**CS673F17 Software Engineering** 

**Team 4 - Project Name**

**Project Proposal and Planning**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Dan Budris | Team Lead, Configuration Leader | *DB* | 9/20/2017 |
| Eman Zayer | QA Leader | *EHZ* | 9/23/2017 |
| Jash Bhankharia | Implementation Leader, Environment and integration Leader | *JNB* | 9/24/2017 |
| Mithil Shah | Configuration Leader | *MS* | 9/24/2017 |
| Asli Terzioglu | Requirements Leader | *AT* | 9/24/2017 |
| Oindrilla Chatterjee | Design Leader | *OC* | 9/24/2017 |
| George Goddard | Backup Project Leader, Environment and Integration Leader | *GFG* | 9/24/2017 |
| Yergali Askarbayev | Implementation Leader | *YA* | 9/24/2017 |
|  |  |  |  |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| **1.0** | Dan Budris | 9/20/2017 | 9/24/2017 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Overview](#_87t9hln2vjz0)

[Related Work](#_mps353x5ezyl)

[Detailed Description](#_fg3z0hpd4q9v)

[Management Plan](#_ds8oyr75pnh1)

[Process Model](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.27177f40uci)

[Risk Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.a4oqwntk3mw)

[Monitoring and Controlling Mechanism](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.ywdoc2clc9yt)

[Schedule and deadline](#_tadq5mb0pici)

[Quality Assurance Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.72e1f4uawy2r)

[Metrics](#_b2haznn3yyz2)

[Standard](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.vc72k6dweldv)

[Inspection/Review Process](#_f1c69ifi68h7)

[Testing](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.r5d5mhtlf0kq)

[Defect Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.54a4wuncjg1c)

[Process improvement process](#_jhct37ebxxpn)

[Configuration Management Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.hw41vg4ykxen)

[Configuration items and tools](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.bwlb4d4vdox2)

[code commit guidelines](#_yyauft6zr9hw)

[References](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.8mva2050iy7t)

[Glossary](#_ty3i2nqffhtc)

# Overview

Distributed software teams are becoming the norm. Developers, project managers, and their stakeholders work together on complex systems from across timezones. There are numerous pitfalls when working in a distributed team, especially effective communication among individual team members. While a chat application aids in this communication, face-to-face communication is necessary for effective collaboration.

Toward that goal, we will develop a peer-to-peer video and file sharing feature to integrate into the communication feature of the existing application. This feature will provide an integrated button to open a video chat window, and an option to share files as a direct message to another user.

# Related Work

Slack, Hipchat, Skype, Google Hangouts

The major standing out difference is that Talkon is clean integrated audio/video chat application into a process management tool unlike Google Hangout or Skype which are standalone in their own nature. It is a browser based application aimed at achieving high definition quality unlike skype.

# Proposed High level Requirements

* 1. Functional Requirements
     1. Essential Features

1. Individual video chat between users embedded into the existing chat application
   1. Browser embedded audio and video
   2. Multi-platform, multi-browser support
   3. Clean integration with existing chat application
   4. Switch between video and screen sharing
      1. Desirable Features
         1. File sharing option between individual users
            1. Individual user file storage
            2. Shared file storage
            3. Integration with Google Drive
            4. Support for file transfer interruption
         2. Video Chat:
            1. HD video
      2. Optional Features
         1. Conference line for video and audio chat
   5. Nonfunctional Requirements
      1. Secured, with support for SSL
      2. Clear and concise training document with detailed steps for using the feature
      3. Training video for use of the feature
      4. Low lag in video and audio
   6. Implemented Features (to be completed at the end of project)

# Management Plan

# (For more detail, please refer to SPMP document for encounter example)

## Process Model

Agile

Process:

An iterative and incremental development approach will be used. The project will be spread across 4 iterations and every iteration will have a pre defined task before the iteration starts. There will be two meetings every week in order to discuss the ongoings of the project. Every iteration should implement a fully functional and tested feature.

## Objectives and Priorities

Listing down the objectives of the project and every objective is prioritized based on the iteration it is targeted on:

1. Iteration 0: Come up with the essential and desirable features along with updating the project documents. Set up the Github version control and docker environment on virtual machine for every member of the team.
2. Iteration 1: Understanding previously written code would require writing a bunch of test cases to get a feel of the application.
3. Iteration 2: Create a peer to peer video chat application on top of the already existing communication tool along with running test cases.
4. Iteration 3: Provide a file sharing option and integrate it with cloud services like iCloud and/or Google drive.

## Risk Management (need update constantly)

1 - Lowest, 5 - Highest

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Risk ID | Category | Risk | Likelihood | Impact | Action |
| 1 | Resources | Drop of team members | 1 | 4 |  |
| 2 | Learning | Inadequate experience for the development technology | 3 | 3 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Monitoring and Controlling Mechanism

We will be using Slack, Pivotal Tracker and Github to keep track of the ongoings of the project. The Configuration leader will be responsible to review code and merge the pull requests. The Requirements leader will be taking care of pivotal tracker jobs. Slack will be used the primary communication tool between team mates.

Couple of meetings should happen every week. One meeting would be happening post lecture every Wednesday at 8.30pm. The second meeting would be every Sunday 10am in one of these two locations:

1. <https://www.google.com/maps/place/Boston+University+Photonics+Center/@42.3492397,-71.1088264,17z/data=!3m1!4b1!4m5!3m4!1s0x89e379f05b90cbc3:0xc3d04462b5cb0870!8m2!3d42.3492358!4d-71.1066377> - Ground floor.
2. PAL student lounge - MUGAR library 3rd floor.

## Schedule and deadlines (need update constantly)

* + 1. September 27: Iteration 0 Presentation Due
    2. October 18: Iteration 1 Presentation Due
    3. November 8: Iteration 2 Presentation Due
    4. December 6: Final Presentation Due

# Quality Assurance Plan

# (For more detail, please refer to SQAP document for encounter example)

Quality assurance will be applied throughout the entire software development life cycles starting from the analysis of the requirements, software product release until the project closure

For each phase of the software development, the following tasks should be accomplished:

* **Requirements phase:** the proposed functionality of the product should be explicated and clarified. Yet, this will be continuously refined until the requirements are clearly stated and all doubts are closed.
* **Specification and Design phase:** weekly review will be conducted to assure that the defects are identified and rectified
* **Implementation phase:** the code should be reviewed contentiously after completing each single construct
* **Software testing phase:** functional and technical testing will to be conducted prior to the actual customer testing with all issues to be resolved before the final release. But, As we progress in identifying the exact functionalities, the precise number of test cases will be defined and quality measures will be updated accordingly

## Metrics

* + 1. Definition (e.g. define what metrics will be used, , how to keep track of metrics, and how to analyze the metrics for process improvement. Two types of metrics should be included: product metrics and process metrics. Particularly include product complexity (LOC, # of files, # of classes, # of methods etc.) cost (in terms of man hours), defect and defect fix rate etc.

Product Metrics:

Number of lines of new lines of code:

Number of classes:

Number of methods:

Process Metrics:

Cost (person hours per iteration & total)

Defects

Defect completion rate

* + 1. Results (to be completed at the end of each iteration)

## Standard

* + 1. Documentation
       1. All classes, functions and methods should be documented in the code, above the class/function/method signature. This documentation should include a high-level description of what the class/function/method does, and inputs and outputs including expected types.
       2. Code should be self-documenting. If the functionality is obvious through the use of naming conventions then additional documentation is not necessary. If it is unclear what a particular segment of code does or is used for, it should be documented.
    2. Coding Standard
       1. Python (extracted from [Google Python Style Guide](https://google.github.io/styleguide/pyguide.html)):
          1. Avoid global variables.
          2. Do not terminate your lines with semicolons and do not use semicolons to put two commands on the same line
          3. Maximum line length is 80 characters. Do not use backslash line continuation. Make use of Python’s implicit line joining inside parentheses, brackets and braces. If necessary you can add an extra pair of parentheses around a statement.
          4. Use parentheses sparingly. Do not use them in return statements or conditional statements unless using parentheses for implied line continuation. It is fine to use parentheses around tuples.
          5. Indent your code blocks with 4 spaces (no tabs).
          6. Two blank lines between top-level definitions, be they function or class definitions. One blank line between method definitions and between the class line and the first method. Use single blank lines within a function or method as judged appropriately to make groupings of statements easier to read.
          7. No whitespace inside parentheses, brackets or braces. No whitespace before a comma, semicolon, or colon. Do use whitespace after a comma, semicolon, or colon except at the end of a line. No whitespace before the open paren/bracket that starts an argument list, indexing or slicing. Surround binary operators with a single space on either side for assignment, comparisons, Booleans, and arithmetic. Do not use spaces around the ‘=’ when used to indicate a keyword argument or a default parameter value. Do not use spaces to vertically align tokens on consecutive lines.
          8. Generally only one statement per line.
          9. Naming: module\_name, package\_name, ClassName, method\_name, ExceptionName, function\_name, GLOBAL\_CONSTANT\_NAME, global\_var\_name, instance\_var\_name, function\_parameter\_name, local\_var\_name.
          10. Imports should be on separate lines.
          11. Even a file meant to be used as a script should be importable and a mere import should not have the side effect of executing the script’s main functionality. The main functionality should be in a main() function.
       2. JavaScript (extracted from [Google JavaScript Style Guide](https://google.github.io/styleguide/jsguide.html)):
          1. Braces are used for all control structures even if only a single statement. The first statement of a non-empty block must begin on its own line.
          2. Non-empty blocks use Kernighan and Ritchie (K&R) style:

No line break before the opening brace.

Line break after the opening brace.

Line break before the closing brace.

Line break after the closing brace if that brace terminates a statement or the body of a function or class statement, or a class method. Specifically, there is no line break after the brace if it is followed by else, catch, while, or a comma, semicolon, or right parenthesis.

* + - * 1. An empty block or block-like construct may be closed immediately after it is opened, with no characters, space, or line break in between (e.g. {}), unless it is a part of a multi-block statement (one that directly contains multiple blocks: if/else or try/catch/finally).
        2. Each time a new block or block-like construct is opened, the indent increases by 2 spaces. When the block ends, the indent returns to the previous indent level. The indent level applies to both code and comments throughout the block.
        3. Any array literal may be optionally formatted as if it were a block-like construct.
        4. Any object literal may optionally be formatted as if it were a block-like construct.
        5. Class literals (whether declarations or expressions) are indented as blocks. Do not add semicolons after methods, or after the closing brace of a class declaration (statements -- such as assignments -- that contain class expressions are still terminated with a semicolon).
        6. When declaring an anonymous function in the list of arguments for a function call, the body of the function is indented 2 spaces more than the preceding indentation depth.
        7. One statement per line followed by a line break and terminated with a semicolon. Do not rely on automatic semicolon insertion.
        8. Column limit is 80 characters.
        9. Line-wrapping:

Prefer to break at a higher syntactic level.

When a line is broken at an operator the break comes after the symbol. This does not apply to the “dot” (.), which is not actually an operator.

A method or constructor name stays attached to the open parenthesis that follows it.

A comma stays attached to the token that precedes it.

When line-wrapping, each line after the first (each continuation line) is indented at least 4 spaces from the original line, unless it falls under the rules of block indentation.

* + - * 1. Vertical whitespace. A single blank line appears:

Between consecutive methods in a class or object literal.

Exception: A blank line between two consecutive properties definitions in an object literal (with no other code between them) is optional. Such blank lines are used as needed to create logical groupings of fields.

Within method bodies, sparingly to create logical groupings of statements. Blank lines at the start or end of a function body are not allowed.

* + - * 1. Horizontal whitespace: no whitespace at the end of a line. A single space can appear in the following places:

Separating any reserved word from an open parenthesis that follows it on a line.

Separating any reserved word from a closing curly brace that precedes it on a line.

Before any open curly brace with two exceptions:

Before an object literal that is the first argument of a function or the first element in an array literal.

In a template expansion, as it is forbidden by the language.

On both sides of any binary or ternary operator.

After a comma, semicolon. Spaces are never allowed before these characters.

After the colon in an object literal.

On both sides of the double slash that begins an end-of-line comment.

After an open-JSDoc comment character and on both sides of close characters.

* + - * 1. Vertical token alignment: do not use spaces to vertically align tokens on consecutive lines.
        2. Do not use line continuations for string literals. Use concatenation instead.
        3. It is rarely correct to do nothing in response to a caught exception. When it truly is appropriate to take no action whatsoever in a catch block, the reason this is justified is explained in a comment.
        4. Switch-case:

Fall-throughs are always commented.

Default case is present even if it contains no code.

* + - * 1. Naming:

Package names: lowerCamelCase

Class names: UpperCamelCase. Typically nouns or noun phrases.

Method names: lowerCamelCase. Typically verbs or verb phrases

Enum names: UpperCamelCase. Individual items within the enum are named in CONSTANT\_CASE

Constant names: CONSTANT\_CASE

Non-constant field names: lowerCamelCase. Typically nouns or noun phrases.

Parameter names: lowerCamelCase. One-character parameter names should not be used in methods.

Local variable names: lowerCamelCase

Template parameter names: concise single-word or single-letter identifiers in all-caps, such as TYPE or THIS or T.

## Inspection/Review Process

## Through all these phases of the software development, the Communication and Feedback process will always be considered to improve the software quality based on the weekly progress of the software development; disagreements, suggestions and opinions about all aspects should be evaluated along with the auditions and evaluations at the end of each phase in the project

## Testing

functional and technical testing will to be conducted prior to the final customer testing with all issues to be resolved before the final release. But, as we progress in identifying the exact functionalities, the precise number of test cases will be defined and quality measures will be updated accordingly.

## Defect Management

As we proceed in testing, all reported defects will be assessed, prioritised and resolved accordingly. The following criteria will be applied:

1. Defect Prevention:Identify and analyze the causes of defect towards reducing its occurrence
2. Baseline delivery: the defect will be assessed based on the base predefined milestone
3. Defect Discovery: A defect is discovered when it is raised to the developers attention and acknowledged as a defect based on the milestone Baseline delivery.
4. Defect Resolution and Process Improvement.a resolution process will be immediately.

# Configuration Management Plan

(For more detail, please refer to SCMP document for encounter example)

## Configuration items and tools

## Github <https://github.com/danbudris/BU_MET_CS673>

**Docker**

As we build out the application, we will need to be able to deploy the newest version in a consistent and testable way. Toward this end each iteration will include a packaged Docker container with the newest version of the application for display and testing.

## Change management and branch management

Code Commit Process for a new feature:

Create a feature branch off of development

Create a personal branch off of the feature

Develop of of your branch, and merge to feature

Merge of feature to development requires pull request and approval

Merge to master requires pull request and approval

* + 1. We will be using two main branches, a master branch and a development branch. The development branch will be used for all active development. Feature branches will be branched from the development branch. The master branch will only be used for ‘production’ deployments and deliverables, and will require a pull request in order to merge.

## Master (requires approved pull request; deliverables cut from Master)

## Development (working branch to which feature branches are merged

## Feature Branches (work on a specific feature)

* + Bugfix Branches (work on solving a specific bug)

## Code commit guidelines

## Test before commit; no commit should be implemented untested. Clean commented code with clear documentation within the code area should be done. Stray/unused code should not be committed.

## Always a commit should be done only after a parent branch refresh i.e. “pull”. Each merge should follow the naming convention “feature/Feature\_Name”, “buxfix/Bug\_Information”.

## Each commit should include a detailed comment that explains the content of the commit and the reasoning behind it.

## Each pull request raised for a merge should lead to an Auto-Merge without any conflicts.

## Each merge should be reviewed and approved by another team member.

## No commits or stashing changes on the master branch should be done.

## Each merge should contain a detailed merge message referencing the reasoning behind the merge and the purpose.

# References

(For more detail, please refer to encounter example in the book or the software version of the documents posted on blackboard. )

1. How to build a Web RTC chat application: http://web-engineering.info/node/57

# Glossary