Level 100 Pikachus

Josh Jacobs
Jon Churchill
Chris Knight

Table of Contents

Acronyms

- *GUI* – Graphical User Interface
- *ASML* – Automated Sentry Missile Launcher
- *INI* – Initialization file extension
- *XML* – Extensible Markup Language file extension
- *Mt. Dew* – Mountain Dew
- *VS* – Visual Studios
- *Prof* – Professor
- *XP* – Extreme Programming
- *TDD* – Test Driven Development
- *SVN* – Subversion
- *URL* – Uniform Resource Locator

Roles and Responsibilities

Josh:

- Git/GitHub merge/repository master – Manages merges in the repository
- In charge of AMSL hardware – Keeper of the AMSL
- Stakeholder – Substantial stake in grade
- Developer – Writes code
- Tester – Tests code

Jon:

- Beverage master – Maintains continuous supply of beverages(Mt. Dew) for team
- Code master – Tiebreaker on who does what in terms of coding
- Stakeholder – Substantial stake in grade
- Developer – Writes code
- Tester – Tests code

Chris:

- Scrum master – Makes sure that things are getting done when they should be
- Quality control – Makes sure that acceptance criteria is met
- Stakeholder – Substantial stake in grade
- Developer – Writes code
- Tester – Tests code

Risk Definitions

Assessment

- Extreme – Without these, project fails entirely
- High – Without these, project can work but with major flaws
- Medium – Without these, project is fully functional but not complete
- Low – Minor point losses if these don't get done
- Trivial – No impact on functionality, just bells and whistles

Priority

- ∞ - Needs to be done first, everything else runs on this
- 100 – Needs to be done, but needs items in ∞ to work
- 1 – Would be nice if done, but not crucial
- -∞ - Do these after a guaranteed 100% on the project

Mitigation Framework Definitions

- Avoid – Completely ignore until everything else is done
- Share – Split up between team members
- Reduce – Split into smaller chunks
- Retain – Deal with it by spending more time on it

Risks Matrix

Risk	***Degree***	***Priority***	***Resolution***	***Mitigation***
Lights go out	Extreme	-∞	Avoid	Mit_01
Unable to integrate plug-ins	Extreme	1	Retain	Mit_02
Unable to detect targets	Medium	100	Retain	Mit_03
Unable to display live video feed	Medium	100	Share	Mit_04
Camera failure	Medium	-∞	Avoid	Mit_05
Unable to create GUI	Extreme	∞	Share	Mit_06
Unable to analyze images from video	Medium	100	Share	Mit_04
Can't calibrate	High	100	Retain	Mit_03
Unable to learn multi-threading	Extreme	-∞	Avoid	Mit_07
Other team and their shenanigans	Trivial	-∞	Avoid	Mit_08
Unable to display target outlines	Low	1	Retain	Mit_03
Unable to reset timer	Low	1	Retain	Mit_03
Can't figure out target distance	Medium	100	Share	Mit_04
Unable to stop once started	Extreme	1	Share	Mit_04
Turret acceleration doesn't work	Low	1	Share	Mit_04
Team breaks up	Extreme	-∞	Avoid	Mit_09
Unable to play custom sounds	High	1	Share	Mit_10
Unable to fire missiles	Extreme	∞	Share	Mit_04
Prof. unable to install our work	Extreme	∞	Avoid	Mit_11
GitHub problems	Extreme	∞	Retain	Mit_12
Doesn't shut down after 2 minutes	High	1	Retain	Mit_03
Improper time management	Extreme	∞	Reduce	Mit_13
Unable to read INI/XML files	High	100	Share	Mit_04
C#/Visual Studios problems	Extreme	∞	Avoid	Mit_14
ASML hardware malfunction	Trivial	-∞	Avoid	Mit_15

Mit_01 – If lights go out, we will see if our program still works. If it doesn't, we will try using our INI/XML reader. If that doesn't work we will complain to the professor that this is not fair.

Mit_02 – If we are unable to integrate the plug-ins, we will ask the professor how to integrate the plug-ins. (Or google)

Mit_03 – We will continue to work put time into the algorithm. Continue testing until it works.

Mit_04 – We will continue to work put time into the algorithm. Continue testing until it works, but we will split this up amongst the team to help this go faster.

Mit_05 – If the camera fails, we will simply fall back on the INI/XML reader.

Mit_06 – If we are unable to create the GUI, we will have the team work on it together to ensure that it gets created.

Mit_07 – If we can't learn multi-threading we will keep asking the professor questions until we have learned multi-threading.

Mit_08 – If we suspect that the other teams have plotted anything that could be detrimental to our dominating of the competition, we will let the professor know after severe retaliation has been resolved.

Mit_09 – If the team breaks up, we will complain to the professor that this is bullshit and demand a passing grade at bare minimum.

Mit_10 – If we are unable to play custom sounds we will split the work up in the group to try different ways of getting the program to play custom sounds.

Mit_11 – If the professor is unable to install our work we will blame his computer, and Windows. Mainly Windows. (Windows sucks)

Mit_12 – We are 100% sure that there will be many GitHub problems, and because of this, we will be unable to avoid them. To get around this, we will each keep a backup copy of the repository on our computers so that we can still have our code able to be worked on.

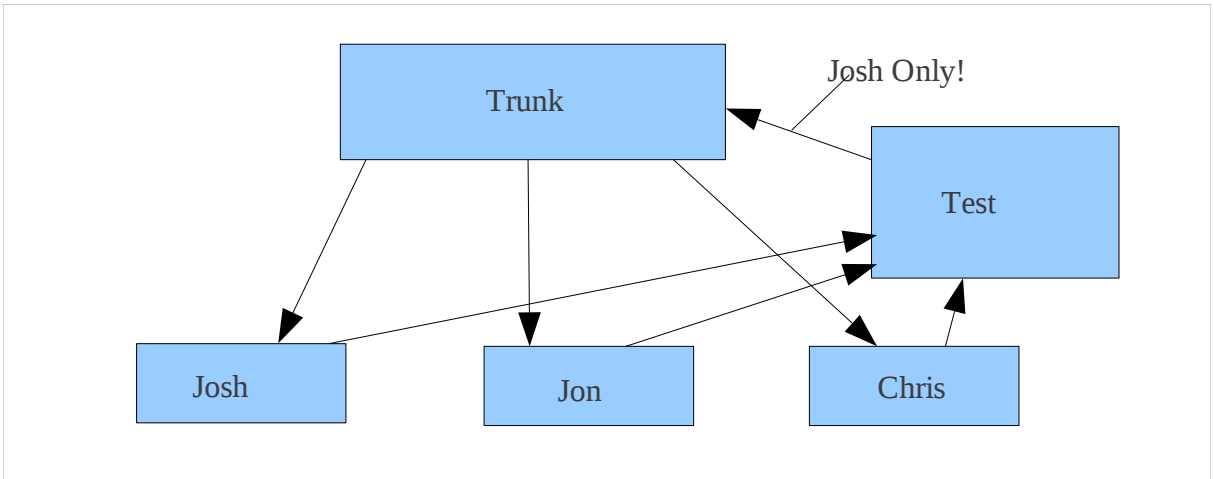
Mit_13 – It is nearly impossible for us to improperly manage our time because we have 12, 14, and 14 hours per week each that we are already at the school, but not in a class. So the bulk of the project will be done during those times when we are already able to work together.

Mit_14 – If we have C#/Visual Studios problems, we will either ask the professor for help getting through the problems, or we will try writing it in a text editor and see if that fixes it. Then get help as to why it works in the text editor while not working in VS.

Mit_15 – If the ASML physically breaks, we will use another teams ASML.

Code Repository / Project Management

URL of our GitHub repository: https://github.com/glenn218k/ASML



Trunk is our main branch, and is only allowed to be touched when the code is fully functional and contains 0 bugs, and when Josh approves of the merging. Trunk is able to be pulled from, but only pushed to /merged with Test.

Test is where we will put our newly modified source codes to test them with the code from trunk to see if the new code is trunk worthy. Test is able to be pushed to/merged with but is the only branch able to push to/merge with Trunk.

We will each have our own branches that we can work freely on, and are able to push to/merge with Test when we have communicated that we are going to do so, as to not push/merge at the same times. We are able to pull from Trunk, but not push to/merge with Trunk.

Time Boxing / Product Grooming

Initial Planning – Jon will be assigning us what we need to get done for each sprint that will last < 2 weeks.

Recalibration – We will meet Monday and Wednesday to see if we need to alter any plans or if we need to fix something or if we run into unforeseen problems.

Next Time box planning – We will use this time to set up our next sprint by prioritizing what tasks and risks we will be focusing on, as well as reviewing current sprints progress.

Acceptance – Chris will evaluate our tasks to see if they meet the acceptance criteria and if they have we will merge the code and then get working on the next sprint.

Process Definitions

Test Driven Development (TDD) – TDD is a programming process that helps write effective code from the start. At the beginning, you write a test case that fails. It is then that you write code to make the test pass, thus being sufficient for completing the task that the code was intended to be used for.

Extreme Programming (XP) – XP is a programming process that has one person typing the code while one or more persons is watching over the shoulder of the coder to ensure that the code is at the best quality from the beginning. By doing this, you cut back on typos as well as not thought out methods and classes.

When we are working on smaller, individual pieces of the code, we will be using TDD, but not XP. This is because we feel that it is more effective to have two people coding at the same time for basic level codes. When we are working on larger, combining of smaller pieces of code, we will use TDD and XP. This is because having just one person attempt this should be much more difficult because they won't know every piece of code that they are combining. Using XP in this situation should help catch a lot of quick mistakes.

Process of Release

To prepare our project for submission, we will go through a series of steps to make sure that the right files are getting turned in. First off, we will make sure all external resources needed are referenced from inside our lib folder. This will make sure that all the files needed to build the solution are all in one place. The next step is to make sure the bin and obj folders are not included in the project. Files that we need to make sure we include are the modeling project, the software management, any write up the specific project requires, and the git hub information. Once all these files are in their correct place, Josh will zip up the submission folder and send it to Chris and Jon. We will both download the zip, and try to launch the software. If it builds we will check and make sure all the files that are suppose to be there are there. If this is all right, we will tell Josh and he will send the exact same zip file to the professor as the submission.