of data
  + An otherwise unnecessary piece of data may need to exist so that you can represent some other data
  + The database may not allow multiple values when you need them

Database normalization

* a process of rearranging a database to put it into a standard form that prevents anomalies
* First Normal Form
  + 1NF
  + says the table can be placed meaningfully in a relational database
  + The table has a sensible down-to-earth structure like the kind your grandma used to make
  + official requirements
    - Each column must have a unique name
    - The order of the rows and columns doesn’t matter
    - Each column must have a single data type
    - No two rows can contain identical values
    - Each column must contain a single value
    - Columns cannot contain repeating groups
* Second Normal Form
  + 2NF
  + official requirements
    - It is in 1NF
    - All non-key fields depend on all key fields
* Third Normal Form
  + 3NF
  + official requirements
    - It is in 2NF
    - It contains no transitive dependencies
      * is when a non-key field’s value depends on another non-key field’s value

# -Chapter7-

## Source Code Control

* This is essential
* Enables you to go back through past versions of the software and see exactly what changes were made and when
* If a program stops working, you can pull out old versions of the code to see which changes broke the program
* Preven multiple programmers from tripping over each other as they try to modify the same code at the same time

## Profilers

* let you determine what parts of the program use the most time, memory, files, or other resources

## Static Analysis Tools

* study code without executing it
* focus on code’s style

## Refactoring Tools

* rearranging code to make it easier to understand, more maintainable, or generally better

## Training

* important

## Selecting Algorithms

* common algorithms
  + Sorting and arragning pieces of data
  + Quickly locating items in databases
  + Finding optimal paths through street, power, communication, or other networks
  + Designing networks to provide necessary capacity and redundancy to prevent single points of failure
  + Encrypting and decrypting data
  + Picking optimal investment strategies
  + Finding least cost construction and production strategies
* Top-Down Design
  + Step-wise refinement]
    - start with a high-level statement of a problem, and you break the problem down into more detailed pices
    - Examine the pieces and break any that are too big into smaller pieces
    - continue breaking pieces until you have a detailed list of the steops you need to perform to solve the original problem
  + Be on the look out to save some work
    - don’t duplicate code
  + if descripition of task is too long turn it into a method
  + Descripitions should be very very very short
* Tips for writing code
  + Be Alert
    - Code during your most productive hours of the day
    - Do the busy work during the hours you are more likely to fall asleep at

# -SPR-2 Document-

## Sprint development:

### Establish assignments for ALL team members

### At the end of this Sprint you should have a comprehensive document for all aspects of this project that will be iterated and detailed further in SPR-3 while code is developed which will probably reveal more that you need to be doing

### The project shall first be developed as a requirement testing prototype

### The project will ultimately be developed as a working product that makes use of a Human Compatible Graphic User Interface(HCGUI)

### There must be examples demonstrating additional instructional assistance provided to the user that includes

* + A help tool
  + Pop up tips
  + A User’s Manual
  + Configurable explanatory output that mirrors the results derived from manipulating the GUI by the user
  + This output should serve to inform the user of the effect that the changes they make in using the interface has upon the results

### The HCGUI should be developed in a conceptual design format using the form tools in your IDE and should represent your first thoughts about how the program will work and what it will display

### Outline what you think will work for your design

* + You may not use this

### The functions of your project that must be included as components of your design for this sprint are

* + Those discussed in class
  + Included in your requirement spec
  + Indicated in a special color for the slides accompanying this Sprint in Canvas

## Of utmost importance in this Sprint is a comprehensive Requirements Engineering effort:

### One of the components of your Sprint document will be a listing of a Requirements spec using a series of one-line statements enumerating the possible needs you might satisfy with your design grouped by the Actor.

* + These will start with the result of the requirements analysis required for SPR-1 which represents a baseline

### Your will refine your requirements spec during this sprint and finalize your baseline in this Sprint

## Requirements form the following actor’s needs:

### Your developments team’s efforts

### Other expectations that would be placed upon the team if you proceed with the final development of this project

### Maintenance

* + Distribution
  + Installation
  + Testing
  + Bug fixes
  + Etc.

### The user

* + What you learn from your research regarding how the user can best make use of a managing a professional level employee database
  + How will you make sure that they can see what they need to
    - See
    - Control identity visibility
    - Maintain privacy
    - Make sure that the data that is entered is correct

### The Stakeholder

* + The needs expressed by the primary Stakeholder gathered during interviews

### The IT department

* + What hardware will you have to support?
    - Currently laptop or PC
  + What systems are likely to be available in a business environment

### The HCGUI must provide an intuitive, logical, and concise pipeline to the use of the data

### The requirements engineering spec should be refined and developed based upon the improved and updated requirements that came from SPR-1

### You should complete a few iterations of High-Level Design for this Sprint

## Requirements Focus:

### We are starting out with the first attempt at a requirements spec

### Review your team member MoSCoW charts and develop a near final single MoSCoW chart

* + This chart should include all of the tasks that are facing the development team as well as the various users shown in the **Chapter 4 slides**

### Provide a link to the file(pdf) to the stakeholder showing expert recommendation for the project requirements including improvements to the prototype that your team has derived from your research

### The shareholder will make comments on the proposed list

### You may split the charts into sections based upon time or by Sprint but keeping track of duplicates can best be handled in an overview chart first

* + Sorting the chart as one chart can be easier

### Set up a connection between your tasks and your requirements such that you can track the completion via the completion of the set of tasks necessary of that requirement

### Develop an ongoing burndown list from that data

### Categorize by user your requirements between function and non-functional

* + Within those categories group the requirements by Must, Should, and Could haves

### Develop Use Case Scenarios from those categorizations

### Describe how you might solve the Use Case scenario in plain English

### Meet with the user for a final discussion of the requirements spec.

### Complete a concisely descriptive requirements spec.

### Keep your meeting minutes up to date

### Develop a method to include the collection of meeting minutes in your PDF file

### There must be a summary form for each meeting with the requisite factors discussed and signatures

### Assign 1 or 2 teammates to review and suggest documentation software for helping to create the diagrams that your Sprints will require

## High Level Design Focus:

### High-level design shows how the major pieces of the finished application fit together and interact with each other

### Address the highlighted issues found in SPR-1

### Expand the elements of SPR-1 that apply to SPR-2 so that they

* + Correct SPR-1
  + Cover SPR-2
  + Show a more detailed development some of which you will derive from your HLD efforts
  + Update your list of data fields as a result of your MoSCoW evaluations

### We are still in the assumption phase

* + You should consider, in your strategies, any assumptions you can make

### Complete your final review of your requirements spec with the shareholder and the shareholder has formally stated their approval on the requirements spec document

### Your task is to chop up your design up into self-contained pieces that are small enough to implement

### Items to specify in HLD are

* + Security
    - OS
    - App
    - Data
    - Network
    - Physical
  + Hardware
  + GUI design
  + Internal interfaces
  + External interfaces
    - Tkinter/TTK
    - GitHub
    - PyQt
  + Architecture
  + Reports/outputs
  + Database design and issues
  + Training
    - User manual
  + UML diagrams
    - Class diagram
    - Use case diagram
    - State machine or logic flow diagram
    - Sequence diagram
    - Any other useful ones
  + Prepare concise but effective discussion and diagrams for/on each item

### Your collection of management plans should be developed in terms of the

* + Side effects
  + Purposes
  + Possible concerns that are needed to prepare them
  + Do research on wiki
    - Software testing/wiki
    - Software quality/wiki
    - Software management/wiki
    - Software risk management/wiki
    - Verification and Validation/wiki

### Work with the first item in this section’s list in mind

### You should have the ‘code’ defined as comments and doc strings from the prototype with notes about required revisions

### New comments and Docstrings for the new requirements

### Description/discussion of your code thoughts in plain English for the HLD completed at the end of this topic section

test

# -SPR-2 Work Preview-

Requirement Gathering:

* Begin a MoSCoW form using the Delphi methods to develop a Team MoSCoW form very early in this sprint
* Have at least one meeting with the Stakeholder
* Review team MoSCoW form with the stakeholder based upon submitted pdf file comments and prepare final requirements spec accordingly
* Begin requirements specs from the
  + Prototype requirements
  + Project type research
  + Brainstorming thoughts developed in SPR-1
* Formalize the requirements using final MoSCoW and interview with Stakeholder
* Study the document preparation in the 2 samples provided in the course materials

Project management:

* Develop Work Breakdown Structure with overview roles as the key area development
* Develop Pert Chart and Gantt Chart templates using the Sprint efforts as timeboxes
  + At a minimum sketch each of three above charts
* Are you going to use classes for the GUI? Can you develop templates for each class type?
  + Low-Level design
  + Development
  + Testing
  + Deployment
  + Metrics
  + Maintenance
  + Record any
    - Definitions
    - Thoughts
    - Ideas
    - Concerns for each
* Develop a meeting minutes summary form and MS Teams record all meetings and fill in the highlights using the form
* Choose a couple of Agile process models as candidates for discussions developing pro and con comparisons of the chosen process models each used with Scrum
  + Example:
    - Combine Kanban with the required Scrum concepts
    - Develop a training assistance approach for team members so that they can be ready for the next Sprint responsibilities in code development
    - Develop a consistent format for the Sprint document and prepare the submission

High-Level Design

* Outline coding approach using the original Docstrings and comment lines and adding new Docstrings and comment lines for all recognized components and potential components
* Add plain English code development that satisfy the Docstrings and comment lines
* Test ideas through peer review
* V&V that you have covered requirements in general form

# -SPR-2 Thoughts-

* Make sure you know what an **outline and short explanation** means for requirements specifications
* Place specifications by Actor (there are at least 5 in this case) even if there is duplication
* Your use case diagrams should also be ‘by Actor’
  + Put use case diagrams with specifications
  + Work breakdown structure starts with this information followed by a Pert Chart then a Gantt chart
* File should be in color
* Consult stakeholder to make sure requirements are clear and concise
* Read SPR-2 document carefully
* You will achieve a final project baseline from the Stakeholder’s review of you two sections that represent complete SPR-1 & SPR-2 documents
* You will have an updated baseline from the new SPR-2 materials that can only be further represented as greater than SPR-2 baseline requirements via approved Change Order
  + Change orders will only be approved if they are in addition to the required effort for this project and its documentation at the beginning
  + The approvals will occur upon review of the total upcoming SPR-2 submission
  + **The only thing you need to prepare for early review at this point is** any CO’s that the Stakeholder has requested…
* Request an approval/review of your GUI
* Need review of your major change ideas in order to prepare a CO

# -SPR-2 Rubric-

## Basic Setup SPR-1 comments Addressed for:

* File storage Organization and location are established including a share to the shareholder
* Prototype candidates are established
* Team specific communication tool(s) are established with a share to the shareholder
* Meetings conducted in team communication space
* Sprint documents available from the shared document warehouse for the shareholder to review and comment on

## Listing of individual procedures, directions, concerns, and considerations:

* Procedures for developing the Sprint have been initiated
  + Document versioning
  + Coding standards
  + Scheduling
  + Consistent formatting
  + Provide the document with a professional appearance

## Sprint Requirements Section:

* Team and product requirements specs reviewed with Shareholder and completed for work to date
* Requirements spec is complete and follows the **chapter 4 slides** for requirements characteristics
* Team MoSCoW chart has been reviewed by the stakeholder, updated, and approved

## Sprint High Level Design Section:

* Key Sprint Documents contents and principle features are addressed based upon required High Level Design phase efforts
  + Updated Software lifecycle stages
  + High level phase processes developed using plain English code
  + Plans are initiated at high level phase
    - QC
    - Risk Management
    - Testing
    - Scheduling
    - V&V
* High-Level design
  + Outlined coding approach using Docstrings and Comment lines to outline all recognized components and potential components
  + Decided whether or not to use classes for the GUI and explain the choice reasoning
  + Prepared Plain English description of code solution
  + Requirements spec is addressed in HLD
  + V & V completed for SPR-2 document and HLD with respect to requirements spec

## Project Management:

* Developed SPR-2 Work Breakdown Structure with overview roles as the key area development
* Developed SPR-2 Pert Chart with the HLD level CPM and HLD level Gantt chart with calendar based timeboxes
* Minimum required UML diagrams are developed that represent the HLD level state
  + Class
  + Use case
  + Sequence State or Logic Flow
* Pert chart has been updated to determine current CPM based on major WBS components
* Team task list has been added to Gantt chart reflecting all major project requirements spec content

## General Document:

* Developed a commented and organized SPR-2 layout of the project refined and complete using the applicable sprint 1 documents as a basis
* Meeting minutes in outline achievement and scheduling form are included with links to corresponding recordings
* File naming
* Team naming
* Cover sheet
* Division of Sprint into daily/weekly sections
* Table of contents
* Page layout
* Developed a consistent format that is completed to the proper detail level for this Sprint document
* SPR-2 submissions meet course published standards

# -Current Sprint Task Management Outline SPR-3-

Requirements Gathering:

* Identify customers
  + What is it they do and what will they need
  + What do they currently use
  + What will they recognize as an improvement

Team Management:

* Requirements management and documentation/ change orders
* Work Breakdown Structure
* Project development tools(IDE, Charts, UML diagrams, forms, GitHub)

Software

* Bug Tracking
* Testing

Team Management forms:

* Pert Chart definition
* Pert Chart template
* Gantt Chart definition
* Gantt Chart template
* CPM/Gantt Chart definition

Document Management

* Tells team members what to do
* Sharable to all team members
* Collaborative including shareholder(s)
* Editing control
* Get most recent
* Dates and version numbers
* Tags
* Standardize tools

Document Management operations

* Fetch most recent version
* Check out/in a document
* Develop document standards and standard forms

Histiorical Documents

* Change control board
* Bug tracking
* Design Process
* Meeting minutes
* Time and effort
* Cost estimates

Low-Level Design

* Develop specific classes
* Build Inheritance Hierarchies
* Database normalization

Development

* Rules for code improvement
  + Begin the use of Pylint and pytest
  + use top down design and develop and test algorithms
  + Reduce into appropriate seize pieces next-Eliminate duplication of effort in coding-Consider separate and conquer when long tasks appear
  + Comment first
    - begin the programming effort by writing the comments before the code
    - use doc strings
  + Write self-documenting code
  + Learn to develop descriptive names for
    - classes
    - methods
    - properties
    - variables
    - communications
  + Eliminate magic numbers wherever you can
  + Keep it small
    - Develop code incrementally
  + Stay focused
    - develop single concept classes and methods
  + Use assertions
    - sometimes used to write the code itself
    - build a tracking record
  + Develop effective error handlers
  + Use tests of invariants(unchanging program states and data)
  + Use #if DEBUG\_1 and #endif
    - turn on and off debugging
  + Use exceptions
    - Write exception handlers first
  + Don’t repeat code
    - Defer optimization
  + For any non-working code create a notification with and error handler
  + The code must run without user interdiction

Testing

* Develop as part of a quality control plan
* Testing Goals
  + Establish Bug Priorities
  + Develop key components of a quality control/bug management plan
  + At a minimum catch the most irritating
    - Recover from bugs gracefully
  + Testing
    - Study a piece of code to determine that it meets the requirements and, nearly impossibly works correctly in all circumstances
* Levels of testing
  + Develop from sample testing plan
  + Coordinate with testing effort estimates
* Bugs
  + Lincoln Index
    - E1…#of tester discovered errors
    - S-# of errors in common
    - Where E = probability \* bugs found for each
      * E1,E2 =Total bugs found by testers 1 and 2
      * S = Bugs found in common
        + L =( E1 X E2)/ S
  + Sometimes a bug fix breaks more than it fixes
  + Requires a thorough understand of what the program is to accomplish and how it accomplished it
    - one of the primary reasons for good documentation fo the code
  + The bug may make other features work correctly
* When to fix
  + Analyze on a cost/benefit basis using categories such as
    - severity
      * the cost to the users
    - Frequency
      * measure the user tolerance
    - Difficulty
      * How problematic and thus costly
    - Riskiness
      * How independent is the source of the problem from other parts of the program
  + Integration testing
    - tests the program’s entire functionality to see if anything is working differently after adding each new code component or bug fix
    - Integration testing follows a unit test
* Acceptance testing
  + should be confirmed at many of the development stages
* Alpha test
* Beta test
* Installation test
  + Tests installation on new computers
* Usability test
  + Tests the acceptability and intuitive responsiveness of the user interface
* Tracking
  + Develop continually updated report using bug tracking software

Deployment

* Scope
  + The purpose of deployment is the development of a plan to handle an upcaoming development risks that will be used once this product is released
  + This is, in effect, a pseudo deployment and yet should address the course description of deployment issues
* The plan
  + Use your developed plan that you would use to ‘hang’ the results of your plan effort on (reports).
  + This plan should allow you to report to you shareholder(s) how the deployment of your product will be handled and why it will solve potential problems normally encounterd by a deployment effort
  + This plan should allow you to report any concerns you have about changes to the reuitements specification and the potential side effects that you are concerned about
  + This plan should advise the stakeholder of your recommendations for accommodating the deployment and what the stakeholder should plan on accommodation during the initiation of the distrubtion of your product
  + This plan should state what will happen if you cannot suffessfully deploy your product due to special circumstances and at what point will you not bea able to adapt the current release to work in the target company
  + you must describe a proposed cutover plan
  + ou must describe how you will handle testing and bug reports made by the employees
  + You must describe how the employees will report suggested requirements specification additions

**NEED TO FINISH READING DEPLOYMENT SECTION…**

# -SPR-3 Work Preview-

* New set of
  + Pert
  + WBS
  + Gantt
* Copy documents to a review folder for the stakeholder to add final comments during the course of the Sprint
* Demonstrate your design ideas such that the stakeholder can determine if they have nearly completed the work for the real project construction and that the features work as desired
* Upate any required changed to your management plans

# -SPR-3 Document-

Requirement changes?

* Confirm the requirements of the development environment and that all addons are functional
* Make sure that your source control is properly and functionally configured(GitHub or GDocs)
* In preparation for Development of final Development, you must complete versions of your chosen algorithms and test vigorously
* Build your developnebt solution using a Top-down design approach
* Update your comments, add new ones, and develop any necessary Docstring improvements as you proceed through refinement
* Present your SPR-3 document and provide a functioning solution containing the major project components in a .py file and a documentassemnly in a .pdf file provided as a link in Canvas. The .py file must run and so you need to manage bugs by interrupting and posting error notices for any code that you don’t have yet working
* Update your bug tracking data in preparation for the upcoming Testing Sprint SPR-4

LLD

* Perform a visual inspection debugging effort and V&V effort
* Confirm that the LLD design will function with the desired Hardware, Network configuration, Development environment, and Source Code Control
* Finalize the database type you intend to use and demonstrate the handling of data using pseudo code. Follow the 1NF requirements for design of th database
* Conduct a V&V exercise for all of your major customer requirements to confirm that you are building the right thing and are building it correctly
* Turn in your pdf such that it contains all of your work for SPR-2 in the sam fashion that you did for SPR-1

Development

* Test your algorithms to make sure that they will work for the desired results
* Perform top-down design to refine your code as you proceed
* Test examples of the classes that you intend to use before developing a full set of them
* Follow incremental development practices
* TEST AS YOU GO
* Shar your work with teram members and expect them to review and comment. All team members will need to participate in the LLD and Development phases with the exception of the necessary document improvements and construction

Documentation

* All required document will need to be updated based upon the HLD efforts. Make revisions to documents as you go which will save much time in the long run
* Maintain the WBS, Pert, and Gantt chart during your Sprint meetings.
* V&V

Checklist

* Include quality control, risk management, testing, and maintenance plans documentation(1-2 pages max each). include in final form your continued development and any additional new plans and results from each book chapter topics(testing, const estimation, planning, etc.)
* Finalization fo the demonstration of the EmpDat solution including use case diagrams that have been modified from the previous Sprint, and the addition of the changes project documents
* Finalization of the functional and non-functional requirements documents including any new recommendation/additions and excluding any completed by change order
* Finalization of the Class diagrams that includes the design of how you managed the integration of the HCGUI wit the classes, functions you have implemented of your solution
* Update any Change Request content starting at the establishment of the baseline stipulated in the Sprint 3 beginning
* Provide a list of any new functional and non-functional requirements your tema would recommend for future consideration
* Confirm that all of your documents required for previous Sprints are up to date as they pertain to additional/updated requirements for Sprint3.
* Provide Decomposed sprints and backlogs for Sprint 3
* Turn in the complete pseudo code in a separate pdf file
* Include the results and documentation of any change orders that your team provided for consideration for extra credit and approval and the subsequent statement that they are included and complete

# -SPR-3 Rubric-

* The Development state of EmpDat must meet the basic needs of UVu as expressed during project communications . This Sprinjt must demonstrate that all features can be properly functioning in the final code form with at least on major component functional
* Updated Requirement specs with change orders if any
* UML classz Diagrams that reflet the contract baseline
* The Class Diagrams should follow these requirements
  + Interfaces, Abstract Classes, Child Classes derived from Base Class are allowed
  + Strictly follow “high cohesion and loose coupling” design guidelines and document how this was implemented
* Decompose Sprint-3 to backlogs and Team Sprints(deadlines from SPR3 Pert Chars/Gantt charts)
* Screen print the backlogs and team sprints and include them with submitted Meeting logs with
  + signatures
  + task assignment
  + management results
  + cost estimation performance
  + scheduling
  + work progress documents
  + change orders
  + All doucements required for SPR-2 should now reflect SPR-3(if the are still applicable)
  + supporting unmodified documents that are necessary for overview of SPR3 should be included but those who purpose have been updated by other documents should be eliminated from the document
* A complete LLD done in pseudo code that demonstrates what is to be completed and that will serve to allow any team member to develop the required methods, classes, input, and output for development
* update all GUI features as necessary to demonstrate current code approach. Capture images of the GUI components and include them in the document
* Implement all unfinished/missing coding for the Developmonet phase in Python wit the GUI components and include them in the document
* Implement all unfinished/missing coding for the Development phase in Python with the GUI in place and projet management documentation for Sprint 3 functionality(a working interface with operational code for all interface components). All major features approved as part of the baseline shall be demonstrable to be in near working order in Python. (prepare and submit demonstrative components of all features of your solution and unfinished details must contain error handling such that the main intent of the program runs without interruption
* Use case diagrams with supporting code snippets and screen shots that represent these features so that someone reading the document could begin to prepare a sales brochure and/or help complete the coding effort.

# -SPR-3 Sample File-

* Document Versioning
  + Gdrive
  + Example of naming format
  + Explaing naming conventions
  + **VIEW DOCUMENT FOR MORE DESCRIPITIVE INFO AND IDEAS**
* Coding standards
  + pylint or black
* Software Management
  + Using Gdrive as a repositore for all project documents and code
  + Requirements become tasks that get put onto the Trello board and are delegated to people based on role and how busy people are
  + Everything is checked by using V&V and any major documents are sent to reviewed by the shareholder
* Risk Management
  + Chart Describing what risks the program could face and the likelihood and severity of said risks accruing
    - additional chart may be needed to state the impact of the risk making it through to delployment and what steps the team is taking to avoid the risk happening
  + commonly a 5x5 table
  + Should only need to be 3x3 for this project
  + State that this is a smaller project so a less detailed risk management plan is being chosen
  + Brainstorm bugs that could make it through to deployment
* Software Quality plan
  + Determine the definition fo ‘done’
  + Follow the V&V process for all documents
  + Have shareholder review all major documents
  + Follow naming conventions/PEP-8 to assure readability and understanding of the code
  + Conisistent communication and scheduled team and shareholder meetings
  + Explain the practices that we follow that help guarrentee a quality product
  + code uploading procedures
* Testing Plan
  + pseudo-code for tests
  + Explain what software we are using to test and how we plan to go about testing
* Bug Tracking
  + explain naming conventions for bug documents
  + explain how someone reports a bug they found
  + explain how team will go about resolving bugs by priority
* Back/log
  + take a picture of the Trello to show what is left in the backlog
* Testing
  + Write tests for current classes that are beign created
  + document which ones pass/fail testing
* Change order request form
  + Request #
  + Date form was submitted
  + approved or not
  + Requirement name
  + Description
  + Man Hours requested for change order

-Current Sprint Task Management Outline SPR-4-

Team Management:

* Work Breakdown Structure
* Project development tools(IDE, Charts, UML diagrams, forms, GitHub)

Software

* Bug Tracking

Team Management forms:

* Pert Chart definition
* Pert Chart template
* Gantt Chart definition
* Gantt Chart template
* CPM/Gantt Chart definition

Document Management

* Tells team members what to do
* Comparison of documents for changes verify that all carry forward and new changes are included at each version break
* Standardize tools, formats (fonts, approach, content)
* Identify author, sources, etc.

Document Management operations

* Fetch most recent version
* Check out/in a document
* Develop document standards and standard forms

Histiorical Documents

* Change control board
* Bug tracking
* Design Process
* Meeting minutes
* Time and effort
* Cost estimates

Low-Level Design

* Develop specific classes
* Build Inheritance Hierarchies
* Database normalization

Development

* Rules for code improvement
  + Begin the use of Pylint and pytest
  + use top down design and develop and test algorithms
  + Reduce into appropriate seize pieces next-Eliminate duplication of effort in coding-Consider separate and conquer when long tasks appear
  + Comment first
    - begin the programming effort by writing the comments before the code
    - use doc strings
  + Write self-documenting code
  + Learn to develop descriptive names for
    - classes
    - methods
    - properties
    - variables
    - communications
  + Eliminate magic numbers wherever you can
  + Keep it small
    - Develop code incrementally
  + Stay focused
    - develop single concept classes and methods
  + Use assertions
    - sometimes used to write the code itself
    - build a tracking record
  + Develop effective error handlers
  + Use tests of invariants(unchanging program states and data)
  + Use #if DEBUG\_1 and #endif
    - turn on and off debugging
  + Use exceptions
    - Write exception handlers first
  + Don’t repeat code
    - Defer optimization
  + For any non-working code create a notification with and error handler
  + The code must run without user interdiction

Testing

* Develop as part of a quality control plan
* Testing Goals
  + Establish Bug Priorities
  + Develop key components of a quality control/bug management plan
  + At a minimum catch the most irritating
    - Recover from bugs gracefully
  + Testing
    - Study a piece of code to determine that it meets the requirements and, nearly impossibly works correctly in all circumstances
* Levels of testing
  + Develop from sample testing plan
  + Coordinate with testing effort estimates
* Bugs
  + Lincoln Index
    - E1…#of tester discovered errors
    - S-# of errors in common
    - Where E = probability \* bugs found for each
      * E1,E2 =Total bugs found by testers 1 and 2
      * S = Bugs found in common
        + L =( E1 X E2)/ S
  + Sometimes a bug fix breaks more than it fixes
  + Requires a thorough understand of what the program is to accomplish and how it accomplished it
    - one of the primary reasons for good documentation fo the code
  + The bug may make other features work correctly
* When to fix
  + Analyze on a cost/benefit basis using categories such as
    - severity
      * the cost to the users
    - Frequency
      * measure the user tolerance
    - Difficulty
      * How problematic and thus costly
    - Riskiness
      * How independent is the source of the problem from other parts of the program
  + Integration testing
    - tests the program’s entire functionality to see if anything is working differently after adding each new code component or bug fix
    - Integration testing follows a unit test
* Acceptance testing
  + should be confirmed at many of the development stages
* Alpha test
* Beta test
* Installation test
  + Tests installation on new computers
* Usability test
  + Tests the acceptability and intuitive responsiveness of the user interface
* Tracking
  + Develop continually updated report using bug tracking software

Deployment

* Scope
  + The purpose of deployment is the development of a plan to handle an upcaoming development risks that will be used once this product is released
  + This is, in effect, a pseudo deployment and yet should address the course description of deployment issues
* The plan
  + Use your developed plan that you would use to ‘hang’ the results of your plan effort on (reports).
  + This plan should allow you to report to you shareholder(s) how the deployment of your product will be handled and why it will solve potential problems normally encounterd by a deployment effort
  + This plan should allow you to report any concerns you have about changes to the reuitements specification and the potential side effects that you are concerned about
  + This plan should advise the stakeholder of your recommendations for accommodating the deployment and what the stakeholder should plan on accommodation during the initiation of the distrubtion of your product
  + This plan should state what will happen if you cannot suffessfully deploy your product due to special circumstances and at what point will you not bea able to adapt the current release to work in the target company
  + you must describe a proposed cutover plan
  + ou must describe how you will handle testing and bug reports made by the employees
  + You must describe how the employees will report suggested requirements specification additions
  + Create a risk management plan for potential deployment mistakes

**NEED TO FINISH READING DEPLOYMENT SECTION…**

# -SPR-4 Work Preview-

## Testing

* Develop a ranking of importance for Bugs and the recommended testing method(s) for each.(must fix, should fix, could fix, can’t fix)
* Choose the Testing Categories and Techniques from your Testing Plan and formalize them
* Estimate the metrics for each bug
* Integrate your prosed solutions with your Qality Control/Bug Management plan. Update your QC and BM plans and produce reports from your BM plan of your bug correction progress and completion
* Divide bug testing and repair among all team members
* Perform user acceptance using team members, Unit Tests, Assertion tests – prepare reporting documents
* Provide a reporting form for team members to use, Update our bug report regularly
* Complete a final V&V report
* Update your ongoing Use Case tests (Combined with User acceptance is a Usability test)
* Treat the overall testing to an intermediate Alpha Test and a final Beta Test
* Use a suggested Bug Tracking tool

## Deployment

* Address the anticipated Physical Environment
* Address the anticipated Hardware
* Address the required Documentatino
* Address the Training Support Methods (Your team members as well as the product training for the users)
* Establish all the needs to be included with your software
* Document for the user via a readme.txt file what is necessary t odeploy your product and assuming that there is the original product that your used as a prototype already in use
* Make sure the definition of *done* is agreed upon by the entire team
* Due to the installed product in existence develop a Cutover strategy and plan. Include a rollback plan and emergency recovery strategy in the plan
* Develop and test a User Manual that is clear, concise, illustrated, and easy to use

## Documentation

* All required documents will need to be updated upon the Development efforts. Much of this is simply adding to what you have previously built. You should make revisions to these documents as yougo which will save much time in the long run
* You need to maintain the WBS, Pert, and Gantt chart during your Print meetings
* Note that testing (especially V&V) should be performed on your documents as well as your code

# -SPR-4 Document-

## Testing and Deployment

* Confirm the requirements fo the development environment and that all addons are functional
* Make sure that your source control is properly and functionally configured
* In preparation for Testing you will need a well organized bug report template
* Update your comments, add new ones, and develop any necessary Docstring imporovements as you proceed through refinement. Update burndown, WBS, Pert, and Gantt/CPM charts, Plans and Reports (especially focused on bug tracking and repair and the deployment plan, point of no return, and rollback plan
* Follow the special color instructions in the slides for Chapters 6 and 7 and confirm that you have implemented all that apply to your project. Chapters 8 and 9 content will also be fully implemented in SPR-4
* Present your SPR-4 document and provide a functinoaing solution in a .py file and a Document assembly in a .pdf file in Canvas. The .py file must run and so your need to manage bugs by interrupting and posting error notices for any code that you don’t have yet working
* With exception of the course provided files which will be held in a UVU testbed folder, your submitted files should be extracted from your supplied zip file and be able to run without error on a windos 10 pc
  + document contains all of the requested information in the Sprints up to and including the Sprint in the current project state
  + Code examples with the use case information should be snippets of code and screen captures of the corresponding GUI
  + The plain English and pseudo code contents are not required but all other applicable artifacts must be in current state updated format
    - an applicable artifact is one that explains, supports, or is created as a result of the current Sprint

## Checklist

* Include updates to your quality control, risk management, testing, deployment, cutover, and rollback plans documentation and reports(2 pages max each for the indepth plans and reports)
* Include in final form your continued development and add additionally new plans and results from each book chapter topics (testing, cost estimation, planning, etc.)
* Finalization of the bug corrected demonstration of the EmpDat solution including new, modified, or explanatory use case diagrams that come from the previous Sprint and the addition of the changed project documents
* Finalization of the functional and non-functional requirements documents including any new recommendations/additions
* Finalization of the Class diagram that includes the methods and parameters and ALL classes. Parent classes must have been fully developed in LLD and all classes needed to be functioning as part of Development so that they can be properly tested
* Update any Change Request content starting at the establishment of the baseline stipulated in the Sprint 3 beginning. Recall that any approved Change orders must be included as a separate project with all relevant documentation
* Provide a final list of any new non-functional requirements your team would recommend for future consideration
* Confirm that all of your documents required for previous Sprints are up to date as they pertain to additional/updated requirements for Sprint 4.
* All documents that apply to the final project should be included with the appropriate updates and revisions
  + burndown for spr3 is not applicable but a burndown for spr4 is required
* OTOH the requirements spec should be revised if necessary and included in Sprint 4 now as a collection of functional and non-functional requirements groupings
* Provide the Decomposed sprints and backlogs for Sprint 4
* Turn in a single link that leads to the team GDrive as a working, operable program copy in one zip file and the documentation in pdf form in a separate zip file, one per team member in the correct sprint subfolder
* To unify the file naming your TEAM SUBMITTAL file that each team member should submit should include only the content shown in this file naming format
  + ex. T4\_SPR4\_CS2450-###-S(.pdf and .zip)
* Include results and documentation of any change orders that your team provided for consideration for extra credit and approval and the subsequent statement that they are included and complete

# -SPR-4 Rubric-

## Testing and Deployment

* IED will insure that a fully representative Class Diagram that includes all bug correction changes is fully represented
* Decompose Sprint-4 backlogs and the sprint.
* Screen print backlogs and sprints and include them with submitted Meeting logs, task assignment and management results, cost estimation performance tracking, scheduling, work progress documents, change orders, etc…
* update SPR-3 documents that are applicable
* WBS, PERT, and Gantt chars
* Update all GUI features as necessary
* Implement all unfinished/missing coding and project management documentation from sprint 3 functionality
* All features approved as part of the baseline shall be demonstratable to be in nearly complete working order
* Prepare and submit demonstrative components of all the debugged code of your solution
* Include use case diagrams with supporting code snippets, and screen shots that represent these features so that someone reading this document could begin to prepare a sales brochure and user manual.
* Include a readme.txt file for a fresh, moderate computer literate user installation
* Submit a set of testing plans and results of the tests run on the up to date code
* plans
  + V&V
  + assert
  + User Acceptance Testing
  + Unit Testing
  + Functional/Non-Functional testing
  + Usability testing
* Quality Assurance review of the Sprint Project Document and the Project Code that has been completed
* Include screen captures and associated code snippets depicting the results of at least one each of the required plans plus any that the team deems appropriate
* Fill out the reports using the forms that you have previously developed
* Follow the previously noted instructions and sprint practices about submitting the working code and Sprint Document as noted in the submissions document in Canvas

# -SPR-4 Sample File-

## Deployment Plan

* Summary
* Recover from emergency
* Roll Back Plan
  + General Rollback M.O
  + Point of No Return
  + Point of Defeat
* Cutover Strategy
  + Training with Admins
  + Testing with only Admins
  + Moving Employees Over
  + Admins Train Employees
  + Review Testing
  + Convert Database
  + Complete Switch to The New Software
  + Finish
  + Image of deployment plan
* V&V for Development
* V&V for Deployment
* User Manual
* Testing plan
* V&V for testing
* Bug tracking plan
* Unit Testing
* Coverage Test

# -Class Lectures-

## 1/31/2022

### Misc. items

* + Every needs to turn in the link or everyone will suffer
  + V&V
    - Look at what’s due in the print at the beginning of the sprint
  + Agree on what “done” is
  + Communicate for 15 minutes every night
  + Use a version tracking product
    - GitHub
    - Google docs folders that say week1, week2, week3
  + TDD
  + Pair Programming
  + Comment Driven Development
  + Refactor
    - Wrote too much without a plan
  + Coding standards
    - PEP-8
    - “BLACK” python add-on
    - Pytest
  + Prepare a requirement specification for the Shareholder after meeting him…

### Project requirements

* + Take document you already had done and paste it into sprint two document
  + In sprint three include stuff in spr-3 document and useful items
  + Treat the prototype as a review, not just saying what it does. But what it doesn’t do, what you would like it to do. And express that you are going to use it.
  + Link is to the root folder, not document
  + If all of sprint one is not complete, need to fix it

### Lecture slide additional info

* + Rad is collection of agile methodologies
  + If choose Kanban
    - Evolve document in two ways
      * Put stuff in it at beginning of sprint
      * Not completed stuff in backlog
      * Stuff not finished goes in next sprint’s Gantt chart

### Questions

## 2/2/2022

* Misc. items
* Project requirements
  + Work breakdown structure
    - Shows who is in charge of making sure certain categories of work get done.
  + Burndown chart
    - Measured by hours on the side
    - Measured by weeks on the bottom
    - Estimated hours vs actual hours spent
    - Measure weekly hours in a separate chart first
* Lecture slide additional info
* Questions

## 2/7/2022

* Misc. items
* Project requirements
  + Decide what can carry over from SPR-2
  + MoSCoW chart can carry over
* Lecture slide additional info
* Questions

## 2/9/2022

* Misc. items
  + Manage time early on in sprint
* Project requirements
  + Get pictures of GUI
  + Should be done with major components of python code
  + have a single window that pops up whenever something is clicked on that isn’t finished
  + Code snippets that show of your program is working
    - and pretty images
  + Work Breakdown
  + Quality Analysis form…
  + Bug tracking software
  + Say Sprint 3 charts
  + Python Wildcards?3e2w
* Lecture slide additional info
* Questions

## 2/14/2022

* Misc. items
  + Pert chart holds only the items for SPR-3
  + Gantt chart has the sub-tasks of the items in the Pert chart
  + Google algorithms
  + be more specific on WBS
* Project requirements
  + Search algorithm
  + Development environment, IDE
  + Pytest
  + Pylint/Black
  + Quality control plan in resource file
* Lecture slide additional info
* Questions

## 2/23/2022

* Misc. items
  + have to fix defects
  + Acceptance testing
    - have a random person use it and see what their opinion on it is
* Project requirements
  + bug plan
    - comment on bug priority
  + bug reports
  + bug summary
  + testing plan
  + test report
  + test summary
* Lecture slide additional info
* Questions

## 3/2/2022

apendice for final sprint??

## 3/16/2022

* Misc. items
* Project requirements
  + metrics
    - estimated time for bugs
    - when bug needs to be fixed by
  + Alpha test and beta test
    - Describe how you intend to integrate and use employees to report in a alpha test and a beta test
      * in deployment
  + Address if and or how we would train new users on the software
  + point of no return, due date for changes to be made
* Lecture slide additional info
* Questions