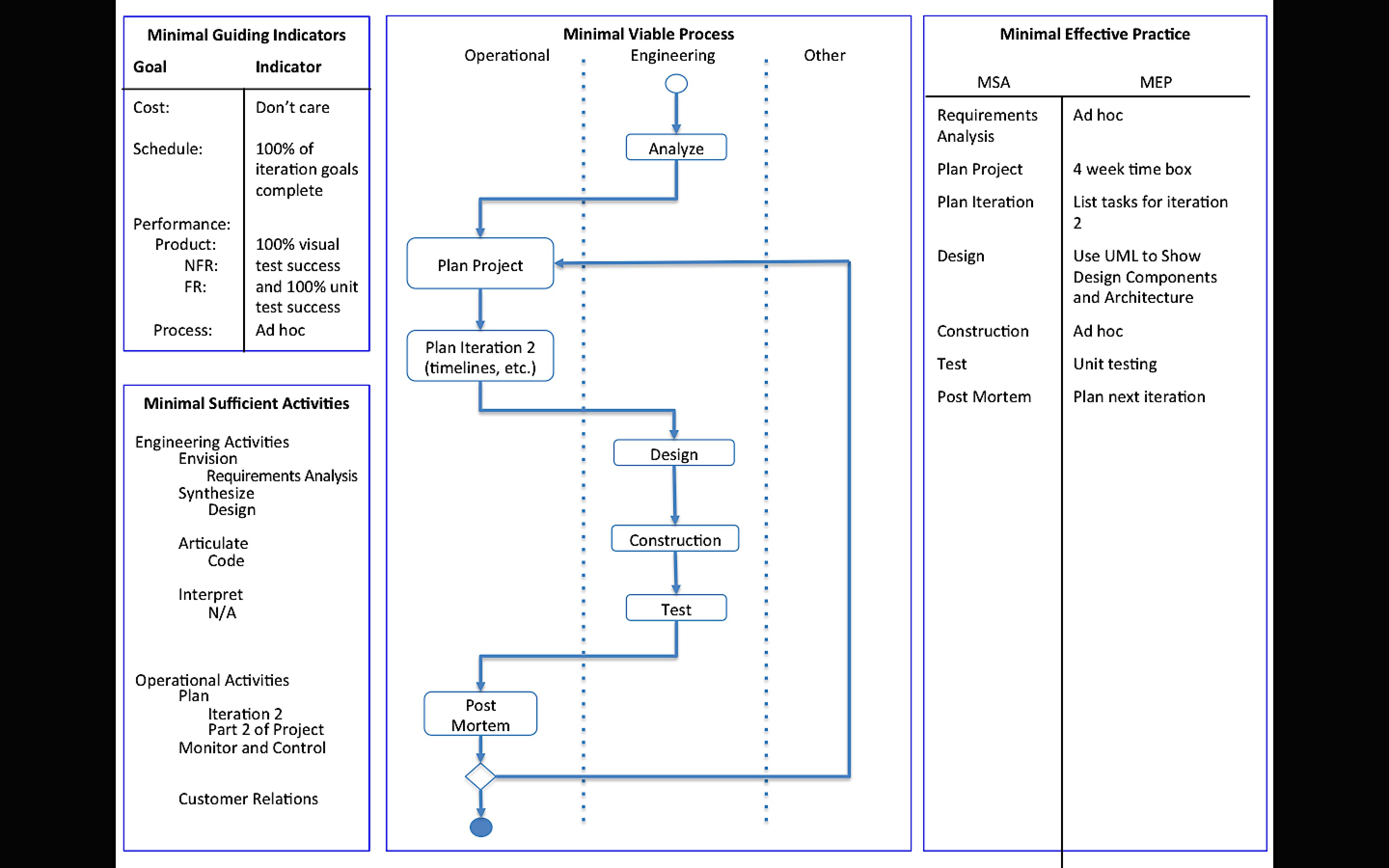
Iteration 2

# Process for Iteration 2



The process above has been slightly updated. The requirements analysis was thorough enough to only be completed once (during the first iteration). Hence, the analysis is not repeated for every iteration. For performance measurement, I will be performing unit tests along with the visual examination of the images. Also, during the design phase, I will be constructing a UML diagram via a PlantUML plugin to show the design components and architecture of the system.

# Requirements Analysis

The requirements have not changed since Iteration 1. They are as follows:

Given a secret image file and two innocent image files, the tool should

* Be able to read in image files and store the pixel information
* Use the extended visual cryptography scheme to encode the secret image pixels into the two innocent images
* Store the encoded images in new image files
  + The filenames and location can be specified by the user. If not, the files are named share1 and share2 and gets stored on the Desktop.

Given two encoded image files, the tool should

* Be able to read in the files and store the pixel information
* Use the extended visual cryptography scheme to decode the secret image from the encoded images (similar to super imposing them)
* The image revealing the secret gets stored in a new image file
  + The filename and location can be specified by the user. If not, the file is named secretMsg and gets stored on the Desktop.

The visual cryptography tool will only work with PNG and JPEG images. The images involved with the encoding process must have the same dimensions. The tool can handle images of any coloring.

# Plans for Project

Iteration 1:

* Create a graphical user interface
* Get the tool working for strictly black and white images
* Test the tool to check the quality of the encoded shares and the decoded message

Iteration 2:

* Research and implement the visual cryptography scheme with grayscale images
* Add features to project to help boost robustness (i.e. add in checks to keep the user from breaking the tool easily)

Iteration 3:

* Add the ability to encode and decode multicolor images

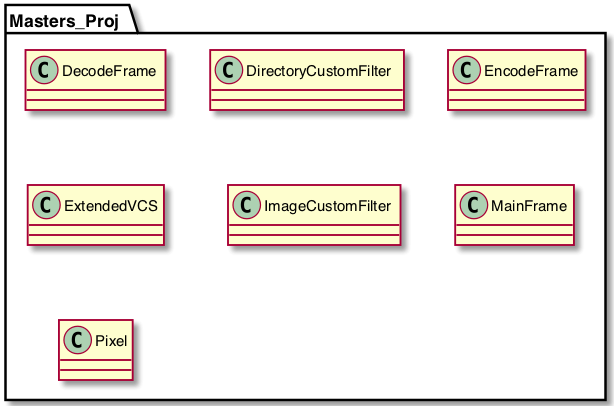
Iteration 4:

* Analyze the tool and look for ways to improve efficiency (performance and memory storage)

# Plans for Iteration 2

* Research current extended visual cryptography techniques and halftone visual cryptography techniques for handling grayscale images
* Based on the research findings, update the current cryptography scheme to improve the encryption and decryption of grayscale images
* If time permits, add more robustness to the tool, such as input validation

# Design

Before changing the Java files to handle the grayscale images, I exported the PlantUML diagram of the visual cryptography tool. Figure 1 shows the class relations.

The research uncovered multiple ways of hiding grayscale images within other grayscale images. One technique utilizes several forms of visual cryptography. First, it breaks the secret image up into *n* shares. If those *n* shares were super imposed, then the secret image would be revealed. This first step is the visual cryptography scheme with pixel expansion. Then the innocent images need to be generated to have the same dimensions as the secret shares. Finally, you embed the secret shares into the generated images. To decrypt and reveal the secret image, you just have to superimpose the embedded shares. A second technique uses pixel expansion and error diffusing. Some other variations use dithering to group the pixels and treat them as strictly black and white images.

Figure 1: UML Diagram from Iteration 1

I plan on trying to merge multiple techniques. Since, I want to take the grayscale capabilities as a baby step towards using colored images, I need to look at methods that utilize the RGB values. I plan on taking the idea of breaking the secret image into secret shares to be embedded in the innocent images. As stated with the project requirements, my tool will only allow for two innocent shares. Hence, I only need to break the secret into two shares. That being the case, I propose grabbing the RGB value from the secret image and randomly splitting the gray concentration. With the image being grayscale, the red, green, and blue values will be the same. If the concentration was 128, then I would calculate a random number between 0 and 128 inclusive to be the gray concentration on the first secret share. The second secret share would be the original concentration minus the first share’s gray concentration.

As for embedding the shares with the innocent images, I plan on summing the gray concentrations of the two images into a new embedded image. I am unsure of how well this will work, so I may make adjustments while constructing based on the visual test results. Any changes will be documented in the construction portion.

For decrypting the secret message, I will initial try just adding the gray concentration of the two images. Depending on the visual test results, I may also try averaging the two concentration values. Again, changes and the reasoning for the changes will be discussed in the construction section.

# Construction

Unfortunately, due to time constraints from my calendar external to the project, I was unable to make progress in the construction of the visual cryptography tool.

# Test

With the scheduling issues, I was unable to make the unit tests for this visual cryptography tool.

# Post Mortem

During Iteration 2, my workload outside of this project greatly escaladed, so I was unable to put to complete my iteration goals. I did complete the design portion, where I determined how I planned to add the encryption and decryption functionality with grayscale images.

Since this iteration did not go as planned, I will need to reevaluate my project plan during the beginning of Iteration 3. I will also need to figure out how I plan to make up the time lost from this iteration.

# Source Code

Not included since there was no change from Iteration 1.