**Software Design**

**Document**

**for**

**Box.com and eDefender Integration**

Version 1.0.0 approved

**Prepared by** Jesus Antonio Alcocer, Justin Bradshaw, Santiago Del Rio Obando,

Anthony Diaz, Anthony Gonzalez, Santhosh Kumar, Yongkang Liu,

Alexander Mendez, Gilbert Perez-Sanchez, Jian Verdad

Santa Barbara Public Defender’s Office

November 1, 2024

Table of Contents................................................................................................................... <pg 2>

Revision History..................................................................................................................... <pg #>

1. Introduction................................................................................................................ <pg #>
   1. Purpose........................................................................................................... <pg #>
   2. Document Conventions……………………................................................. <pg #>
   3. Intended Audience and Reading Suggestions................................................ <pg #>
   4. System Overview........................................................................................... <pg #>
2. Design Considerations............................................................................................... <pg #>
   1. Assumptions and dependencies..................................................................... <pg #>
   2. General Constraints........................................................................................<pg #>
   3. Goals and Guidelines.....................................................................................<pg #>
   4. Development Methods................................................................................... <pg #>
3. Architectural Strategies.............................................................................................. <pg #>
4. System Architecture................................................................................................... <pg #>
   1. ....................................................................................................................... <pg #>
   2. ....................................................................................................................... <pg #>
5. Policies and Tactics.................................................................................................... <pg #>
   1. Specific Products Used.................................................................................. <pg #>
   2. Requirements traceability............................................................................... <pg #>
   3. Testing the software....................................................................................... <pg #>
   4. Engineering trade-offs................................................................................... <pg #>
   5. Guidelines and conventions........................................................................... <pg #>
   6. Protocols........................................................................................................ <pg #>
   7. Maintaining the software............................................................................... <pg #>
   8. Interfaces........................................................................................................ <pg #>
   9. System's deliverables..................................................................................... <pg #>
   10. Abstraction..................................................................................................... <pg #>
6. Detailed System Design............................................................................................. <pg #>

6.x Name of Module........................................................................................... <pg #>

6.x.1 Responsibilities...................................................................................... <pg #>

6.x.2 Constraints............................................................................................. <pg #>

6.x.3 Composition........................................................................................... <pg #>

6.x.4 Uses/Interactions.................................................................................... <pg #>

6.x.5 Resources............................................................................................... <pg #>

6.x.6 Interface/Exports.................................................................................... <pg #>

1. Detailed Lower level Component Design

7.x Name of Class or File..................................................................................... <pg #>

7.x.1 Classification.......................................................................................... <pg #>

7.x.2 Processing Narrative(PSPEC)................................................................ <pg #>

7.x.3 Interface Description.............................................................................. <pg #>

7.x.4 Processing Detail.................................................................................... <pg #>

7.x.4.1 Design Class Hierarchy................................................................... <pg #>

7.x.4.2 Restrictions/Limitations.................................................................. <pg #>

7.x.4.3 Performance Issues........................................................................ <pg #>

7.x.4.4 Design Constraints......................................................................... <pg #>

7.x.4.5 Processing Detail For Each Operation........................................... <pg #>

1. Database Design
2. User Interface
   1. Overview of User Interface.......................................................................... <pg #>
   2. Screen Frameworks or Images..................................................................... <pg #>
   3. User Interface Flow Model........................................................................... <pg #>
3. Requirements Validation and Verification................................................................. <pg #>
4. Glossary..................................................................................................................... <pg #>
5. References................................................................................................................. <pg #>

**Revision History**

| Name | Date | Reason For Changes | Version |
| --- | --- | --- | --- |
| Initial Setup | 11/1/24 |  | 1.0.0 |
| Jian | 12/5/24 | Worked on Section 3: Architectural Strategies | 1.0.0 |
| Yongkang | 12/5/24 | Modified on 5.1, 5.2, 5.3(partially) |  |
| Santiago | 12/5/24 | 2.1, 2.4, Section 7, |  |
|  |  |  |  |

**1. Introduction**

**1.1 Purpose**

Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.

The purpose of this document is to provide the design details that will be used for Discovery Bates Naming Application. This software is a transition from a desktop-based system to a cloud-based one. This is done through AWS Lambda for backend processing and Box.com for file storage and management. The application is designed to process legal discovery PDF files, extract their Bates numbers, and rename the files accordingly to the Bates number

This Software Design Document (SDD) covers the design of the cloud-based solution, including component designs and integration with Box.com. It also includes future features that are planned for subsequent versions of the product

**1.2 Document Conventions**

Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.

Font Conventions:

* Headings: Sections titles are in bold that use hierarchical numbering systems such as 1.1, 1.2, etc.

**1.3 Intended Audience and Reading Suggestions**

Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.

This document is intended for various stakeholders involved in the development and deployment of the Discovery Bates Naming application, including:

* Developers
* Project Managers
* Testers
* System Administrators

Suggested Reading Order:

1. Not determined as of yet

**1.4 System Overview**

Provide a general description of the software system including its functionality and matters related to the overall system and its design (perhaps including a discussion of the basic design approach or organization

The **Discovery Bates Naming Application** is a cloud-based solution designed to automate the processing of legal discovery PDF files. The application extracts Bates numbers, validates their sequence, and renames the files accordingly, ensuring consistent and organized storage. Files are stored securely in **Box.com** and, for intermediate processing, in an **AWS S3 bucket**.

The system leverages a **serverless architecture** using **AWS Lambda functions** for scalability and efficiency. Upon a file upload event to **Box.com**, the application processes the files in memory to preserve data confidentiality and complies with data privacy requirements. The end-to-end workflow ensures seamless automation, reducing manual effort and minimizing errors.

### 1.4.1 Key Features

**1.4.1 Integration with Box.com**

* Triggered by an HTTP API Gateway call when files are uploaded to **Box.com**.
* Retrieves file details, including file ID, folder ID, and file name, from the event payload.

**1.4.2 Custom Payload Creation**

* An initial AWS Lambda function captures the file metadata and creates a custom payload, which is passed to the next Lambda function for processing.

**1.4.3 Bates Number Extraction and Validation**

* The second Lambda function uses the file ID to download the PDF file contents.
* Extracts Bates numbers from the document, ensuring that they are **sequential** and meet validation criteria.

**1.4.4 Automated File Renaming and Organization**

* Renames files based on the extracted Bates numbers and additional metadata (e.g., PD Case Number and Disc Number).
* Stores renamed files in an **AWS S3 bucket** for temporary storage before transferring them to Box.com.

**1.4.5 Serverless Scalability and Security**

* Uses **AWS Lambda** to handle processing tasks dynamically, ensuring scalability with varying workloads.
* Processes files entirely in memory within the Lambda functions to protect sensitive data and ensure compliance with privacy policies.

This system provides a robust and efficient solution for legal professionals by automating discovery file management, enhancing accuracy, and streamlining the workflow.

**2. Design Considerations**

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

**2.1 Assumptions and Dependencies**

Describe any assumptions or dependencies regarding the software and its use. These may concern such issues as:

* Related software or hardware
* Operating systems
* End-user characteristics
* Possible and/or probable changes in functionality

This section outlines the assumptions and dependencies that affect the design, implementation, and operation of the **Discovery Bates Naming Application**.

#### 2.1.1 Assumptions

#### 2.1.1.1 **File Naming and Folder Structure in Box**

* It is assumed that users uploading files to **Box.com** will adhere to the required naming convention: **pdcasenumber\_discnumber**.
* Files in the uploaded folder are in **PDF format** and contain legible Bates numbers for processing.

**2.1.1.2 Bates Numbers Format**

* Bates numbers are sequential and consistently formatted across all pages of each PDF document.
* Bates numbers can be extracted reliably using PDF text extraction libraries.

**2.1.1.3 End-User Characteristics**

* The system assumes end-users (e.g., legal professionals) have basic knowledge of using **Box.com** and are familiar with the upload process.
* Errors related to invalid file naming conventions or missing Bates numbers will be handled via notification emails sent to the uploader.

**2.1.1.4 Data Privacy and Security**

* All processing will occur in memory within **AWS Lambda** to ensure no sensitive data is stored outside of the **public defender's office Box.com system**.
* Users trust the application to handle files securely and comply with applicable data protection regulations.

**2.1.1.5 Future Scalability**

* The system assumes that AWS Lambda's scaling capabilities will handle increased workloads as the volume of discovery files grows over time.

#### 2.1.2 Dependencies

**2.1.2.1 Box API**

* The system depends on the **Box API** for file metadata retrieval, file content downloading, and renaming files back in the Box storage system.
* **Dependency Risk:** Any changes to the Box API (e.g., authentication methods, endpoints) may require updates to the Lambda functions.

**2.1.2.2 AWS Services**

* **AWS Lambda:** Core processing occurs within Lambda functions.
* **AWS S3:** Temporary storage for renamed files before re-uploading to Box.
* **AWS HTTPS API Gateway:** Used to trigger Lambda functions based on events.

**2.1.2.3 Libraries and Tools**

* The system depends on the following libraries and tools to extract Bates numbers, interact with external systems, and handle processing tasks:

**2.1.2.3.1 Pymudf**

* Used for extracting and validating Bates numbers from PDF documents.
* Assumes that the library provides accurate and reliable PDF text extraction and operates efficiently within the AWS Lambda runtime environment.

**2.1.2.3.2 Requests**

* Used for making HTTP requests to APIs, including the Box API, for file metadata retrieval, content downloads, and uploads.
* Assumes compatibility with the Lambda runtime and efficient handling of API requests.

**2.1.2.3.3 Boxsdk(Python)**

* A Python SDK for interacting with the Box API.
* Used for managing file uploads, downloads, metadata handling, and folder organization in the Box storage system.
* Assumes continued compatibility with the latest Box API endpoints and features.

**2.1.2.4 Box Folder and File Permissions**

* The Box folders and files involved in the process must have appropriate permissions for the application to access, rename, and move files.

**2.1.2.5 File and Payload Size**

* The system assumes that individual files and API payloads remain within the size limits of AWS Lambda and the Box API. Large files or folders may exceed these limits, requiring segmentation or adjustments.

### Summary

This section identifies key assumptions, such as reliance on specific file formats and user adherence to naming conventions, and dependencies, including integration with the **Box API**, AWS services, and third-party PDF parsing libraries. These factors provide a framework for understanding the system's current limitations and potential areas for future adaptation.

**2.2 General Constraints**

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is not exhaustive):

* Hardware or software environment
* End-user environment
* Availability or volatility of resources
* Standards compliance
* Interoperability requirements
* Interface/protocol requirements
* Data repository and distribution requirements
* Security requirements (or other such regulations)
* Memory and other capacity limitations
* Performance requirements
* Network communications
* Verification and validation requirements (testing)
* Other means of addressing quality goals
* Other requirements described in the requirements specification

You will not need to include all of these. Only the ones that will influence the design of your software

The design and implementation of the Discovery Bates Naming Application are subject to constraints that directly influence the system’s architecture, performance, and functionality:

1. Hardware or Software Environment:
   1. AWS Lambda: The software is designed to run as serverless functions within contrast of AWS Lambda, which can include limited runtime execution, memory allocation limits, or cold start latency for infrequently used functions (rare case)
2. End-user environment:
   1. Box.com integration with the application is tailored for organizations that are under use of Box.com for file storage and management. Users must have access to Box.com and the appropriate permissions to upload and manage files
3. Availability/Volatility of Resources:
   1. Lambda Layer dependencies imported via Lambda layers must remain stable. Updates or deprecations of these layers could impact the functionality
4. Security Requirements:
   1. Data protections ensure that the software’s data is encrypted at rest and in transit and should be in compliance with Box.com’s encryption standards and AWS’s security policies. But, should also be in compliance with government regulations
5. Network Communication:
   1. The software relies on stable internet connectivity to communicate with Box.com and AWS services
6. Memory Limitations:
   1. Lambda imposes limits on memory allocation and storage for temporary files up to 512 MB.

**2.3 Goals and Guidelines**

Describe any goals, guidelines, principles, or priorities which dominate or embody the design of the system's software. For each such goal or guideline, unless it is implicitly obvious, describe the reason for its desirability. Feel free to state and describe each goal in its own subsubsection if you wish. Such goals might be:

* The KISS principle ("Keep it simple stupid!")
* The Software has a mandatory delivery date that must be met (end of the cd3337 class)
* Emphasis on speed versus memory use
* The product should work, look, or "feel" like an existing product

We used many ways of making sure that the goals, principles, and priorities were all in conjunction when working on the project:

2.3.1 Simplicity:

* Ensure the design and implementation of the system remain simple, clear, and maintainable
* A straightforward design minimizes complexity, reduces development time that can be used on other goals, and can help in easier debugging and future enhancements

2.3.2 Mandatory Delivery Deadline:

* Ensure there are deadlines that need to be met when developing the program and have a release of the functional beta version of the application by **February, 2025.**
* The deadline aligns with the project timelines for stakeholders and also meeting this date is essential for completing the initial rollout and gathering feedback for subsequent iterations that can improve on the application

2.3.3 Emphasis on Performance:

2.3.4 Seamless User Experience:

* Design the application in order to integrate seamlessly with the existing workflow that Santa Barbara Public Defender’s Office has set, more specifically, integrate with the overarching e-Defender System
* Users should experience minimal disruption as they transition from the desktop application to a cloud-based application, ensuring that users have a sort of familiarity when working with the cloud-based application and don’t find themselves confused

2.3.5 Error Handling:

* Incorporate detailed error logging and notification mechanisms, such as the Box email notifications to handle error processing
* A robust error handling ensures transparency and allows for users to quickly identify and resolve issues that may arise when processing documents

2.3.6 Scalability and Flexibility:

* Design the software to be scalable for future enhancements, such as:
  + File organization into Box folders based on their PD Case Number and Disc Number
  + Add features for handling error notifications using Box.com email
* By anticipating for future needs that our application might need, we can ensure that the software remains useful as organizational requirements change and evolve

**2.4 Development Methods**

Briefly describe the method or approach used for this software design. If one or more formal/published methods were adopted or adapted, then include a reference to a more detailed description of these methods. If several methods were seriously considered, then each such method should be mentioned, along with a brief explanation of why all or part of it was used or not used.

These would be things such as the ‘Water Fall Development’ methods, ‘Agile Development’, ‘Unplanned Mad Scramble Development’, or other development models and variations. Describe how these were applied in the case of your project.

The development of the **Discovery Bates Naming Application** followed an **Agile Development** approach. This method emphasizes iterative progress, adaptability, and close collaboration with stakeholders to ensure the evolving system meets user needs effectively.

#### 2.4.1 Application of Agile Development

**2.4.1.1 Iterative Development**

* The project was divided into smaller iterations (or sprints), each focusing on specific functional components such as **PDF parsing, Bates number validation, file renaming,** and **error handling.**
* Each iteration included planning, development, testing, and review phases to ensure incremental progress.

**2.4.1.2 Continuous Feedback**

* Stakeholders, including legal professionals who will use the system, provided feedback during development. This ensured that the application aligns with real-world workflows and user requirements.
* Adjustments were made promptly based on this feedback, allowing for flexibility in addressing changing priorities.

**2.4.1.3 Focus on Working Software**

* The Agile principle of delivering working software early and frequently was followed. Functional pieces of the system were completed and tested independently before integrating them into the larger workflow.

**2.4.1.4 Collaboration and Communication**

* Regular meetings and updates facilitated clear communication among team members and stakeholders. This ensured alignment on goals, progress tracking, and quick resolution of issues.

**2.4.1.5 Adaptability**

* Agile allowed the team to respond effectively to unforeseen challenges, such as API constraints, processing limitations, or unexpected changes in user requirements.

#### 2.4.2 Why Agile Was Chosen

* **Flexibility:** Agile was selected because it allows for incremental progress and adaptation to feedback, making it ideal for a project with evolving requirements.
* **Stakeholder Involvement:** The need for ongoing input from legal professionals made Agile's collaborative nature highly suitable.
* **Risk Reduction:** By developing in smaller, testable increments, potential issues were identified and resolved early, reducing the risk of critical failures late in the development cycle.
* **Time Constraints:** With a fixed deadline, May 2025, Agile ensured continuous progress and iterative delivery of functional components.

This approach ensured that the final system met the functional and non-functional requirements while remaining user-centric, secure, and aligned with the stakeholders' needs.

**3. Architectural Strategies**

* Due to the pre-existence of an application, it was intuitive to re-use many of the same technologies and languages that have been already implemented as well as some new libraries to further develop in this project:
  + Languages:
    - Python
  + Libraries:
    - json
    - re
    - Requests
    - boto3
    - logging
    - BytesIO
    - unquote\_plus
  + Services:
    - AWS Lambda
    - AWS IAM
    - AWS S3
    - Box Skills App
    - Box Custom App
  + Advantages:
    - This allows us to focus more on migrating the application using cloud services rather than recreating pieces of pre-existing functionality from scratch.
* To mitigate the usage of UI for simplifying workflow, we are removing the use of the desktop application Discover Bates Namer and simply asking users to upload a folder containing PDFs and named in a specified format onto Box where it will output a processed copy of the folder and its contents.
  + Using this strategy will allow us to cut the workflow time by cutting out the usage of the desktop application. Additionally, it allows future simplification by shortening the process even more with the potential automation of downloading files, creating folders, and uploading onto Box.
* Error detection will be handled by the AWS Lambda functions as well as an email service provided by AWS to notify a user of any unprocessed files.
  + Although Box has a similar email service, it is best to use AWS’ email service as the errors will occur within its environment, reducing the need for any cross-service communication and complications further down development.
* Memory management will be handled by AWS Lambda’s ephemeral storage between 512 MB and 10,240 MB.
  + Since Lambda’s function calls on each individually uploaded PDF from Box, as long as the PDF’s file-size does not exceed 10,240 MB the software will run normally. Additionally, any PDFs that exceed this limit will trigger the error-handled email service to notify the user of the specific PDF that may need to be trimmed or reduced into a smaller size.
* After beta or final version release, this software will then be integrated into the Santa Barbara Public Defenders Office’s eDefender system and database for use.
* Future plans consist of the following:
  + Rewriting files from AWS S3 output bucket back onto Box
  + Relocating renamed files within folders inside Box
  + Handling error messages and setting up an email notification system
  + Releasing a beta version
* Generalized Approaches:
  + Currently using AWS S3 buckets to temporarily store files. Due to confidentiality and security compliances, will remove the usage of S3 buckets entirely and utilize AWS Lambda’s ephemeral storage.

**4. System Architecture**

**Diagrams will be available after beta release is complete**

This section should provide a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components. Don't go into too much detail about the individual components themselves (there is a subsequent section for detailed component descriptions). The main purpose here is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together to provide the desired functionality.



This is where the level 0 DFD will probably work best.

At the top-most level, describe the major responsibilities that the software must undertake and the various roles that the system (or portions of the system) must play. Describe how the system was broken down into its modules/components/subsystems (identifying each top-level modules/component/subsystem and the roles/responsibilities assigned to it).

Each subsection (i.e. “4.1.3 The ABC Module”) of this section will refer to or contain a detailed description of a system software component.











Level 1 Data Flow Diagrams (DFD) and Control Flow Diagrams (CFD) should probably go here.

Describe how the higher-level components collaborate with each other in order to achieve the required results. Don't forget to provide some sort of rationale for choosing this particular decomposition of the system (perhaps discussing other proposed decompositions and why they were rejected). Feel free to make use of design patterns, either in describing parts of the architecture (in pattern format), or for referring to elements of the architecture that employ them. Diagrams that describe a particular component or subsystem in detail should be included within the particular subsection that describes that component or subsystem.

**5. Policies and Tactics**

Describe any design policies and/or tactics that do not have sweeping architectural implications (meaning they would not significantly affect the overall organization of the system and its high-level structures), but which nonetheless affect the details of the interface and/or implementation of various aspects of the system. Make sure that when describing a design decision that you also discuss any other significant alternatives that were considered, and your reasons for rejecting them (as well as your reasons for accepting the alternative you finally chose). Such decisions might concern (but are not limited to) things like the following (Must include 5.1, 5.2, and 5.3. The rest of these categories or custom ones can be added as needed.):

**5.1 Choice of which specific products used**

(IDE, compiler, interpreter, database, library, etc. ...)

* **AWS Lambda**: Used for serverless processing of files, ensuring scalability and cost-effectiveness.
* **Box API**: Handles file uploads, metadata management, and folder organization.
* **PyMuPDF and Requests Libraries**: Used in Lambda Layers for PDF parsing and file handling.

**5.2 Plans for ensuring requirements traceability**  
 …Describe…

* Create a table mapping each requirement in the SRS to the corresponding software module or function in the SDD.
* Use comments in the code to link specific implementations to their related requirements.
* Maintain documentation updates in parallel with code changes to ensure accuracy.

**5.3 Plans for testing the software**

…Describe…

5.# Engineering trade-offs  
 …Describe…

5.# Coding guidelines and conventions

…Describe…

5.# The protocol of one or more subsystems, modules, or subroutines

…Describe…

5.# The choice of a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality

…Describe…

5.# Plans for maintaining the software

…Describe…

5.# Interfaces for end-users, software, hardware, and communications

…Describe…

5.# Hierarchical organization of the source code into its physical components (files and directories).

…Describe…

5.# How to build and/or generate the system's deliverables (how to compile, link, load, etc.)

…Describe…

5.# Describe tactics such as abstracting out a generic DatabaseInterface class, so that changing the database from MySQL to Oracle or PostGreSQL is simply a matter of rewriting the DatabaseInterface class.

For this particular section, it may become difficult to decide whether a particular policy or set of tactics should be discussed in this section, or in the System Architecture section, or in the Detailed System Design section for the appropriate component. You will have to use your own "best" judgement to decide this. There will usually be some global policies and tactics that should be discussed here, but decisions about interfaces, algorithms, and/or data structures might be more appropriately discussed in the same (sub) section as its corresponding software component in one of these other sections.

* **Unit Testing**: Test individual Lambda functions, including file renaming and metadata extraction.
* **Integration Testing**: Test AWS Lambda and Box API interactions to ensure smooth communication.
* **Performance Testing**: Ensure the system can process up to 500 files per batch with minimal delay.
* **User Acceptance Testing**: Conduct testing with staff at the Santa Barbara Public Defender’s Office to confirm the system meets their needs.

**6. Detailed System Design**

Most components described in the System Architecture section will require a more detailed discussion. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

This is where Level 2 (or lower) DFD’s will go. If there are any additional detailed component diagrams, models, user flow diagrams or flowcharts they may be included here.

**6.x Name of Component (Module)**

**6.x.1 Responsibilities**

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements specification.

**6.x.2 Constraints**

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, post conditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

**6.x.3 Composition**

A description of the use and meaning of the subcomponents that are a part of this component.

**6.x.4 Uses/Interactions**

A description of this components collaborations with other components. What other components is this entity used by? What other components does this entity use (this would include any side-effects this entity might have on other parts of the system)? This concerns the method of interaction as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated subclasses, superclass’s, and metaclasses.

**6.x.5 Resources**

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

**6.x.6 Interface/Exports**

The set of services (classes, resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

Much of the information that appears in this section is not necessarily expected to be kept separate from the source code. In fact, much of the information can be gleaned from the source itself (especially if it is adequately commented). This section should not copy or reproduce information that can be easily obtained from reading the source code (this would be an unwanted and unnecessary duplication of effort and would be very difficult to keep up-to-date). It is recommended that most of this information be contained in the source (with appropriate comments for each component, subsystem, module, and subroutine). Hence, it is expected that this section will largely consist of references to or excerpts of annotated diagrams and source code.

**6.1 PDF Processing Component:**

**6.1.1 Responsibilities:**

* Receives PDF Files from Box event handler
* Extracts bates number from PDF contents
* Validates Bates number format
* Generates new file name based on Bates number
* Manage files renaming process

**6.1.2 Constraints:**

* Must process each PDF within a certain amount of time before Lambda function times out (Currently set to 1 minutes runtime)
* Memory usage limited to Lambda function (Limit of 512 MB)
* Only processes valid PDF formats
* Requires consistent Bates number format

**6.1.3 Composition:**

* PDF reader: Extracts text content
* Bates extractor: identifies and pulls bates numbers
* Name generator: Creates new file names
* Validation Engine: Ensures data consistency
* Error logger: Records processing issues through CloudWatch Logs

**6.1.4 Uses/Interactions:**

* Receives triggers from Box event handler
* Sends results to S3 bucket that is storing the renamed files
* Updates processing status to monitoring systems
* Interacts with AWS CloudWatch for logging

**6.1.5 Resources:**

* AWS lambda runtime environment
* PDF processing libraries such as PyMuPDF and Requests
* Memory allocation for PDF processing
* CPU resources for text extraction
* Temporary storage for processing (will be removed in future implementation)

**7. Detailed Lower level Component Design**

Other lower-level Classes, components, subcomponents, and assorted support files are to be described here. You should cover the reason that each class exists (i.e. its role in its package; for complex cases, refer to a detailed component view.) Use numbered subsections below (i.e. “7.1.3 The ABC Package”.) Note that there isn't necessarily a one-to-one correspondence between packages and components.

Below is a breakdown of the major components, their roles, and the details for each relevant class or function.

### 7.1 Lambda Function: BoxInputFunction

#### 7.1.1 Classification

* **Type:** AWS Lambda Function
* **Role:** Handles the HTTP API Gateway event triggered by a file upload to **Box.com**.

#### 7.1.2 Processing Narrative (PSPEC)

* Parses the event payload from the **Box API**, extracting file metadata such as **file\_id**, **folder\_id**, **access\_token,** and **file\_name**.
* Creates a custom payload and invokes the second Lambda function to process the file.

#### 7.1.3 Interface Description

* **Input:** Event payload from API Gateway (JSON format).
  + Key fields:
    - file\_id: Unique identifier for the uploaded file.
    - folder\_id: ID of the Box folder containing the file.
    - file\_name: Original name of the file.
* **Output:** Custom payload sent to the second Lambda function (JSON format).
  + Fields:
    - **file\_id**
    - **folder\_id**
    - **file\_name**
    - **disc\_number**
    - **pd\_case\_number**
    - **access\_token**

#### 7.1.4 Processing Detail

##### 7.1.4.1 Design Class Hierarchy

* Standalone Lambda function with no class inheritance.

##### 7.1.4.2 Restrictions/Limitations

* Dependent on the correct event structure from **Box API**.
* Assumes folder names follow the **pdcasenumber\_discnumber** naming convention.

##### 7.1.4.3 Performance Issues

* Minimal; processes lightweight metadata only.

##### 7.1.4.4 Design Constraints

* Must run within the AWS Lambda execution limits, 1 minute timeout and 128 MB memory. (can be adjusted if necessary up to 15-minute timeout and 10240 MB memory).

##### 7.1.4.5 Processing Detail for Each Operation

1. Parse the API Gateway payload.
2. Validate that all required fields are present.
3. Extract **disc\_number** and **pd\_case\_number** from **file\_name**.
4. Construct the custom payload.
5. Invoke the next Lambda function with the custom payload.

### 7.2 Lambda Function: DiscoveryBatesNamer

#### 7.2.1 Classification

* **Type:** AWS Lambda Function
* **Role:** Downloads the file from Box, extracts Bates numbers, validates their sequence, and renames the file.

#### 7.2.2 Processing Narrative (PSPEC)

* Retrieves the file from Box using its **file\_id**.
* Parses the file for Bates numbers using the **Pymudf** library.
* Validates the sequence of Bates numbers.
* Renames the file based on the extracted Bates numbers and stores it temporarily in an S3 bucket.

#### 7.2.3 Interface Description

* **Input:** Custom payload from the Event Handler Lambda (JSON format).
  + Key fields:
    - **file\_id**
    - **file\_name**
    - **disc\_number**
    - **pd\_case\_number**
    - **access\_token**
* **Output:** Renamed file stored in S3.

#### 7.2.4 Processing Detail

##### 7.2.4.1 Design Class Hierarchy

* Standalone Lambda function using helper methods for PDF parsing and validation.

##### 7.2.4.2 Restrictions/Limitations

* Requires PDFs to have machine-readable text.
* Fails if Bates numbers are non-sequential or missing.

##### 7.2.4.3 Performance Issues

* File download and parsing depend on the file size and network latency.

##### 7.2.4.4 Design Constraints

* Must process files within AWS Lambda’s runtime and memory constraints.
* Limited to files supported by **Pymudf**.

##### 7.2.4.5 Processing Detail for Each Operation

1. Retrieve the file using **Boxdev** API and **file\_id**.
2. Parse the file using **Pymudf** to extract Bates numbers.
3. Validate Bates number sequence.
4. Rename the file to include:
   * Bates range.
   * Disc number.
   * PD case number.
5. Upload the renamed file to an S3 bucket.

**7.x Name of Class or File**

**Not fully developed yet**

**7.x.1 Classification**

The kind of component, such as a subsystem, class, package, function, file, etc.

**7.x.2 Processing Narrative (PSPEC)**

A process specification (PSPEC) can be used to specify the processing details

**7.x.3 Interface Description**

**7.x.4 Processing Detail**

**7.x.4.1 Design Class Hierarchy**

Class inheritance: parent or child classes.

**7.x.4.2 Restrictions/Limitations**

**7.x.4.3 Performance Issues**

**7.x.4.4 Design Constraints**

**7.x.4.5 Processing Detail For Each Operation**

**~~8. Database Design~~**

N/A

**9. User Interface**

The user interface is the application, from the point of view of the users. Do your classes and their interactions (the logical and process views) impose restrictions on the user interface? Would removing some of these restrictions improve the user interface? Use some form of user interface flow model to provide an overview of the UI steps and flows. Don't go into too much refinement. You should include screen shots or wireframe layouts of significant pages or dialog elements. Make sure to indicate which of the system level modules or components that each of these user interface elements is interacting with.

**9.1 Overview of User Interface**

Describe the functionality of the system from the user’s perspective. Explain how the user

will be able to use your system to complete all the expected features and the feedback

Information that will be displayed for the user. This is an overview of the UI and its use. The user manual will contain extensive detail about the actual use of the software.

**9.2 Screen Frameworks or Images**

These can be mockups or actual screenshots of the various UI screens and popups.

**9.3 User Interface Flow Model**

A discussion of screen objects and actions associated with those objects. This should include a flow diagram of the navigation between different pages.

**10. Requirements Validation and Verification**

Create a table that lists each of the requirements that were specified in the SRS document for this software.

For each entry in the table list which of the Component Modules and if appropriate which UI elements and/or low level components satisfies that requirement.

For each entry describe the method for testing that the requirement has been met.

**11. Glossary**

An ordered list of defined terms and concepts used throughout the document. Provide definitions for any relevant terms, acronyms, and abbreviations that are necessary to understand the SDD document.  This information may be listed here or in a completely separate document.  If the information is not directly listed in this section provide a note that specifies where the information can be found.

Key Definitions:

* Bates Number: Commonly used in the discovery phase of legal cases, bates number are a sequential identifier assigned to each page of a legal document for tracking and referencing
* Box.com: A cloud-based file storage platform used for document organization and collaboration
* AWS Lambda: A serverless computing service provided by Amazon Web Services that executes code in response to events, eliminating for the need to manage and maintain servers
* Metadata: Structured data that provides information about other data that are attached to the file. In this context, metadata can include PD Case Number and Disc Number for files that were uploaded
* Discovery Documents: Documents that are exchanged between parties during the legal discovery process in legal cases

Acronyms and Abbreviations:

* SRS: Software Requirements Specification
* AWS: Amazon Web Services
* API: Application Programming Interface
* PDF: Portable Document Format

**12. References**

<List any other documents or Web addresses to which this SDD refers. These may include other SDD or SRS documents, user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

Brad Appleton <brad@bradapp.net>  <http://www.bradapp.net>

<https://www.cs.purdue.edu/homes/cs307/ExampleDocs/DesignTemplate_Fall08.doc>

<https://github.com/box/box-python-sdk-gen/blob/main/docs/folders.md#get-folder-information>

<https://developer.box.com/reference/>

<https://developer.box.com/guides/skills/handle/payload/>

<https://pymupdf.readthedocs.io/en/latest/how-to-open-a-file.html#opening-remote-files>

<https://docs.aws.amazon.com/lambda/latest/dg/python-layers.html>

<https://docs.aws.amazon.com/lambda/latest/dg/packaging-layers.html>

<https://docs.aws.amazon.com/lambda/latest/dg/creating-deleting-layers.html>

<https://docs.aws.amazon.com/lambda/latest/dg/adding-layers.html>

<https://developer.box.com/reference/get-files-id/>