Software Life Cycle Documents

for

iText Based PDF Viewer

Version 1.0

Prepared by

Dhruvkumar Patel

Jace Robinson

Pranav Pranav

CS 7140 Advanced Software Engineering

June 27th 2016

Table of Contents

1. Introduction

This is a collection of the software lifecycle documents for the *iText Based PDF Viewer*. This report contains the requirements, specification, design, implementation, and testing documents.

1.1 Purpose

The purpose of this document is to completely describe and define the whole lifecycle of development for the *iText Based PDF Viewer* product. This application will allow users to view and annotate files following the Portable Document Format (PDF) version 1.7 [4]. The viewer will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

iText Based PDF Viewer will be developed using the free and open source library iText [1]. This extensive library can assist in manipulating the PDF internals while using the Java programming language. For display pdf into javafx window we used PDF-Renderer [6] library. As a classroom project, the requirements, specification, design, implementation, and testing documents will all be created for this product.

1.2 Intended Audience and Reading Suggestions

There are several types of readers who may view this document. Below are various categories of users along with suggested sections to read.

Developers:
A developer will be interested in learning basically all documents necessary to create the product. Therefore this group should be reviewing the entire document. As shown in the table of contents, specific sections will refer to the specific document (for example design document) that the developer may need to reference.

Project Manager:

A project manager will want to understand a bigger picture view of the product in order to manage the work. He or she is encouraged to read all of section 2, 3, and 5. This will give a big picture view of the product, users and constraints. To learn more details the manager can briefly review section 6 and 7.

Users:
Users of the product may read this document for a number of reason such as wanting to know some motivation for the product as discussed in section 2, or a list of functionality in section 3. The user can also read section 6 to understand some of the implementation decisions.

Students:
Students may read this document as an example of requirements document. For educational purposes, it is suggested the student reads the entire document to gain the most understanding.

1.3 Product Scope

The goal of this project is to develop a Java and iText based PDF viewer. The viewer should be able to open files following the PDF format. The user should also be able to annotate the PDF, and save these changes with the file. For more detailed description of functionality see sections 2.2 and 3.

As this is an educational project, the scope will be severely limited when compared to a professional product. The primary focus of the project is to develop the documentation for a software life cycle of requirements, specification, design, implementation, and testing. The product will not allow replacement of text at the level of lines, words, and paragraphs. The viewer is only expected to display textual content. This means the product will not be able to display vector graphics, raster images, or any other types of content commonly stored in PDF. The product will not support hand drawn annotations such as created by a stylus pen on a tablet device.

The product must be built using the iText library to handle transfer of information between PDF and this viewer. iText was chosen due to size, popularity, and quality of user documentation. Some of the other choices considered were jPod, PDFBox, and ICEpdf. An investigation comparing and contrasting the various PDF libraries was not performed and the choice of iText should not imply higher quality.

1.4 References

[1] "iText Developers", *Developers.itextPDF.com*, 2016. [Online]. Available: http://developers.itextPDF.com/. [Accessed: 15- Jun- 2016].

[2] "Tracker Software Products :: PDF-XChange Editor", *Tracker-software.com*, 2016. [Online]. Available: https://www.tracker-software.com/product/PDF-xchange-editor. [Accessed:15- Jun- 2016].

[3] "XODO PDF Reader & Annotator", *Xodo.com*, 2016. [Online]. Available: http://xodo.com/. [Accessed: 15- Jun- 2016].

[4] *PDF Reference*, 6th ed. Adobe Systems Incorporated, 2006.

[5] "Doxygen: Main Page", *Stack.nl*, 2016. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/. [Accessed: 16- Jun- 2016].

[6] “PDF-Renderer”, *Java.net - The Source for Java Technology Collaboration.* [Online]. Available: https://java.net/projects/pdf-renderer.

1.5 Document Terminology

Throughout the document the following words defined below should be taken as precise and accurate definitions.

“PDF File” - For the purposes of this Requirements Document, “PDF file” refers to a file format in compliance with the standards defined in the PDF reference document version 1.7 [4].

"text" - For the purposes of this Requirements Document, "text" refers to "Text Objects" described by Section 5.3 of the PDF reference document [4] with the added restriction that ”text" is composed solely of sequences of Latin characters irrespective of font specified in the PDF for display of the characters. Latin characters are defined in Appendix D.1 of the PDF reference [4].

"annotation" - refers to a subset of annotations described in section 8.4 of the PDF reference document [4]. The subset chosen are underline, highlight, strikethrough, rectangular box, and comment. Table 8.28 of the PDF reference [4] describes rectangular box, and table 8.30 of the PDF reference [4] describe underline, highlight and strikethrough, and table 8.23 describes comment. A visual example of each annotation is given in figure 3.2.1 and figure 3.2.2 of this document.

“upload” - refers to the display of only text and annotations encoded within PDF documents selected from within a file system by a user. The text will display to the output monitor device of the computer.

“save” - refers to storing PDF documents with user selected file names at user selected file system locations.

“insert” - “insert” refers to adding content to the current state of the PDF document.

“delete” - refers to removing content from the current state of the PDF document.

“location” - refers to the position in the grid defined in section 4.2 of the PDF reference document [4].

“search” - refers to matching input text with all occurrences of text contained in the current page of document such that textInput = textDocument. The locations of the matching text within the document are determined.

“PDF page” - refers to a grouping of PDF content as defined in section 3.6 of PDF reference document [4].

1.6 Definitions, Acronyms, and Abbreviations

GUI	Graphical User Interface
JAR	Java Archive
JDK	Java Development Kit
PDF	Portable Document Format
UI	User Interface

UML	Unified Modeling Language

1.7 Figures

2. Overall Description

2.1 Product Perspective

As discussed, in section 1.3, the goal of this project is to develop a Java and iText based PDF viewer where the user should also be able to annotate the PDF, and save these changes with the file. And as discussed, in section 1.1, the project will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

As a class project, will be limited to a proof of concept. It will be written in Java 8 and JavaFX will be used for its front end. Exploiting the iText library, the ability of extracting texts and adding annotations will be required. See sections 2.2 and 3 for more details.

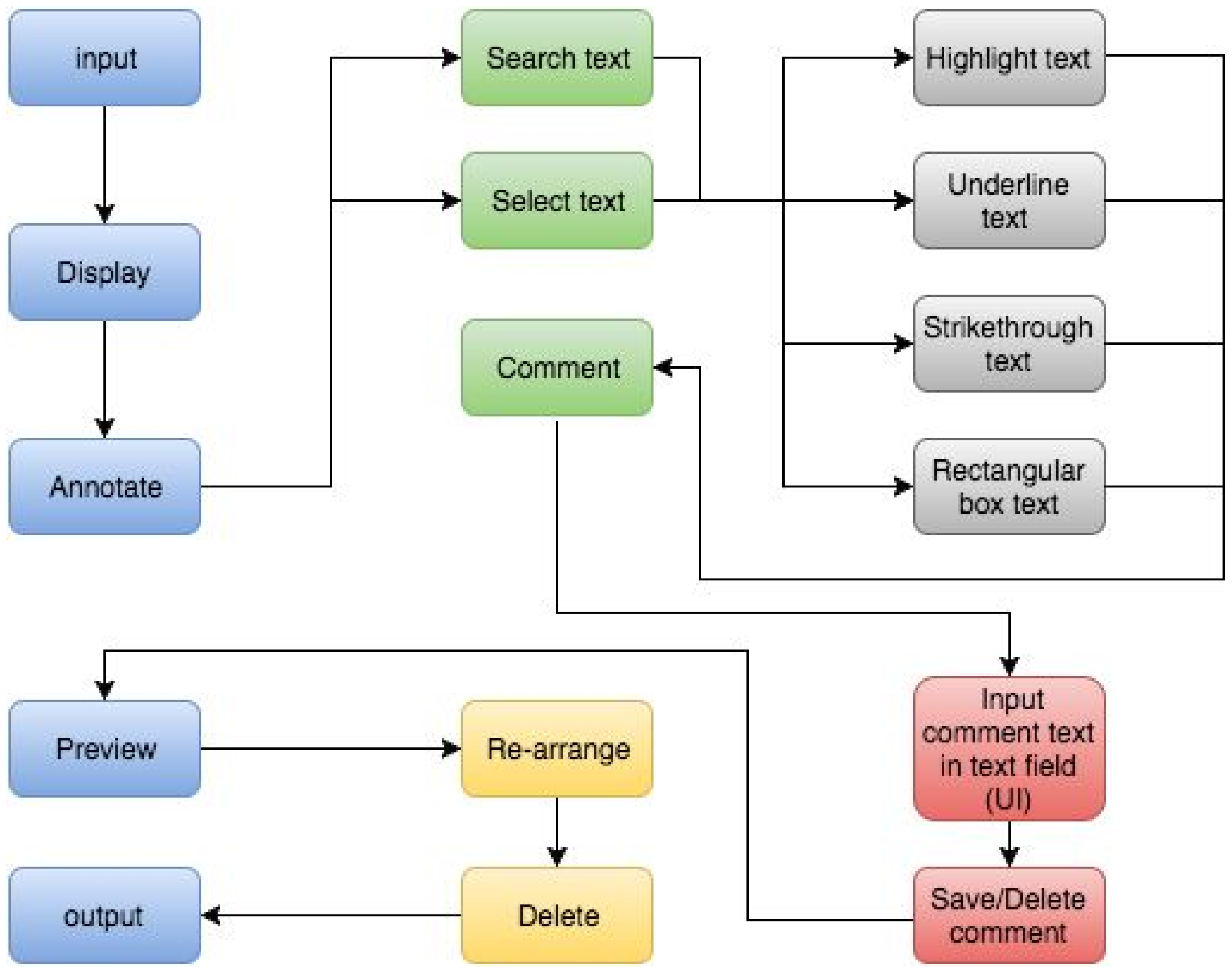


Figure 2.1.1: Work flow of the product.

As seen in figure 2.2.1, all blue color boxes are the basic functionalities of the product. Input, display, annotation, preview and output are discussed in more detail in following sections [2.2]. All green color boxes are sub functionality of the “Annotation”. All grey color boxes are different options for “Select text”. All red color boxes are sub functionality of the “Comment”. All yellow color boxes are sub functionality of the “Preview”.

2.2 Product Functions

Following are the functionalities given by the product

1. Input: User Interface (UI) which accepts input as uploaded file defined by PDF [4].

2. Display: UI to display the uploaded PDF in a way that content of the given PDF are not editable at the level of lines, words, or paragraphs, but to be viewed as same. Further the display will be used to annotate text as well.
3. Annotate: UI in the display will be used to annotate text in a number of ways and operations performed given below.
    a. Select text: There are four ways to select text.
        i. Text Highlighting: Mouse cursor will be provided to select and highlight the text
        ii. Under lining: Mouse cursor will be provided to select and underline the text
        iii. Strikethrough: Mouse cursor will be provided to select and strikethrough the text
        iv. Rectangular Box: Mouse cursor will be provided to select and box the text
    b. Comment: operations can be done as part of the annotation to the selected text.
        i. Open text field for keyboard input which can be save or delete. No edit strictly. UI should have fresh look and feel.
        ii. Save or delete the comment.
    c. Search text: operations will be created to allow user to search for specific text and display the location of matching text within the document.
4. Preview: An option provided to user to do below operations.
    a. Rearrange: the user will able to re-arrange all the pages of the given PDF pages.
    b. Delete: the user will able to delete the pages he/she wish to. Comments will be also deleted with the deleted pages if any.
5. Output: Save the project including the given preview page(s) and associated comment into a PDF [4] file as output.

2.3 User Classes and Characteristics

There are two main groups of users expected to use the product.

Professional PDF User:
A professional PDF user is interested in most to all of the product’s functions. Users from this group are likely to work with PDF’s very frequently, and make use of the ability to add and

remove annotations, and search for specific text locations. This is the most important audience for the product.

Novice PDF User:
A novice PDF user is only interested in a subset of the products function. Specifically, this user group will only be interested in viewing PDFs, and will not add or remove annotations or search for text. This is a lesser important audience for the product.

2.4 Operating Environment

The application will be developed to work on up-to-date software at the time of the creation of this document. As the software will be built using Java and delivered as a .jar file, to run the software a user must have the Java Runtime Environment (JRE) version 1.8.0_91 or higher installed. The application will be usable with Windows 10 version 1511, and Ubuntu version 14.04.

2.5 Design and Implementation Constraints

There are several constraints affecting the development of this product. The constraints are listed below.

1. The product must be developed in six weeks. Due to this short time frame, the scope has been severely reduced.
2. The software development life cycle will follow a waterfall model. This model has the sequence of deliverables of requirements, specifications, design, implementation, and testing.

2.6 User Documentation

The product will be delivered with two manuals.

1. The first manual will describe the source code. This is intended for developers and not for users. The document will be created using *Doxygen* to describe the source code and display class diagrams [5].
2. A second document will be created to describe all of the features available to the user. This non-technical document will provide written tutorials on how to open a PDF, search for text, create annotations, and save PDF.

2.7 Assumptions and Dependencies

There are a few critical dependencies for this product that are listed below.

1. The viewer is developed to display PDF version 1.7 formatted files [4]. It is assumed that future versions of PDF will remain backwards compatible with this version.
2. The viewer is developed using free and open source library developed by iText. It is assumed the library remains free in the future.
3. The viewer is to be developed using JavaFX available in Java 8. It is assumed current and future JRE's past version 1.8.0_91 will be backwards compatible and execute the deliverable .jar file.
4. As a test to the saving and uploading of PDF documents, the PDF viewer XODO [3] will be used. A requirement defined in section 3 is a PDF file created in this product must be viewable in XODO. It is assumed that XODO will continue to be available during the development process. It is also assumed XODO will continue to follow PDF standards.

3. Software Requirements

Within this section contains a numbered list of all requirements of the product. Numerous definitions of words used in the requirements are given in section 1.5, document terminology.

Following are the numbered requirements for iText Based PDF Viewer.

3.1 External Interfaces

Following are the User Interface requirements for this PDF Viewer.

3101: Facility to upload an existing PDF file.
3102: Display both text and annotation content of a single page of PDF file into an output device such as monitor.
3103: Facility to change current content of display from one page of PDF file to a different page.
3104: Facility to search text.
3105: Facility to save PDF under current filename.
3106: Facility to insert an annotation.
3107: Facility to delete an annotation.
3108: Facility to delete page including the annotations on the page.
3109: Facility for highlighting annotation
3110: Facility for underlining annotation
3111: Facility for strikethrough annotation
3112: Facility for creating rectangular box annotation
3113: Facility for comment annotation

Following are the Software Interface required for the PDF Viewer.

3114: The product shall use iText 7 library to read PDF file.
3115: The product shall use iText 7 library to save PDF file.

3.2 Functional Requirements

Following are the major functionalities of iText Based PDF Viewer.

3201: The product shall upload an existing PDF file
3202: The product shall search particular text
3203: The product shall display location(s) of searched text
3204: The product shall do following on annotations:
- 3204.1: Insert an annotation
- 3204.2: Delete an annotation
- 3204.3: Multiple annotations can appear on single PDF page

3204.4: Different types of text selection annotations are available as in figure 3.2.1:

3204.4.1: Text Highlighting in color yellow

3204.4.2: Underlining in color blue

3204.4.3: Strike-through in color red

3204.4.4: Rectangular Box in color red

3204.5: Comment annotation as in figure 3.2.2

3204.5.1: Have insert text in the comment annotation

3204.5.2: The text of comment annotation can be deleted.

3205: The product shall allow rearranging order of pages of the PDF file

3206: The product shall allow deletion of page(s) of PDF file

3207: If uploaded file PDF file, then the saved file must must be PDF file

3208: Annotations inserted and saved in this product must be viewable in PDF viewer XODO

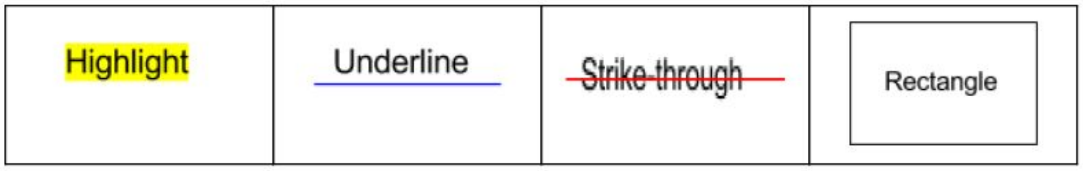
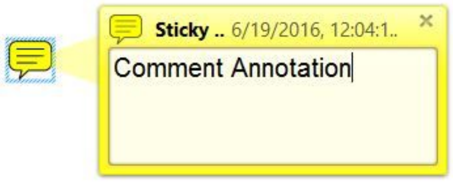


Figure 3.2.1

In this figure, we can see the four types of annotations from left to right, highlight annotation, underline annotation, strike-through annotation, and box annotation



Comment Annotation

Figure 3.2.2

In this figure, we can see an example of a comment annotation. A comment consists of a modifiable text box pointing at a specific location in the PDF document.

3.3 Performance

Following are the performance requirements for iText Based PDF Viewer.

3301: 95% of the time the product must display text content of compliant PDF file within 10 seconds of upload.
3302: 95% of the time the product must display text and annotation content of compliant PDF file within 15 seconds of upload.
3303: 95% of the time the product must display inserted annotation within 5 seconds to output monitor.

3.4 Software System Attributes

In order to guarantee a usable product for customers, the following attribute requirements are listed.

Reliability:
3401: 99% of the time the product shall not crash or hang after one hour of continuous usage.
3402: 99% of the time the product shall not crash when uploading a non-PDF compliant document

Portability:
3403: The product shall satisfy all requirements of this document in both Windows 10 and Ubuntu with versions discussed in section 2.4.

3.5 Additional Requirements

There are a few additional requirements imposed upon the developers.

3501: Code maintainability expectations
Well designed software should be maintainable. There are several techniques that will be followed in the creation of this product. First, requirements, specifications, and design will be created before implementation begins. Second the developers will follow the SOLID principles

of object oriented programming. Third the programmers will be following Design by Contract. 80% of methods must have a pre and post condition, along with class invariants and loop invariants.

3502: Testing requirements
As the developers are creating maintainable code through following the principles described before, this code is also becoming much more testable. The design by contract principles will enforce assertions to ensure certain conditions are maintained throughout the development. All requirements listed in this document can be tested as boolean success or failure through acceptance testing. Unit testing on several classes will also be performed.

4. Software Specifications

The iText Based PDF Viewer product can be specified as separate components. The five components are the PDF Document, PDF Input-Output, Display, Toolbar and Cursor and Keyboard. Specifications will reference the numbered requirements from the requirements document in section 3.

4.1 PDF Document

First we will define some of the math objects used throughout the specification. The PDF Document is the primary object upon which everything operates.

Define: PDFDoc[1...n] as array of PDFPage where each PDFDoc[i] is the ith PDFPage of the document, 1 <= i <= n, where n is total number of pages (natural number).

Define: PDFCanvas[1...m] as array of elements of type *Text Element* or type *Annotation Element*, such that PDFCanvas[i] is the ith element of the PDFCanvas of type PDFCanvas[i].type == (Text || Annotation), 0 <= i <= m, where m number of objects in the page.

Define: *Text Element* as an object with three attributes of content, position and type, with notation Text.content, Text.position, or Text.type. The content is a stream of characters where the length is >= 0. This stream of characters completely describes both the “characters” and “glyphs” necessary to display to screen as defined in PDF reference [4]. The position is 2D rectangle represented as four number (i,j,k,l) where i,j>=0 and k,l >0. The four vertices of the rectangle in cartesian format would be (i,j), (i+k,j), (i,j+l), and (i+k,j+l). Text.type == Text.

Define: *Annotation Element* as an object with three attributes of annotType, position, and type. annotType can be one of four values, highlight, underline, strikethrough, or box. These annotTypes completely describe how to display to the screen as defined in PDF reference [4]. The position is 2D rectangle represented as four number (i,j,k,l) where i,j>=0 and k,l >0. The four vertices of the rectangle in cartesian format would be (i,j), (i+k,j), (i,j+l), and (i+k,j+l). Annotation.type == Annotation.

Next we will define some basic operations on PDFDoc and PDFCanvas.

3101: When a page number c is removed, all pages before c are unchanged, and all pages after have their page number reduced by 1. This operation meets requirement 3206.

removePage(PDFDoc, c), where c is a natural number.
Pre: PDFDoc[1...n] not null and 1<= c <= n.
Post: n == n.old-1 and and PDFDoc[1...c-1].old == PDFDoc[1...c-1] and PDFDoc[c...n] == PDFDoc[c+1...n.old].

3102: To add an annotation to canvas of a page, simply add the annotation element to the existing canvas of page array without changing other elements.

addAnnotation(PDFCanvas, AnnotationElement)
Pre: PDFCanvas[1...m] not null and AnnotationElement not null
Post: m == m.old + 1 and PDFPage[1...m-1] == PDFPage[1...m-1].old and PDFPage[m] == AnnotationElement

3103: To remove annotation on canvas of a page, simple remove the existing annotation element from the canvas of a page.

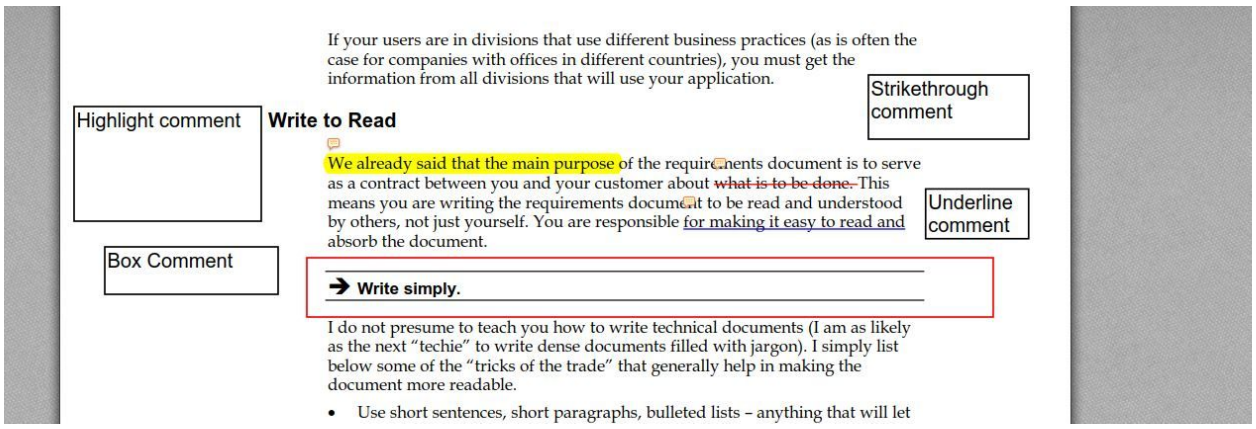
removeAnnotation(PDFCanvas, AnnotationElement)
Pre: PDFCanvas[1...m] not null and AnnotationElement in PDFCanvas
Post: m == m.old - 1 and PDFCanvas[1...m] == PDFCanvas[1...m-1].old and PDFCanvas[1...m] != AnnotationElement and PDFCanvas[1...m.old].old == AnnotationElement

3104: If keyPhrase in PDFCanvas, return position of matching keyphrase.

searchText(PDFCanvas, keyPhrase) where keyPhrase is a sequence of characters
Pre: PDFCanvas not null and keyPhrase not null
Post: ((keyPhrase == substring(PDFCanvas[i].text) and position == PDFCanvas[i].position for some i) || (keyPhrase != substring(PDFCanvas[1...m].text)) && PDFCanvas == PDFCanvas.old

The specifications 3102-3104 meet requirements 3202-3204.

In figure 4.1, we can see a specification by example of the "look and feel" of each type of annotation.



If your users are in divisions that use different business practices (as is often the case for companies with offices in different countries), you must get the information from all divisions that will use your application.

Write to Read

We already said that the main purpose of the requirements document is to serve as a contract between you and your customer about ~~what is to be done.~~ This means you are writing the requirements document to be read and understood by others, not just yourself. You are responsible for making it easy to read and absorb the document.

➔ **Write simply.**

I do not presume to teach you how to write technical documents (I am as likely as the next "techie" to write dense documents filled with jargon). I simply list below some of the "tricks of the trade" that generally help in making the document more readable.

- Use short sentences, short paragraphs, bulleted lists – anything that will let

In Figure 4.1.1 the "look and feel" of the four types of annotations are shown. The *highlight* annotation is shown by the yellow background surrounding the phrase "We are said that the main purpose". The strikethrough annotation is shown by the red line through "what is to be done". The underline annotation is shown in blue on the phrase "for making it easy to read and". The box annotation is shown as the red box surrounding "Write simply.". Lastly for each type of annotation, there is a comment, which contains text provided by the user.

4.2 PDF Input-Output

PDF Input-Output are the components of “uploading” a document that satisfies PDF reference [4] and “saves” a document that satisfies the reference.

Define: *isPDFCompliant(file)* as a boolean test of whether the bytes read from any file satisfy PDF reference [4].

Define: *textEqual(file1,file2)* as a boolean test of whether all the text elements as defined in section 3.1 of this document for file1 exist in file2. Notice that file2 may have more text elements than file1.

3201: isPDFCompliant(input) and isPDFCompliant(output), where input is input file before uploading to software, and output is output file after save operation.

3202: upload(inputFile)
Pre: true
Post: isPDFCompliant(inputFile) || (NOT isPDFCompliant(inputFile) and exit program)

3203: textEqual(output,input). As stated in the requirements, pages of the PDF can be deleted, but not added.

These specifications meet requirements 3201 and 3205-3208.

4.3 Display

The *display* is the finite 2-dimensional grid where the PDFDocument is visible to the user.

Define: grid[1...max_x][1...max_y] as 2-dimensional array such that grid[i,j] contains either a Text element, Annotation element, or is empty. If the grid[i,j] contains a Text element, display the content of the element. If grid[i,j] contains an Annotation element, display the annotType of the element. If the grid[i,j] is blank, display white background.

4.4 Toolbar

The types of components (save button, display field, etc…) are the only detail from the figure to be interpreted as formal specifications. The positioning of the components in the figure are not to be taken as specifications. The visual appeal of the figure is also not the be taken as specification. These specifications meet the requirements 3101-3112.

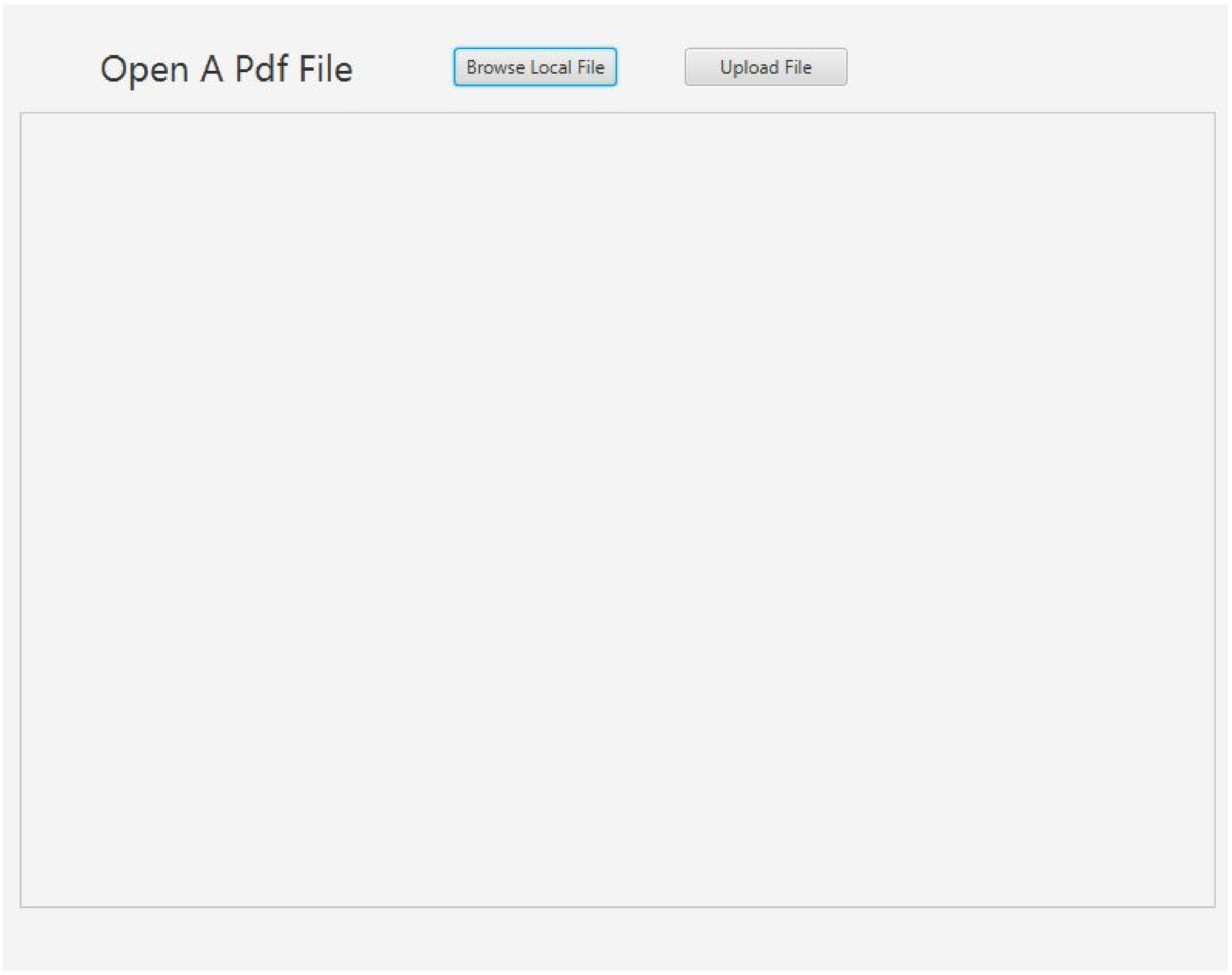


Figure 4.4.1 : screen 1: browse, upload button and display content.

A second screen will contain buttons save, annotate, search, and split functionality button.
Screen 2:- save,annotate, search, split functionality

4.5 Cursor and Keyboard

The user is able to interact with the software through a cursor and keyboard.

The cursor is a device that allows the user to interact with the grid and toolbar components as defined in sections 4.3 and 4.4 of this document. The cursor when interfacing with the display, will be located at a specific (i,j) location corresponding to grid[i,j]. Define an event as an action taken by the user. An event can be used to invoke a response at location (i,j). When interfacing with the toolbar, the cursor will be located at a specific button. When an event is started by the user, if the cursor is located at a button, the button event begins.

The keyboard is a device that allows character input. A sequence of characters can be collected from sequential use of the keyboard.

4.6 Additional Specifications

Requirements 3301-3303, 3401-3403, and 3501-3502 are already unambiguously specified in the requirement document and are therefore not mentioned in this document. Requirement 3503 has been removed now and in all future documents.

5. Software Design

This section describes the issues considered in determining the design.

5.1. Goals and Guidelines

There were three main design goals for this product.

1. Keep it Simple Stupid (KISS)
2. Design by Contract
3. Maximize Use of iText

For the first design goal, a motivation for this project is the development of the software development lifecycle documents. The code is not intended for professional use, and therefore does have extensive development for a variety of use cases.

In the second design goal, we want to emphasize the principles of design by contract. Therefore, the architecture was designed in a way to give precise responsibility for various classes.

The third design goal is to maximize the use of iText software library. Significant effort has been spent to learn and understand the possible usability of iText in the context of this product. Many of the core functions are implemented in iText.

5.2. System Architecture

This section provides information related to system architecture which contains major components and their structures and how they are interact or related with each other. Software architecture provides whole system overview and consider following points as advantages:

1) Describe the structure of the system but hide the implementation details.
2) Includes Data Flow and UML diagram to describes all the components and their interactions with each other.
3) satisfy both functional and quality requirements.
4) In figure 5.2 is the UML Class diagram describes all the classes with attributes and operations.

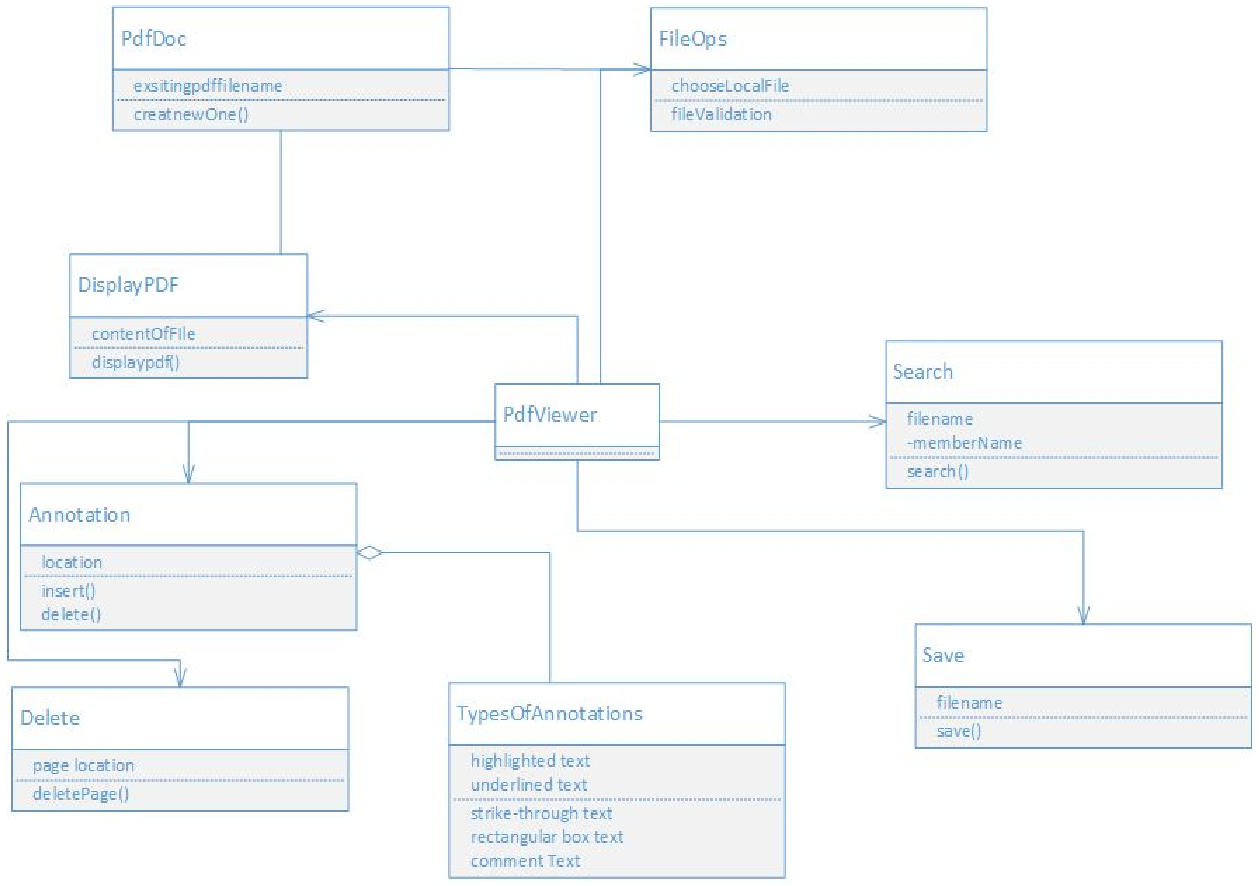


Figure 5.2.1: UML class diagram

Here, in Class diagram TypesOfAnnotations class is a strong part of Annotation class so it describes with aggregation relationship and all the other are connected with direct connection link.

5) Here is the Work Flow Diagram

Work Flow Diagram describes the whole flow of the system. It is start with the initial step and follow each and every step to develop all the features.

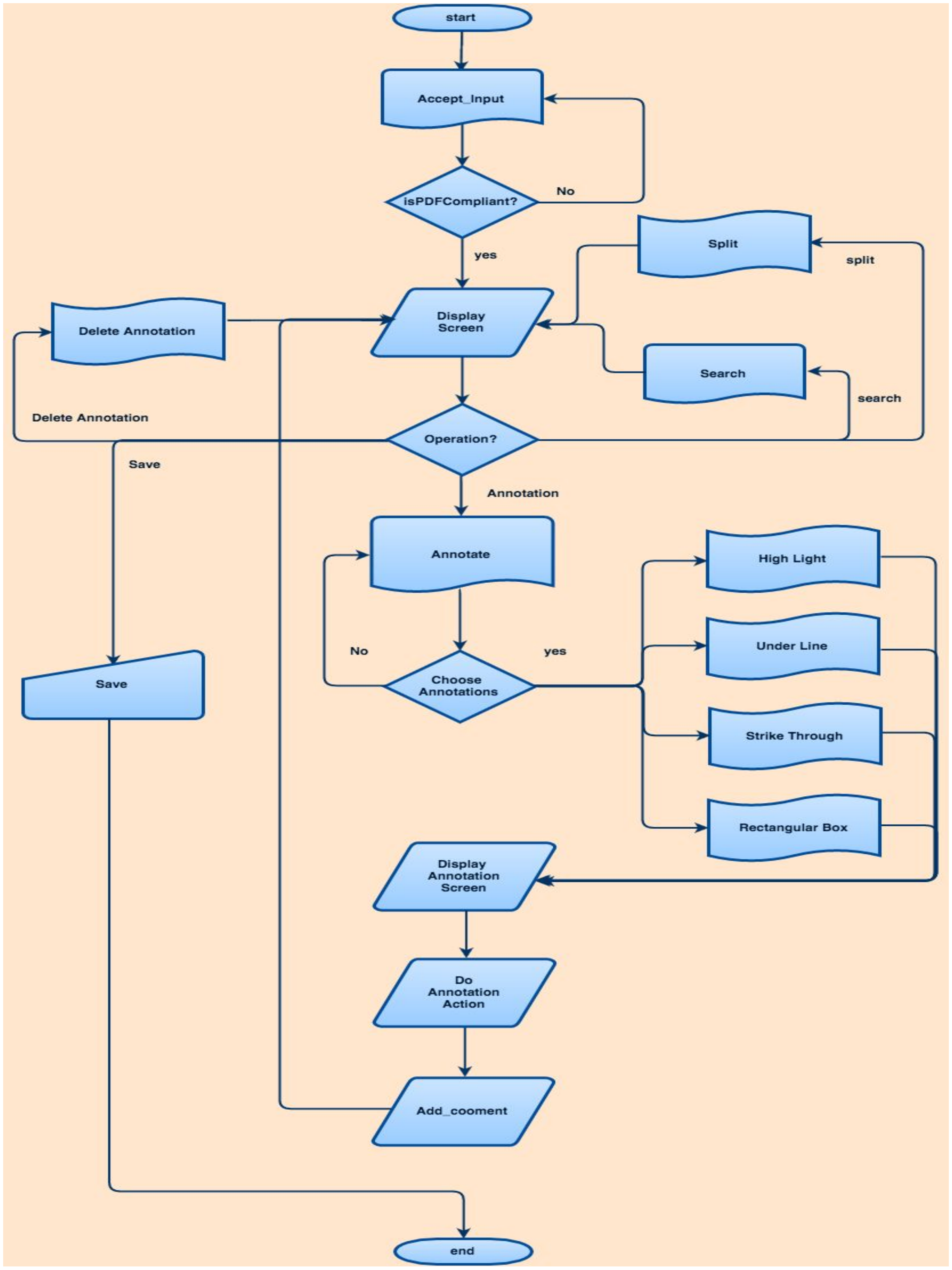


Figure 5.2.2: Work Flow Diagram

Steps of Work Flow Diagram:

1. Start with take pdf file as an input from local machine.
2. Check the file validation it must be a pdf file otherwise exit.
3. Display pdf file in javafx scrollpane screen.
4. Now one can select operations to perform specific things and display the document with particular operation succeeded.
    1) search option to search particular word and it will display with highlighted text
    2) split option to arrange one document by pages
    3) To Insert an Annotation select particular type of annotation among types of annotation.
    4) display pdf on the screen with particular annotation succeeded.
    5) To delete an annotation or delete a page with annotations.
    6) Save option is to save file with existing file name and exit.

5.3. Detailed System Design

Additional details on each class as introduced in section 3 are provided below. The purpose of each class, its class invariants, and pseudo code for each function are given. Given that the design is built around iText, functions that are provided by iText will be referenced as iText.function(args).

5.3.1. PDFViewer

The PDFViewer class is the starting object for this program. The display is started from here.

```
Function run() {
displayPDF()
}
```

5.3.2. PDFDoc

The PDFDoc class maintains the content of the PDF. This content includes all text and annotation elements.

The class invariant is the list of TextElements must not increase size. TextElements can be deleted as entire PDF pages are deleted, but content cannot be edited.

Fields:
List iText.TextElements
List iText.AnnotationElements

Functions:

```
addAnnot(annot) {
        Add Annot to AnnotationElements
}

deleteAnnot(cursor) {
        if(intersect(cursor, iText.AnnotationElements))
                removeAnnotation from AnnotationElements
}
```

5.3.3. DisplayPDF

The DisplayPDF class creates the external interface visible by the user. The user will interact with the interface using a cursor. The cursor can be used to select one of the several GUI buttons of save, search, upload, or a comment area of an annotation. The interaction with a GUI element will trigger the corresponding event.

The class invariant is all elements of PDFDoc must be within bounds of grid.

Fields:
PDFDoc
grid[MAX_x][MAX_y]

```
cursorLocation (x,y)
saveField
searchField
uploadField

Functions:
displayPDF() {
    Initialize GUI elements
}

displayPDF(PDFDoc) {
    For all elements in PDFDoc
        Grid[element.location.x][element.location.y] = element.content
}

moveCursor(cursorLoc) {
    cursorLocation = cursorLoc
}

saveEvent(saveField.text) {
    fileName = saveField.text
    Save.save(fileName)
}

searchEvent(searchField.text) {
    Keyword = searchField.text
    Search.search(keyword)
}

annotationEvent(annotType) {
    Annotation.insert(annotType)
}
```

```
uploadEvent(uploadField.text) {
        fileName = uploadField.text
        FileOps.extract(fileName)
}
```

5.3.4. Annotation

The annotation class describes a single annotation. The annotation will have a 2D position within the display maintained by field Location[]. The user has the ability to insert annotations of a specific type and at a specific location (x,y). The user also has the ability to delete annotations at a specific location if they exist.

The class invariant is location (x,y) must be within display bounds of 0 <= x <= DisplayPDF.max_x and 0 <= y <= DisplayPDF.max_y

Fields:
Location = (float x, float y)
PDFDoc

Functions:

```
insert(annotType, cursorPosition) {
if(annotType == highlightedText ||
annotType == underlinedText ||
annotType == strikethroughText)
  PDFDoc.addAnnot(new iText.PDFTextMarkupAnnotation(cursorPosition, annotType))
 } else if (annotType == rectangularBox) {
  PDFDoc.addAnnot(new iText.PDFSquareAnnotation(cursorPosition))
 }
}

delete(cursorPosition) {
PDFDoc.removeAnnot(cursorPosition)
```

```
}
```

5.3.5. Save

The save class handles the saving of PDF file to the disk and exiting the process.

```
Filename
save(PDFDoc, filename) {
    savePDF to disk
    exit()
}
```

5.3.6. Delete

The delete class will handle the deletion of individual pages from the PDFDoc.

Fields:
pageLocation

Functions:

```
deletePage(pageNumber) {
    For all elements in PDF Doc
        If element page number == pageNumber
            Remove element from PDFDoc
}
```

5.3.7. Search

The search class handles the searching operation, such that a keyword is searched for in the document, and the cursor is updated to the first location if it exists.

Fields:
PDFDoc

Functions:

```
search(keyword) {
        For all elements in PDFDoc
                if(element.content == keyword)
                        displayPDF.moveCursor(element.location)
}
```

5.3.8. FileOps

The fileOps class handles the uploading from document and checking for validity of PDF format.

Functions:

```
extract(input_filename) {
        If(input_filename exists) {
                File = readFile(input_filename)
                if (iText.validPDF(file))
                        PDFDoc = file
                Else
                        PDFDoc = null
        }
}
```

6. Implementation

This section contains the implementation of the product.

6.1 Build Instructions

The source code is given as a tar ball and an executable jar. To see the source code, simple extract the tar ball. To execute the code, double click the jar file or type java -jar JAR_FILE from command line, where jar_file is replaced with jar path name. This command should cause the initial GUI to appear.

Instructions on how to interact with the GUI are given below in the “Smoke Test” section with the screenshots.

6.2 “Smoke Test” and Screen Shots

The smoke test was completed by uploading our requirements document as the test PDF (with some existing annotations added from PDF-XChange editor). Then we displayed contents of the PDF to a GUI using our viewer. Within this viewer the user has the ability to add and remove annotations. Lastly the user saves the file to a PDF and screenshots of the final result in PDF-Xchange editor is given.

There is a few major known issues within the implementation. First the viewer only displays the textual content of a pdf, and does not display the annotations. As a result, when the user is adding and removing annotations, the content is not displayed live. Despite this issue, the annotations are edited in the final output file. Next the “underline” and “strikethrough” annotations are not created properly. The annotations are added but simply displayed as blue and red rectangles respectively.

Within the smoke test screen shot descriptions are instructions on how to interact with the GUI.

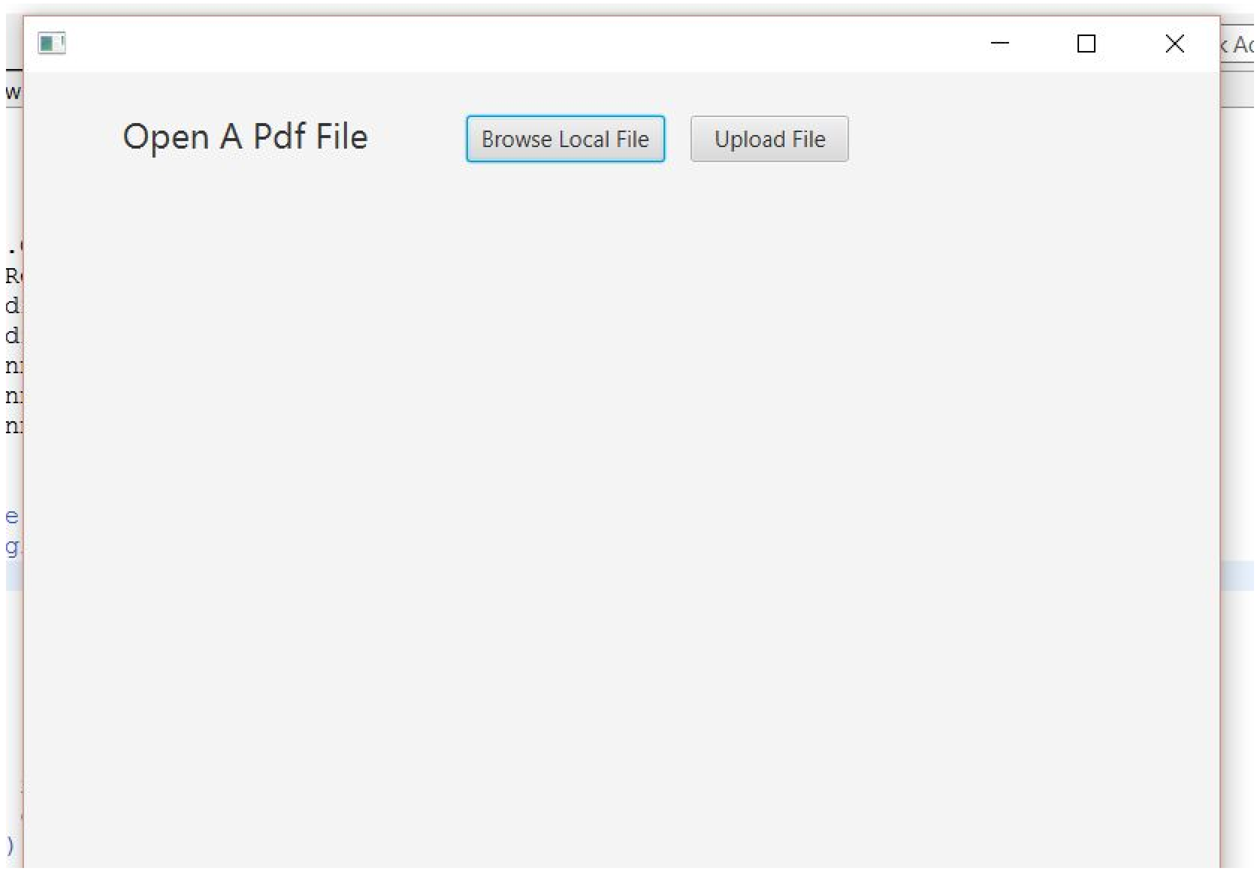
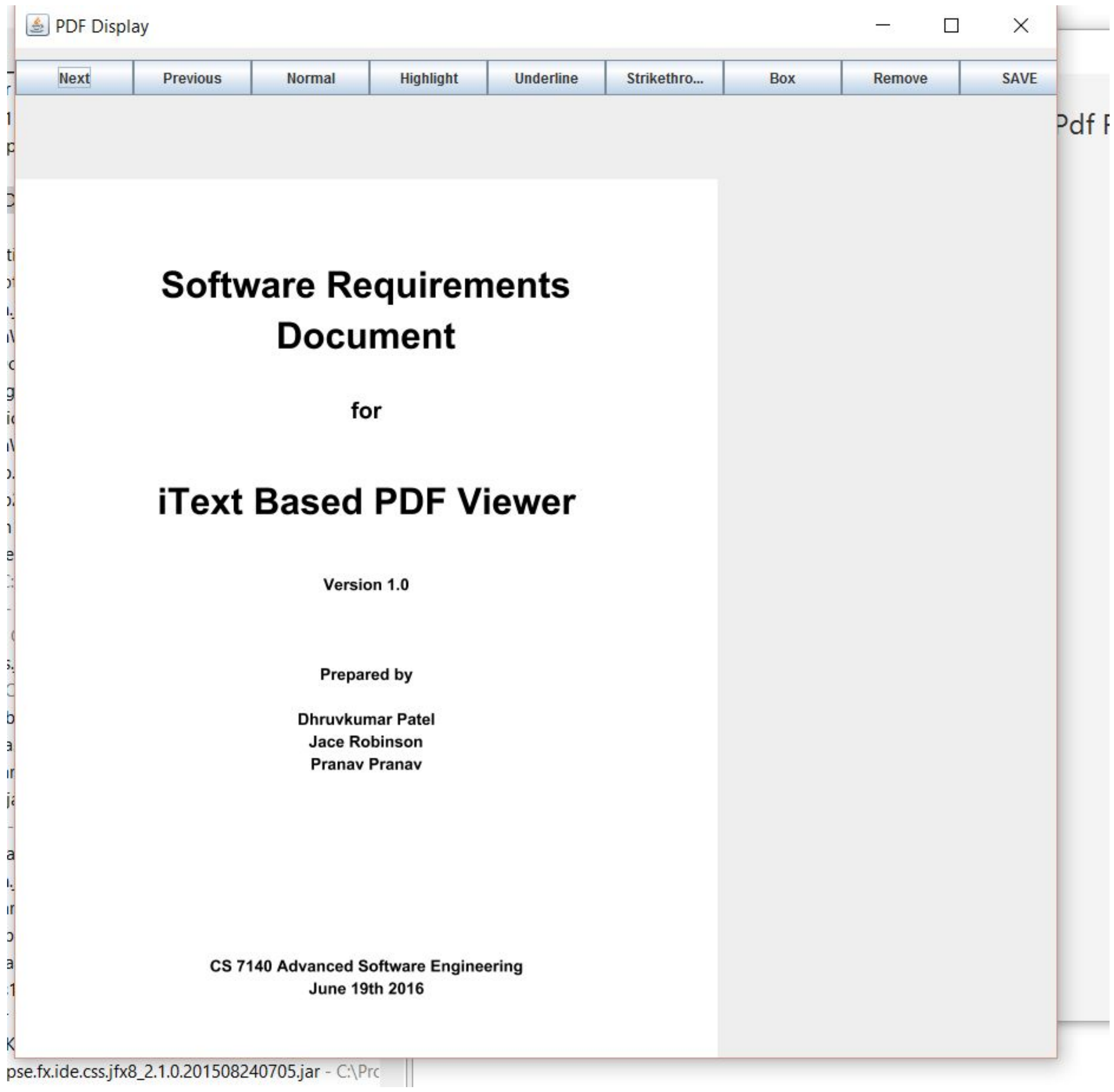


Figure 6.2.1 Initial screen: User is to select a file using browse local file button, then select upload file.



Software Requirements Document

for

iText Based PDF Viewer

Version 1.0

Prepared by

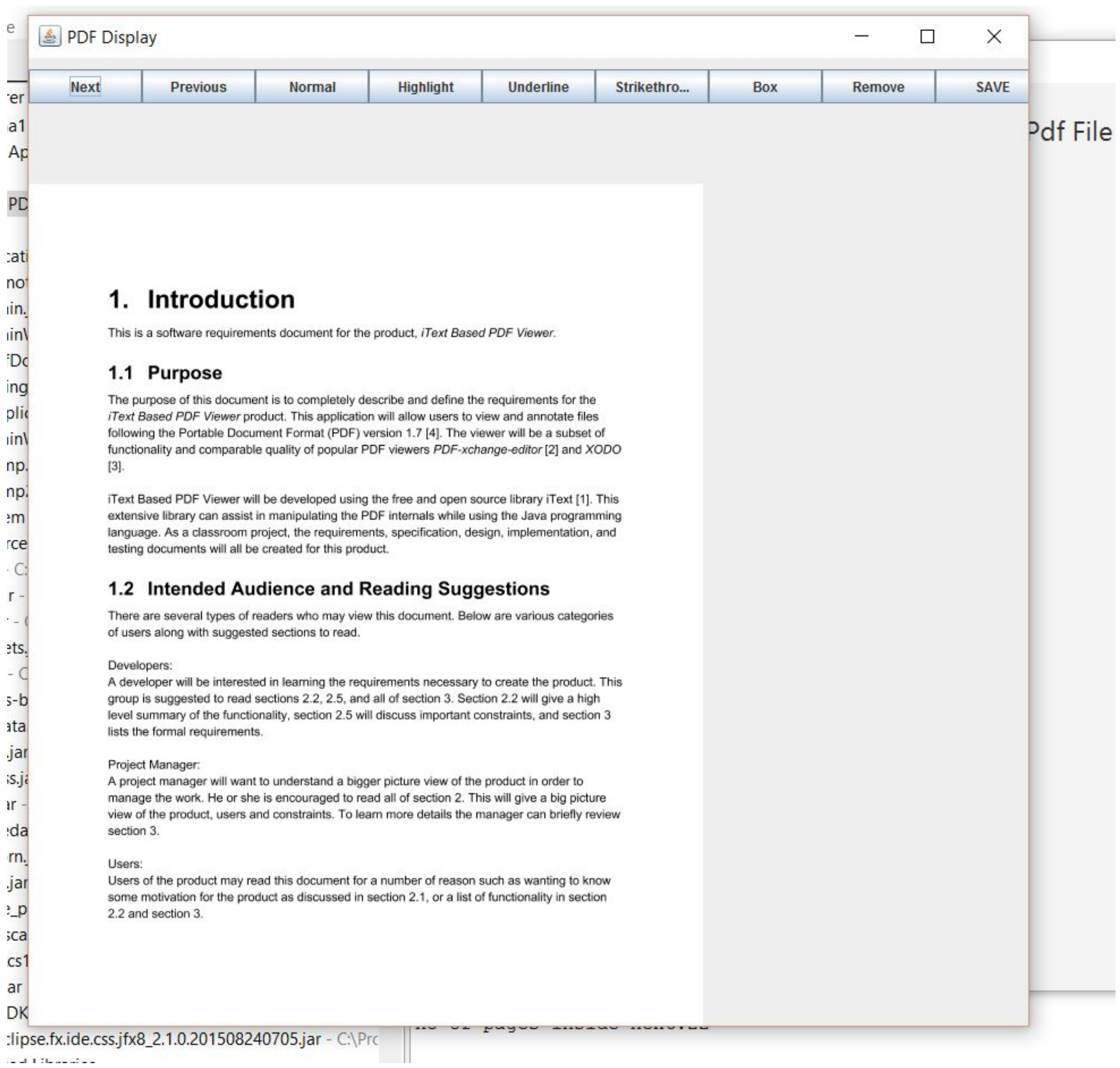
Dhruvkumar Patel
Jace Robinson
Pranav Pranav

CS 7140 Advanced Software Engineering
June 19th 2016

pse.fx.ide.css.jfx8_2.1.0.201508240705.jar - C:\Prc

Figure 6.2.2 Example Viewer: This will bring up the viewer. We can see there are many buttons along the top. Next and previous navigate through the pdf. The mouse cursor will register clicks. When in normal mode (activated by default or clicking normal button), the user clicks will do nothing. When the highlight button is pressed, two user clicks are used to create an annotation rectangle. The first click gives the upper left corner (or lower left), and the second click gives the lower right (upper right). These annotations are not displayed to the screen, but they will be written to the output file. Click each type of annotation button will change which type of annotation is created. The delete button will cause the user clicks to delete an annotation which

contains the cursor location. This functionality is able to remove existing annotations and newly added annotations. Lastly the save button prompts user to select save location.



1. Introduction

This is a software requirements document for the product, *iText Based PDF Viewer*.

1.1 Purpose

The purpose of this document is to completely describe and define the requirements for the *iText Based PDF Viewer* product. This application will allow users to view and annotate files following the Portable Document Format (PDF) version 1.7 [4]. The viewer will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

iText Based PDF Viewer will be developed using the free and open source library iText [1]. This extensive library can assist in manipulating the PDF internals while using the Java programming language. As a classroom project, the requirements, specification, design, implementation, and testing documents will all be created for this product.

1.2 Intended Audience and Reading Suggestions

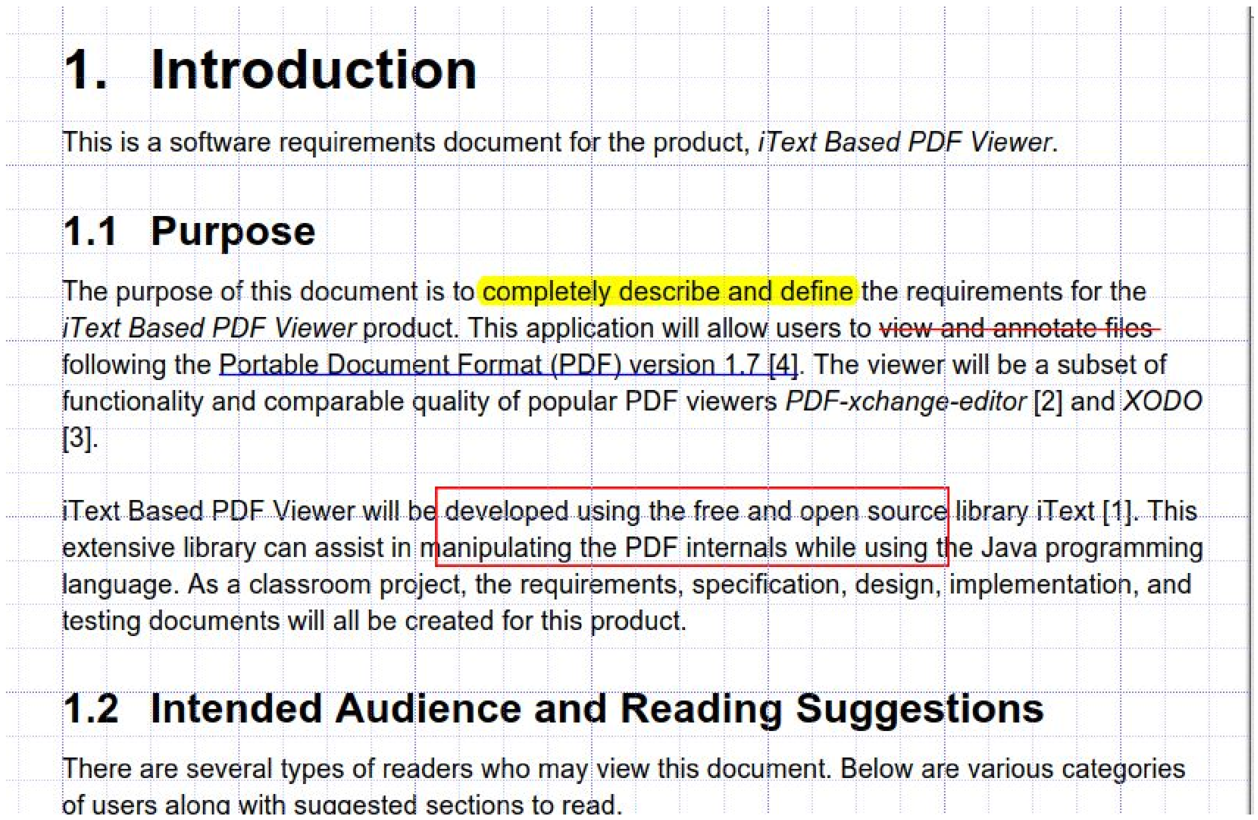
There are several types of readers who may view this document. Below are various categories of users along with suggested sections to read.

Developers:
A developer will be interested in learning the requirements necessary to create the product. This group is suggested to read sections 2.2, 2.5, and all of section 3. Section 2.2 will give a high level summary of the functionality, section 2.5 will discuss important constraints, and section 3 lists the formal requirements.

Project Manager:
A project manager will want to understand a bigger picture view of the product in order to manage the work. He or she is encouraged to read all of section 2. This will give a big picture view of the product, users and constraints. To learn more details the manager can briefly review section 3.

Users:
Users of the product may read this document for a number of reason such as wanting to know some motivation for the product as discussed in section 2.1, or a list of functionality in section 2.2 and section 3.

Figure 6.2.3 Example Viewer 2



1. Introduction

This is a software requirements document for the product, *iText Based PDF Viewer*.

1.1 Purpose

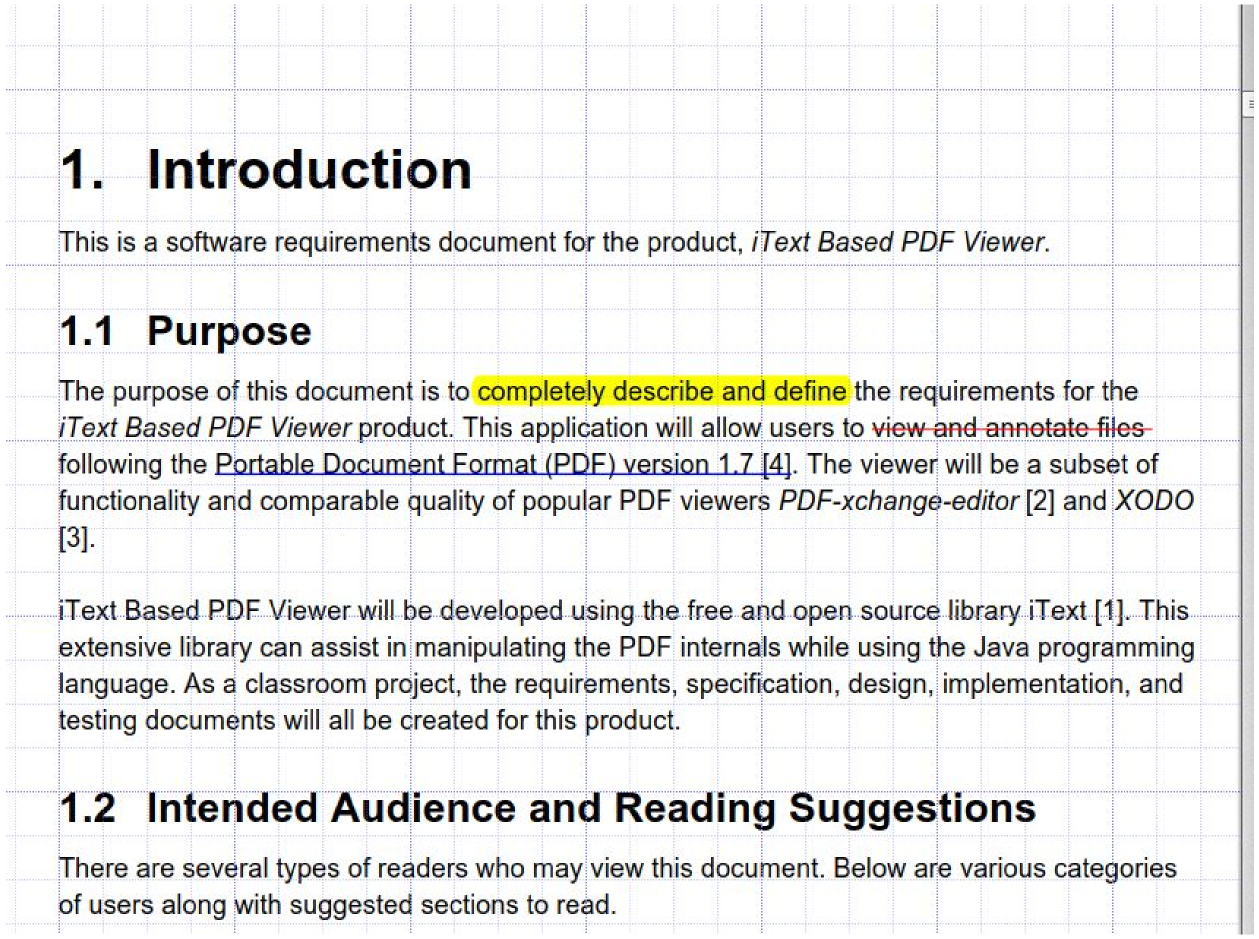
The purpose of this document is to completely describe and define the requirements for the *iText Based PDF Viewer* product. This application will allow users to ~~view and annotate files~~ following the Portable Document Format (PDF) version 1.7 [4]. The viewer will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

iText Based PDF Viewer will be developed using the free and open source library iText [1]. This extensive library can assist in manipulating the PDF internals while using the Java programming language. As a classroom project, the requirements, specification, design, implementation, and testing documents will all be created for this product.

1.2 Intended Audience and Reading Suggestions

There are several types of readers who may view this document. Below are various categories of users along with suggested sections to read.

Figure 6.2.4 Before remove: Here we can see a screenshot of the previously added annotations by PDF-Exchange. Note that these next few screen shots are all taken from the PDF-Exchange editor and not from our viewer.



1. Introduction

This is a software requirements document for the product, *iText Based PDF Viewer*.

1.1 Purpose

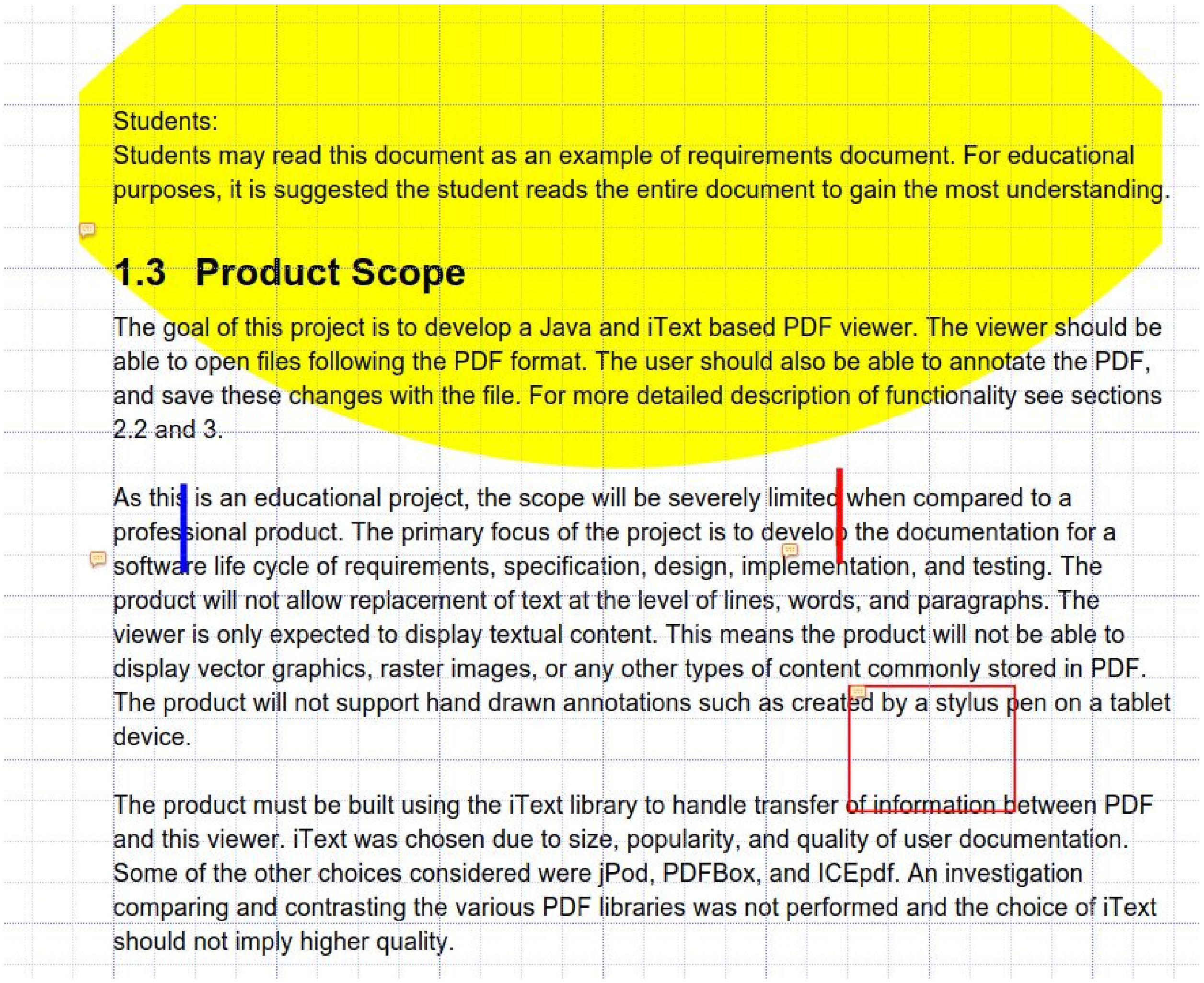
The purpose of this document is to completely describe and define the requirements for the *iText Based PDF Viewer* product. This application will allow users to ~~view and annotate files~~ following the Portable Document Format (PDF) version 1.7 [4]. The viewer will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

iText Based PDF Viewer will be developed using the free and open source library iText [1]. This extensive library can assist in manipulating the PDF internals while using the Java programming language. As a classroom project, the requirements, specification, design, implementation, and testing documents will all be created for this product.

1.2 Intended Audience and Reading Suggestions

There are several types of readers who may view this document. Below are various categories of users along with suggested sections to read.

Figure 6.2.5 After remove: We can see the red box in the bottom portion of the image has been removed.



Students:
Students may read this document as an example of requirements document. For educational purposes, it is suggested the student reads the entire document to gain the most understanding.

1.3 Product Scope

The goal of this project is to develop a Java and iText based PDF viewer. The viewer should be able to open files following the PDF format. The user should also be able to annotate the PDF, and save these changes with the file. For more detailed description of functionality see sections 2.2 and 3.

As this is an educational project, the scope will be severely limited when compared to a professional product. The primary focus of the project is to develop the documentation for a software life cycle of requirements, specification, design, implementation, and testing. The product will not allow replacement of text at the level of lines, words, and paragraphs. The viewer is only expected to display textual content. This means the product will not be able to display vector graphics, raster images, or any other types of content commonly stored in PDF. The product will not support hand drawn annotations such as created by a stylus pen on a tablet device.

The product must be built using the iText library to handle transfer of information between PDF and this viewer. iText was chosen due to size, popularity, and quality of user documentation. Some of the other choices considered were jPod, PDFBox, and ICEpdf. An investigation comparing and contrasting the various PDF libraries was not performed and the choice of iText should not imply higher quality.

Figure 6.2.6 Lastly we can see the result of the annotations added using our viewer. As you can see, the underline and strikethrough annotations are given at the location of the user clicks but not displayed properly.

6.3 Source Code Documentation

The below source code documentation is created from doxygen [5].

7 Software Testing

7.1 Purpose of The Test Plan Document

The Test Plan documents and tracks the necessary information required to effectively define the approach to be used in the testing of the project's product. The Test Plan document is created during the Planning Phase of the project. Its intended audience is the project manager, project team, and testing team. Some portions of this document may on occasion be shared with the client/user and other stakeholder whose input/approval into the testing process is needed.

7.2 Conformance Testing

7.2.1 Test Risks / Issues

Untimely program crashing can be considered as most prominent risk otherwise no major risk.

7.2.2 Items to be Tested / Not Tested

Item to Test	Test Description	Test Date	Responsibility
Functional requirements	All the desired features like, inserting and deleting annotations.	07-20-16	Team
UI/Display	Out of Scope, NOT to be tested	NA	NA

7.2.3 Test Approach(s)

Unit Tests are required to be written at modules level.
At the time of Integration all unit tests are required to be pass.

Critical requirements should be flag for Acceptance testing.

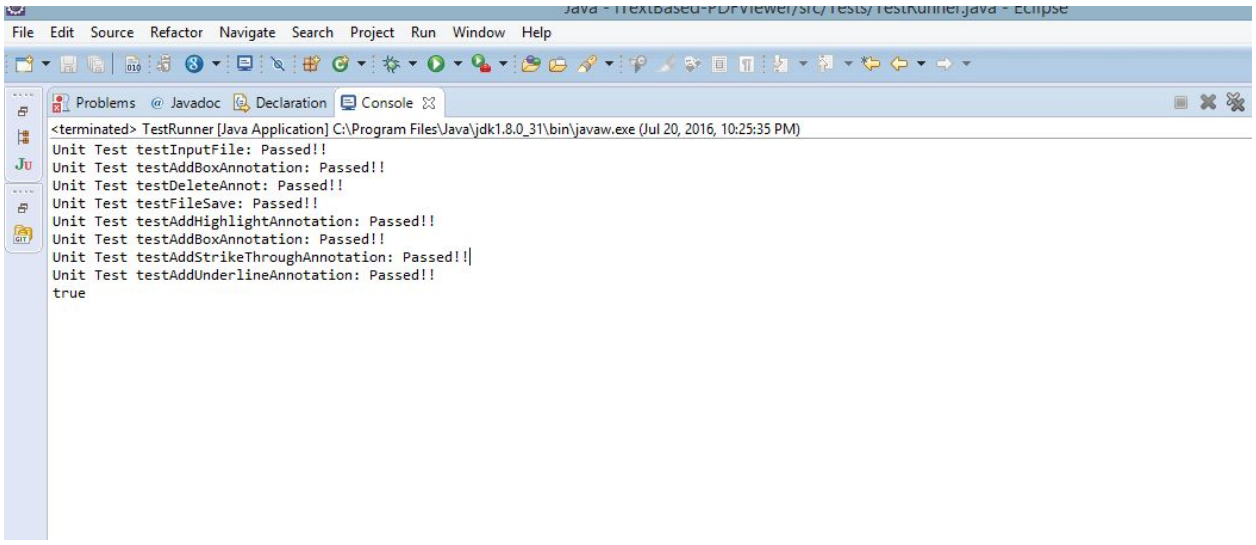
7.2.4 Test Pass / Fail Criteria

Whenever module fulfills the goal of producing the desired output and cross verified by assertions, it is considered as pass otherwise fail.

7.2.5 Test Entry / Exit Criteria

To satisfy the pre condition, all entry level variables should be provided to the unit test to check for the validity of the logic written in the module. Whenever it qualify the pass/fail criteria, it is considered good for exit.

7.2.6 Test Deliverables



```
<terminated> TestRunner [Java Application] C:\Program Files\Java\jdk1.8.0_31\bin\javaw.exe (Jul 20, 2016, 10:25:35 PM)
Unit Test testInputFile: Passed!!
Unit Test testAddBoxAnnotation: Passed!!
Unit Test testDeleteAnnot: Passed!!
Unit Test testFileSave: Passed!!
Unit Test testAddHighlightAnnotation: Passed!!
Unit Test testAddBoxAnnotation: Passed!!
Unit Test testAddStrikeThroughAnnotation: Passed!!
Unit Test testAddUnderlineAnnotation: Passed!!
true
```

Figure 7.2.6.1

Screenshot of Unit Tests all passing during an execution of product.

7.2.7 Test Environmental / Staffing / Training Needs

Standard PC with any OS is required to create testing environment. Team is enough as staff. No training required

7.3 Functional Testing

7.3.1 Test Risks / Issues

Untimely program crashing can be considered as most prominent risk otherwise no major risk.

7.3.2 Items to be Tested / Not Tested

Item to Test	Test Description	Test Date	Responsibility
Annotation operations	Insertion and deletion functionality	07-20-16	Team
File operations	Input and processed output functionality	07-20-16	Team

7.3.3 Test Approach(s)

Same as above.

7.3.4 Test Pass / Fail Criteria

All the test of same category is required to pass in one batch. Otherwise same as above.

7.3.5 Test Entry / Exit Criteria

Same as above.

7.3.6 Test Deliverables

Same as above.

7.4 Performance Testing

7.4.1 Test Risks / Issues

Start with no lag and untimely crashing.

7.4.2 Items to be Tested / Not Tested

Item to Test	Test Description	Test Date	Responsibility
Program start	Product should start on click of exe.	07-20-16	Team
Program stop	Product should give proper output.	07-20-16	Team

7.4.3 Test Approach(s)

Run project with help of readme.txt

7.4.4 Test Pass / Fail Criteria

Manual verification.

7.4.5 Test Entry / Exit Criteria

Same as above.

7.5 Regression Testing

7.5.1 Test Risks / Issues

All modules should be interacting to each other.

7.5.2 Items to be Tested / Not Tested

Item to Test	Test Description	Test Date	Responsibility
All unit test	All unit test should pass	07-20-16	team

7.5.3 Test Approach(s)

Same as above.

7.5.4 Test Pass / Fail Criteria

Same as above.

7.5.5 Test Entry / Exit Criteria

Same as above.

7.5.6 Test Deliverables

Same as above.

7.6 Unit Testing

7.6.1 Test Risks / Issues

Each module should be working and throughput should be as per desired.

7.6.2 Items to be Tested / Not Tested

Item to Test	Test Description	Test Date	Responsibility
testInputFile	Unit test for checking the inputs	07-20-16	team
testFileSave	Unit test for checking the output	07-20-16	team
testAddUnderlineAnnotation	Unit test for checking UnderLine annotation are inserted or not	07-20-16	team
testAddHighlightAnnotation	Unit test for checking Highlight annotation are inserted or not	07-20-16	team

testAddStrikeThroughAnnotation	Unit test for checking Strikethrough annotation are inserted or not	07-20-16	team
testAddBoxAnnotation	Unit test for checking Box annotation are inserted or not	07-20-16	team
testDeleteAnnot	Unit test for checking any annotation are deleted or not	07-20-16	team

7.6.3 Test Approach(s)

Same as above.

7.6.4 Test Pass / Fail Criteria

Same as above.

7.6.5 Test Deliverables

Same as above.

7.7 User Acceptance Testing

7.7.1 Test Risks / Issues

Any kind of bugs at this stage should be prior identified.

7.7.2 Items to be Tested / Not Tested

Item to Test	Test Description	Test Date	Responsibility
Annotation operations	High priority tests	07-20-16	team

7.7.3 Test Approach(s)

All high priority tests are collected together and performed in front of customer including if he/she had demanded any. All should be pass.

7.7.4 Test Pass / Fail Criteria

Customer opinion matters. Apart from customer demanded tests, rest all will be same as above.

7.7.5 Test Entry / Exit Criteria

Same as above.

7.7.6 Test Deliverables

Along with customer demanded tests, rest all will be same as above.

7.7.7 Test Environmental / Staffing / Training Needs

Same as above.

8 Conclusion

8.1 Revisions to Existing Documents

During the revision of the previous documents, the authors made minor changes. First in all documents, some of the introductions and motivations were removed. These sections were all merged into a single introduction section and motivation sections for the entire document. Second there were minor revisions such as updating figure numbers, and adding additional formatting. No additional changes were made to the source code of the implementation.

Additionally there were some requirements known to be incomplete by the others. We cannot implement Search functionality and split pdf by pages or re arrangement pdf by words. In Pdf Rendering part we used PDF Renderer [6] but we are depending on their inbuilt features So, we can not did pdf rendering part properly Otherwise, overall we satisfied all the requirements and specifications in Implementation and testing of iText Based PDF Viewer.

8.2 Summary of Experience

In this final section, the authors describe the hindsight and experience with the document development process. A response for each specific document is provided.

The requirements document is the easiest but most important of all the documents. It is vital to clearly define the features to be included in the software to ensure an agreement between the customers and developers is met. In hindsight, the requirements document within this report is of high quality. The amount of possible confusion to a reader through violation of Bertrand Meyer's seven sins is minimal. The only major downfall of the requirements was the inclusion of too many requirements that were not feasibility completable in the time constraints of the project.

The creation of a specification document was a new experience. Having to describe a PDF Viewer without ambiguity in a specification language was a difficult task. Due to the mathematical experience of the authors and the material covered in class, math and logic was chosen as the specification language. When writing the document, it was difficult to completely describe a PDF document and its viewer without going into too much detail. Overall the authors felt they did a good job as creating a specification which met the requirements.

The design document was the most difficult document for the authors. To be able to create a correct design document, the authors should have an understanding of the product from beginning to end. At the time of the creation of the document, the authors did not have a confident understanding of how to actually “display” the document to the screen. This led to some mistakes in the design document and ultimately in the implementation. In future projects, more time should be spent to fully understand the topic and create a complete design document which meets the specification.

The implementation document was not very difficult but it was important to describe the known issues of the implementation, namely the inability to display annotations and visually incorrect underlining and strikethrough annotations. When a product is unable to meet all requirements by a given deadline, it is still vital for the developers to document known requirements that were met and know requirements that were missed. This is important for the customer and possible future developers to know the state of the product. The implementation of the source code was relatively difficult for two major reasons. First the use of iText required the studying of the API and tutorials. Most functionalities were clearly described but some important functions were not. For example, the ordering of the coordinates of the rectangles for annotations were not clearly defined by iText. We had to reverse engineer the coordinates. Secondly, the Viewer provided by *PDF-Renderer* was only able to display textual content, and could not display annotations. This issue was not noticed during the design and forced some requirements to be missed. In future projects, more time needs to be spent understanding the capabilities and limitations of outside libraries to be included in the project.

The testing document was fairly easy when compared with the above mention documents. Since we learnt the basic of assertion and implemented specification as modules of entry and exit conditions, creating test plan had become straightforward. As per test document, we were exposed to the importance of a test plan and how it saves the day. Test plans not only helps to check on quality of the product but as well ensure the integrity and robustness of the product. Under each phase of testing, conformance, functional, performance, regression, unit and acceptance, we learnt that how each set of tests is validating the notion of implementation meeting design, design meeting specification, and specification meeting the requirements. We were unable to test the UI but instead we tested all the required modules which were mentioned under specification module. In the hindsight, we learn how to estimate the risk involved into each section of testing, how to list all the parts that should be considered under testing and the part that remain out of testing area, how to approach and write test plans, and what are the pass/fail and entry/exit criteria for the different kinds of testing. Test deliverables are very crucial to understand. How to measure and capture the deliverable correctly is an important call to make. Test deliverables are the end results. We have use to “junit.test” to create test plan in Eclipse environment. While executing test cases, another important thing to take care is test environment, staffing and if required any training required.

Overall the development of all documents for a single product has been a valuable experience. It is important to see how having these documents can lead to improved products. With stable documents to reference, developers will be less likely to make mistakes violating the correctness of the documents as the project grows. The development of these documents is able to expose errors in a developer's understanding early and hence reduce future development costs. This is a good practice more professional developers should follow.

9 Full Journals

9.1 Jace Robinson

5-15-16
Initial Research into PDF format…

Start Day 5-31-16
Today I will be setting up a project on github, using intellij IDE, and experimenting with iText.

I have successfully synced project with intelli and imported iText libraries.

Successfully ran initial example that writes to PDF file.

I skimmed the remaining chapters of the itext tutorial, it seems worthwhile to read through when I have time.

End Day 5-31-16

Start Day 6-5-16
Today I am creating initial requirements and specifications for PDF project.

These documents should cover both iText and our product.

Requirements)

Opening

Open an existing PDF following PDF Spec version 1.7.

Open an existing PDF that slightly deviates from PDF Spec version 1.7*** (currently not sure how to define slightly deviating)

Create a new blank PDF.

Saving

Save a modified PDF file under current filename.

Save a PDF file to separate filename specified by user.

Save subset of pages selected by user to separate PDF file with filename specified by user.

Edit and Searching

Find location(s) of text matching a key text provided by user.

Highlight the text matching the key text and call this the MATCHED_TEXT.

Allow selection of text with cursor, this is SELECTED_TEXT.

An annotation consists of a location, MATCHED_TEXT or SELECTED_TEXT in the PDF, a title, and a description.

Allow user to add annotations.

Location of annotation is determined by user cursor location or user input.

Allow user to edit title of annotation.

Allow user to edit description of annotation.

The MATCHED_TEXT or SELECTED_TEXT can be highlighted, underlined, or line-throughed.

Redacting consists of covering a portion of the PDF with a black rectangle such that content in the same location as the rectangle is no longer contained in PDF.

Allow user to redact content in PDF.

Displaying PDF

Allow user to zoom on PDF.

Allow user to rotate a PDF 90 degrees.

Display a continuous scrolling PDF.

Provide ability to navigate by scrolling.

Provide ability to navigate entire page at a time.
User shall be able to view document properties.
Document properties include file size in bytes, location of file, file name, creation date, and latest modification date.

User Interface
Display content of a PDF page to screen.
User has ability to navigate PDF pages using scroll bar.
Text field displaying current page out of total pages.
Button to navigate back a single page.
Button to navigate forward a single page.
Button to zoom in document.
Button to zoom out document.
Slider to adjust zoom level.
Button to enable/disable annotation mode for cursor.
When annotation mode is enabled, the location a user clicks on the PDF will create an empty annotation.
Button to save PDF under current filename.
Button to 'save as' under a different filename.
Button to open PDF.
Sidebar displaying preview of PDF pages.
Sidebar displaying sequential list of all annotations.
Text field to input key text for searching.

Performance
Program must display user interface within 10 seconds on initial execution.
User shall be able to use application in Windows, Linux and OSX with Java SE Runtime Environment version 1.8.0_91 installed.

Robustness
Software does not crash when opening a file of non PDF format.

Documentation

External documentation attempts to explain all user interface GUIs.
Source code contains entry and exit conditions for all functions.

Other (ideas that may or may not be included in requirements based on time constaints)
Allow user to print PDF to connected printer device.
Allow user to email PDF to a specified email address.
Allow user to merge two or more existing PDFs.
Allow user to save PDF directly to a google drive account.
Allow user to encrypt saved PDF.
Allow user to interact with form fields.
Allow user to provide digital signature.

P.S. here is an example of requirements specification I used as reference
https://reqview.com/ext/ReqView-SRS_Software_Requirements_Specification_Example.html.

End Day 6-5-16

Start Day 6-8-16

Today I am meeting with Dhruv and Pranav.

First task is defining high level specifications for the project.

Input: PDF compliant document

Output: PDF compliant documentation

Input-Output: PDF + annotation, input = output

End Day 6-8-16

Start Day 6-14-16

Today I am meeting with Dhruv and Pranav.

Our last meeting was not productive as we were looking at specifications instead of requirements, now that we know what to work on, we should be more productive.

My goal by the end of today is to decided on a requirements document template, divide some work among the different sections of document, and discuss some of the other items on the rubric.

I found a template suggested by IEEE in section 5 here
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574

Erik Buck had the highest quality requirements on piazza so we will use his as a reference to start ours. I was also happy with mine although there were too many.

For help with correctly describing technical details such as text, I will be looking in the official PDF reference document.

We have created a google document containing our requirements document. We are using a format very similar to the IEEE template discussed above.

We have divided the document among user, I will take the introduction and some

End Day 6-14-16

<u>6-15-16</u>
Today I am writing the introduction for the requirements document. This includes purpose, scope, audience, and a few other small sections.

<u>6-16-16</u>

Today I am writing some more sections of the document, including all of section 1, sections 2.3-2.7, and section 3.5. So far we have content in all sections initially complete except 'unique functionality' as proposed in the rubric. Later today Drhu, Pranav and I will be meeting. I would like to spend some time ensuring the requirements in section 3 are complete and follow the necessary qualities described by Bertrand Meyer. The qualities expressed as the 'seven sins of specifier' are…

1) Noise
2) Silence
3) Over-specification
4) Contradiction
5) Ambiguity
6) Forward
7) Wishful thinking

To avoid any issues, we are defining all keywords at the beginning of the requirements section. The document is coming along nicely. **Each individual completed first draft of their section of the work within the desired time frame.** We intend to submit completed document on Saturday night. The current document today is mostly complete and just needs proof read.

<u>Start Day 7-11-16</u>
Today we are beginning the specification document. The first task is to find and understand an appropriate template. I am confident in our requirements document so now it is a matter of writing a spec which meets this. We decided to use basically the same template as requirements but create specifications that meet each requirement.
<u>Start Day 7-12-16</u>
Today we are beginning the design document. We have a rough outline of the main components of our viewer.
A pdf is a collection of objects completely describing the content. Specifically, there exist *text objects* which completely describe the text (this includes font).
PDF *Document Structure* describes pages, fonts, and annotations.
3.4 File Structure contains body which make up document. Sequence of indirect objects
File Structure

How is text stored?? Text is font, position, and characters (text objects)

Glyphs are organized into fonts

Character is abstract entity such as “A” (and not its font)

How does text “wrap” and not extend past border of page?? Using matrix operations to determine “text space”

Annotations

Text Markup Annotation (highlight, underline, strikethrough), 4 (x,y) pairs describing coordinates of rectangle to markup (text will be in the rectangle).

Square and Circle Annotation – rectangle is represented as difference of two rectangles, along with their positions

Annotation are separate objects from the text, they are simply given the same coordinates.

How are pages determined? A page is a collection of content stream

Entire pdf is collection of pages (with other stuff but not relevant to us)

Now looking at iText

Relevant Classes/Functions:

Canvas: Where content is placed, does not know what page it is

Document: root object

DocumentRenderer:

LayoutPosition:

Paragraph: self contained block of text

IElement: takes physical space on canvas

PDFAnnotation

PDFArray

PDFDocument

PDFMarkupAnnotation

PDFPage

PDFSquareAnnotation

PDFTextMarkupAnnotation

Rectangle

Examples and APIs worth keeping track of:

com.itextpdf.kernel.pdf.annot

com.itextpdf.kernel.pdf.canvas

TextMarkup example (with highlight)
http://developers.itextpdf.com/content/itext-7-jump-start-tutorial/chapter-4-making-pdf-interactive
TextExtractionStrategy() to read text from pdf
http://developers.itextpdf.com/examples/content-extraction-and-redaction/parsing-pdfs
PDFStamper to use modify/create new pdf from existing pdf
<u>Start Day 7-13-16</u>
First task is to continue writing specification. I have decided to overhaul a large portion of our existing specification to give more rigorous “math” solution. The main idea to array of pages, where each page is array of text and annotations. The grid is a two-grid. All other operations are built around these basic ideas.
There were some difficulties writing the specification for how the content is “displayed” but a reasonable solution was met.

<u>Start Day 7-19-16</u>
Today I will be continuing the implementation for the pdf viewer. Dhru has a good start available on github, https://github.com/Dhruvkumarpatel/iTextBased-PDFViewer, we will be working from that. The design document was “weak” in my opinion so some extra effort will be necessary for a good implementation.
As of today we are able to open and save pdfs, but are having troubles determining how to “display” the pdf to screen. For now I will develop functionality for adding and removing annotations and simply view the pdf outside of the program.
<u>Start Day 7-20-16</u>
Today we will continue working on PDF Viewer. My first goal of the day is functionality to add and remove annotations from within the hardcoded solution. I am ignoring implementing the viewer for now.
Given an (x,y,width,height, pageNumber), we are able to add a highlight annotation around that text. I will now look to do the same of underline, strikethrough, and boxing. I successfully completed this.
Dhru was able to get the viewer working! Yay! Now that are able to successfully view the pdf, we must new add annotations using GUIs. If we can identify “text positions” or positions based on user clicks, we should be able to add annotations.
Sadly the PDF-Renderer viewer we are using is UNABLE to show ANNOTATIONS. It does successfully display text. We attempted to implement alternatives but are having troubles. To

proceed forward, we are completing the remaining functionality of adding and removing annotations. This will still be possible but the annotations will not display in our viewer. If you open the document in a different viewer such as xodo, the annotation will display.
We now have adding and removing annotations functionality complete. The accuracy of mouse clicks is off by a bit, but is at least on same page as user interacts. The underline and strikethrough are not created properly, they are added to the pdf file but do not actually display underline/strikethrough. This seems to be a problem with the iText implementation as the highlight and box work as expected.
Lastly is the wrap up of implementation and testing document. Dhruv and Pranav were able to create the unit testing, while I completed the doxygen and executable jar file.
All done!

Start Day 7-25-16
Today we are creating the final report for the project. The first task is to merge all documents and make minor changes. Second task is to create new introduction chapter, revisions chapters, and final summary/opinion chapters. The summary provided interesting insight into the progress of the documents. I personally felt confident in our requirements, specification and testing document, but was relatively unhappy with the design and implementation. Due to a lack of time and personally weak programming skills, we were unable to complete those portions to my expectations. Overall the project was a valuable experience to learn the difficulties in creating good documents for a small/medium sized software project.

9.2 Dhruv

Start date : 5-15-16

Professor described Project and we all ready on iText Based Pdf Viewer and decided to implement in Java Programming Language.

Start Date :5-25-16

I installed iText7 Pdf library in my eclipse and start maven project to simple create a pdf file and tried to insert image, Text, annotation into pdf using iText7 Tutorials.

<u>Start Day 5-31-16</u>

Today I joined github Project Repository which is created by my group member and experimenting with iText
using Eclipse IDE.

I have successfully synced project with Eclipse and imported iText libraries.

Successfully implemented initial example what i tried earlier.

Also i used PdfReader functionality from iText 7 Tutorials.

<u>Start Day 6-5-16</u>

Today I am trying to define initial Requirements and Specification of iText Based Pdf Viewer.

I read Pdf Documents for few topics then i wrote following requirements and specification on Piazza.

Requirements)

Requirements :- There are Functional, Non Functional and User Interface Requirements for our PDF-Project.

Functional requirements: 1) user can Read PDF file using this Tool.
2) Display Existing PDF by pages.

3) Search particular word from PDF file and display the cursor to that particular Word.

4) Display all characters, images, etc.

5) user can able to Add,Delete and Edit Annotations.

Non-Functional Requirements : 1) Performance

2) Accessibility

3) Flexibility

Specification :- 1) Design PDF tool which provides Reading, Split a PDF by pages,Edit Annotations, Searching Features.

2) Accuracy is important while extracting a Text.

3) use iText7 library in java to develop all the functionalities.

4) Java Programming language is necessary

5) use JavaFx or Java Swing for GUI part.

Start Day 6-8-16

Today I am meeting with Jace and Pranav.

First task is defining high level specifications for the project.

Input: PDF compliant document

Output: PDF compliant documentation

Input-Output: PDF + annotation, input = output

Start Day 6-14-16

Today I am meeting with Jace and Pranav.

So now we decided to work on Only Requirements Rather than Specification.

we found a template suggested by IEEE in section 5 here
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574
and all three decide to write requirements using this Format of the document.

we discuss our all three's requirements what we wrote on Piazza and tried to make one final Draft of Requirements by points.

we used Google Docs to wrote this document so anyone can write and access document any time.

we divide the sections of the document and commit to complete by next meeting.

<u>start date 6-15-16</u>

Today i wrote my section of portion which contains section 3 in document covered specific Requirements Likes External interface, Functional Requirement, Performance, Software System Attributes and few additional Requirements.

end date 6-15-16

<u>start date 6-16-16</u>

Today we met with the whole document content. So we discussed on each and every section and finalized Report. Also we added Annotations Figure which i generated by program using iText 7 library Tutorials. Also we add Workflow Diagram Figure.

Avoid any errors we just decide to Read through the whole document on saturday and decided to submit on saturday night.

end date 6-16-16

<u>Start Day 7-11-16</u>

we decided to implement specification with the same template of document what we used for Requirement documents.
I learned lifecycle of software engineering system how the system is build? in real world. As of now what i have developed in my career i just start with the
analysis and then directly move on implementation part.

<u>Start Day 7-12-16</u>
we start with specification document and discussed all the math logic, pre and postcondition of all the objects used throughout specification document.
also start to think about design of the system we discussed classes, attributes and operations of the system.
designed UML and Data Flow Diagram using Microsoft Visio tool.
learned how to design whole system and documents.
meanwhile we started coding to make product and which will be helpful to write implementation documents.
discussed design of all the features what we mentioned in the requirement and specification document.
start writing specification and design document.

<u>Start Day 7-13-16</u>
We had a difficulty to display pdf in javafx scrollpane but, then piazza discussion and talked with jace and pranav. Atlast we reached the solutions.
we all divided the section of both documents and we did within time limit.

Start Day 7-14-16

create a javafx project converted into maven and implement github repository and start implementation part with jace and pranav as a collaborator in Github repository. Try to learned iText 7 tutorials and try few things to get better idea. Thinking about pdf rendering and how to display pdf?

Start Day 7-16-16

Try to find pdfrenderer API to display pdf pages. Design basic functionality open existing pdf file and display inside javafx window using pdf renderer Api. Design next and previous button functionality to display pdf by pages and navigate in pdf through pages.try some annotations example using itext7 tutorials.

Start Day 7-17-16

write a review for requirement, specs and design documents.Try PDFBOx, JPedal and other things to display annotated pdf. but, we dont have that functionality for display. because we are depend on PdfRenderer features. start to developing annotation with jace and pranav.

start Day 7-19-16

developed highlights, underline, strikethrough,box annotations. Also developed remove annotation functionality

integrate all the code together. This project give me a good learning skills to

how to manage code while many people are working on same project inside same repository?

start Day 7-20-16

Atlast we developed SAVE functionality where user can able to save file wherever user wants inside local computer.we wrote the Testcases using JUnit and run successfully with all the testcases passed.

implementation document designed by Doxygen which is good tool for documentation.Add source code, readme file and make .exe file so one can open directly to use this tool. wrote Testing Document with all the testcases and screenshots.

Start Day 7-25-16

Overall nice experience while developing this Project. To conclude, in Final Report we changed introduction first chapter which describes overall documents, Add the semi final chapter which describes what changes made after turnin the source code. Add main content from the others documents of this project and try to come up with complete final document. Also add new chapter which describes overall Experience of this project development life cycle.

9.3 Pranav

5-15-16

Group formed. Set channel for communication. Discussed for group meeting whenever required.

Start Day 5-31-16

Read itext on wikipedia and browse through PDF reference from adobe.

End Day 5-31-16

Start Day 6-5-16

Joined project group on github.
Today I am creating initial requirements and specifications for PDF project.

End Day 6-5-16

Start Day 6-8-16
Worked on the requirement and specification.

Requirement

Input: PDF document (abiding official definition of PDF by Adobe)

Output: new PDF document (input plus annotation [Clarification: how annotation is required to display in output?]), all_annotation_together.txt (assuming text file/can be PDF)

GUI:

Load Screen,

Display Screen/s (Clarification: depends on all content on the scrollable pane or pageable pane?),

Edit Screen/s (for annotation: - separate text field to capture user input, NO editing of existing documents; with saving feature and proper success or failure messages).

```
Specification:
-------------------------------------
FileLoad:
assertion1(fn_FileLoad(args), true);
assertion1(fn_CheckFileIntoDir(args), true);
LoadPDF:
assertion1(fn_LoadPDF(args), true);
assertion1(fn_CheckPDFobject(args), true);
ExtractPDFContent:
assertion1(fn_ExtractPDFContent(args), true);
assertion1(fn_ReadContentPrintToLogOrConsole(args), true); //not sure
PreserveContextFromPDF:
assertion1(fn_PreserveContextFromPDF(args), true);
assertion1(fn_CheckByWritingPDFWithNoEdit(Input_PDF), true); //not sure
CaptureEdit:
assertion1(fn_CaptureEdit(args), true);
assertion1(fn_CheckEditWithPrintToLogOrConsole(args), true);
SaveFile:
assertion1(fn_SaveFileWithEditsAndProperMessage(args), true);
```

```
        assertion1(fn_CheckFileFromOutputDir(args), true);
```

We are having our first group meeting today.

Todays discussion is based on course lecture today.

Tried to define high level specification.

End Day 6-8-16

Start Day 6-9-16

As discussed,

1. INPUT

PRE: assertion_pre_input(fn_FileLoad_ExpectFile(File file), true);

POST: assertion_post_input(fn_CheckFile_PDFFile(File file), true); // abide the official definition of PDF (7 bits ASCII file in a COS format.)

2. OUTPUT

PRE: assertion_pre_output(fn_GetObjects_InputPDFObjsAndAnnotationObj(List objs), true); // input PDF is broken down into list of PDF document objects and Annotation objects (type of PDF document)

POST: assertion_post_output(fn_ReturnFile_PDFFile(List Objs), true); // Combining all Objs in proper manner such that the output file is abiding the official definition of PDF (7 bits ASCII file in a COS format.)

3. RELATION

Definition of Input file = Definition of Output file

End Day 6-9-16

Start Day 6-14-16

Second Group meeting.

For our first deliverable we need to produce requirement document not specification document. We pointed out all project requirement that we can figure out from piazza and discussed among us.

Draw figures and debated on requirements. Output was crisp definition of our requirement.

We started filling in the section in the google document created by Jace based on the IEEE format.

We have divided the document among user, I will take the Overall description.

End Day 6-14-16

Start Day 6-15-16

My responsibility is to define section 2. Overall description.

End Day 6-15-16

Start Day 6-16-16

We finished the document. Jace helped me to finish few section of my part in the document.

We are reviewing the document in our group meeting.

Workflow discussed and done with figure.

End Day 6-16-16

Start Day 7-12-16

Group meeting 1: Used rough draft of specs to write first draft of pseudo code.

List down all the classes and its corresponding field.

Draw the first version of the class level diagram.

Draw work from diagram.

Split the task. Formal writing remaining.

Start Day 7-13-16

Group meeting 2: Made formal workflow diagram.

Review document.

Start Day 7-16-16

Meet with dhruv discussed existing issues with rendering of pdf.

A couple of POC implemented.

Start Day 7-20-16

Group meeting: Task divided for the work on remaining of the implementations.

Worked a couple of PoC for Display and rendering.

To meet deadline, rescope goals.

Created unit testing and test report document.