Development Plan
ProgName

Team #, Team Name
Student 1 name
Student 2 name
Student 3 name
Student 4 name

Table 1: Revision History

Date	**Developer(s)**	**Change**
Date1	Name(s)	Description of changes
Date2	Name(s)	Description of changes
...	...	...

[Put your introductory blurb here. Often the blurb is a brief roadmap of what is contained in the report. —SS]

[Additional information on the development plan can be found in the lecture slides. —SS]

1 Confidential Information?

[State whether your project has confidential information from industry, or not. If there is confidential information, point to the agreement you have in place. —SS]

[For most teams this section will just state that there is no confidential information to protect. —SS]

2 IP to Protect

[State whether there is IP to protect. If there is, point to the agreement. All students who are working on a project that requires an IP agreement are also required to sign the "Intellectual Property Guide Acknowledgement." —SS]

3 Copyright License

[What copyright license is your team adopting. Point to the license in your repo. —SS]

4 Team Meeting Plan

[How often will you meet? where? —SS]

[If the meeting is a physical location (not virtual), out of an abundance of caution for safety reasons you shouldn't put the location online —SS]

[How often will you meet with your industry advisor? when? where? —SS]

[Will meetings be virtual? At least some meetings should likely be in-person. —SS]

[How will the meetings be structured? There should be a chair for all meetings. There should be an agenda for all meetings. —SS]

5 Team Communication Plan

[Issues on GitHub should be part of your communication plan. —SS]

6 Team Member Roles

[You should identify the types of roles you anticipate, like notetaker, leader, meeting chair, reviewer. Assigning specific people to those roles is not necessary at this stage. In a student team the role of the individuals will likely change throughout the year. —SS]

7 Workflow Plan

- **Version Control & Workflow:**
  - We will be making use of GitHub with a feature-branch workflow. Each task/issue will get its own branch.
  - All code changes require a pull request and at least one approval from a reviewer before a merge to the master branch.
    - All PRs must follow a standardized naming convention:

    Issue # Name of feature
    - All PRs must also contain a standardized description outlining the details of the change:
      - What?
      - Why?
      - How?
      - Testing?
      - Screenshots (optional)
      - Anything Else
      - Refer to https://arc.net/l/quote/touaacbp
  - Commit messages must contain a short description of the change being added to the feature branch.
- **Issues & Project Tracking Boards:**
  - We will be making use of a Kanban board on GitHub to track our project's progress.
  - This Kanban board will contain Issues and will overall be split up into 5 separate categories:
    - To-Do
    - In Progress
    - Pending Review
    - Reviewed
    - Done
  - Issues will be used with templates for things like bugs and tasks.
  - We will also be making use of Labels:

        * Bug, enhancement, documentation
        * Easy
        * Needs design (design discussion required)
        * Due By ...

- **Milestones:**
  - We will be creating milestones in GitHub corresponding to all major deliverables:
    * POC
    * Rev0
    * Rev1
    * Final
  - Each issue will be linked to a milestone.
- **Continuous Integration / Continuous Deployment:**
  - We will be making use of GitHub Actions to trigger a CI pipeline.
  - Each push and pull request will trigger:
    * Build automation:
      · Utilizing makefiles to compile all modules
      · **(ALREADY COMPLETED)** automatically build .tex files into PDFs
    * Some sort of linter for style compliance
    * Unit test suite
    * Deployment

8 Project Decomposition and Scheduling

As mentioned previously, we will be making use of a Kanban board through GitHub Projects to manage all of our tasks. Each task will be associated with an Issue and linked to milestones corresponding to all major deliverables (POC, Rev0, Rev1, Final). We plan on dividing tasks based on knowledge built through personal experiences as well as interest; however, each task will be reviewed cross-functionally to ensure shared knowledge.
Each major milestone will contain deliverables within them.

Major Milestones

1. Proof of Concept (POC)
2. Revision 0
3. Revision 1
4. Final

Deliverables from Course Outline

- Team Formed, Project Selected
- Problem Statement, POC Plan, Development Plan
- Requirements Doc and Hazard Analysis Revision 0
- V&V Plan Revision 0
- Design Document Revision -1
- Proof of Concept Demonstration
- Design Document Revision 0
- Revision 0 Demonstration
- V&V Report and Extras Revision 0
- Final Demonstration (Revision 1)
- EXPO Demonstration
- Final Documentation (Revision 1)

Task Decomposition

We plan on breaking down all tasks into features that can be completed in a few hours of work rather than days.

Some examples would be:

- Design database schema draft
- Implement orientation correction function for fragments
- Write test plan for OCR module for Sanskrit Input

We want to make sure that the tasks (Issues on GitHub) we create are specific and not open to much ambiguity. We also want to ensure that they are easy to test and estimated in hours. During our weekly meetings, we will be identifying large chunks of work and decomposing them into digestible chunks. This will be completed through discussion and trying to gain an understanding of the task at hand.

Assigning Work

Initially, task assignment will be based on interest and strengths. However, as we move on, we plan on rotating tasks to prevent overspecialization and to ensure everyone has a chance to explore areas of interest. Each task will have both an assignee and a reviewer.

Additionally, each team member will be expected to commit approximately 10 hours per week of work dedicated to this project. This includes meetings with peers or professors, tutorial time, and actual development time.

9 Proof of Concept Demonstration Plan

9.1 Main Risks for Project Success

The most significant risks for our Buddhist manuscript fragment reconstruction platform are:

- **Inconsistent OCR Performance**: OCR accuracy varies dramatically across different manuscript conditions, script styles, and image quality. Ancient Buddhist texts often contain deteriorated characters, non-standard orthography, and multiple script variations that challenge standard OCR engines.
- **Lack of Positive Training Examples**: The absence of confirmed fragment matches creates a training data challenge. Without verified examples of fragments that belong together, we will need to explore unsupervised learning approaches and develop alternative validation strategies for our matching algorithms.

These are risks because they directly impact the core functionality of our system - the ability to accurately identify which fragments belong together.

9.2 Implementation Challenges

The most challenging aspects of implementation will be:

- **OCR Integration and Consistency**: Implementing OCR engines that can handle Sanskrit manuscript variations while maintaining consistent output across different image conditions and script styles.
- **AI Model Training Without Ground Truth**: Creating and training machine learning models for fragment similarity matching without access to verified positive examples of matched fragments. The final model will need to combine multiple features (OCR text, edge patterns, damage signatures) into confidence scores, which presents challenges for training validation. We plan to explore unsupervised learning approaches to address this challenge.
- **Interactive Canvas**: Building a responsive, intuitive drag-and-drop interface that can handle potentially hundreds of fragment images while maintaining performance.
- **Multi-scale Matching**: Implementing algorithms that can suggest matches at different confidence levels.

9.3 Testing Difficulties

Testing will present unique challenges because:

- **Ground Truth**: We need access to known fragment matches from Buddhist Studies scholars to validate our algorithms, which may be limited initially.
- **Subjective Validation**: Fragment matching often involves scholarly interpretation, making automated testing metrics challenging to define.
- **Performance Testing**: Testing the system with large batches of high-resolution fragment images will require substantial computational resources.
- **User Experience Testing**: The interface must be intuitive for scholars with varying technical backgrounds, requiring extensive usability testing.

9.4 Library and Technology Risks

9.5 Hardware and Infrastructure Concerns

- **Processing Power**: Computer vision and AI algorithms may require substantial CPU/GPU resources for model training.

9.6 Demonstration Plan

To address these risks and demonstrate feasibility, our POC will focus on proving that the core technical challenges can be overcome:

- **OCR Functionality Demonstration**: Show OCR engines successfully extracting text from sample manuscript fragments with varying quality levels. We will demonstrate text extraction from both clear and degraded fragments to validate OCR consistency approaches.
- **Image Segmentation and Normalization**: Demonstrate fragment segmentation algorithms that can isolate individual fragments from composite images and normalize them for consistent analysis (rotation correction, standardization).
- **Basic Feature Extraction**: Show extraction of key features from fragments including edges, text content, and damage patterns that could be used for matching.
- **Prototype Similarity Metrics**: Implement basic similarity algorithms that compare normalized fragments and provide confidence scores, acknowledging the limitation that it will be difficult to validate accuracy without ground truth data.
- **Interactive Workspace**: Create a basic interface where users can upload fragments, view OCR results, and see preliminary similarity suggestions.

This demonstration will prove the technical feasibility of the core components. The POC focuses on demonstrating that our technical approach can extract and compare the right types of features for fragment matching. While the absence of ground truth data presents challenges, we will explore unsupervised learning techniques and develop validation approaches that can work with the available data.

10 Expected Technology

For our project, we will use a combination of programming languages, external libraries, dev tools and infrastructure for fragment processing, matching and analysis. While the exact implementation has not been finalized yet and may evolve, the following technologies are expected to be used:

Programming Languages

- Python – primary language for image preprocessing, data pipelines and model integration
- C++ (if required) – will be utilized for performance-critical modules
- JavaScript / TypeScript – for frontend development

Libraries & Frameworks

- OpenCV for image preprocessing, orientation correction, and edge detection
- PyTorch / TensorFlow for machine learning models for script identification and OCR
- scikit-learn for probabilistic fragment-matching algorithms
- SQLAlchemy for database interaction (PostgreSQL or SQLite)
- React for frontend framework to build the researcher-facing interface
- Tailwind CSS for frontend styling libraries
- Flask, FastAPI or Django for backend API layer to connect frontend and ML/database modules

Pre-trained Models

- We will be exploring OCR models for Sanskrit
- We plan on training a lightweight model in the case that pre-trained models are not sufficient

IDE & Dev Tools

- VSCode as the primary development environment
- LaTeX/Markdown for documentation and deliverables
- Doxygen for auto-generating API documentation
- Postman to test backend endpoints

Testing and Code Quality Tools

- Pytest for backend unit testing
  - Can utilize coverage.py to generate coverage reports
- Combination of MyPy and Ruff for Python linting
- Jest for frontend testing
- ESLint and Prettier for frontend linting
- Google C++ Style Guide if we develop modules in C++
- Valgrind for C++ performance if we develop modules in C++

Build & Automation Tools

- Makefiles for automated builds of modules and LaTeX documents
- GitHub Actions as our CI/CD pipeline:
  - Linting and style checks for frontend and backend
  - Unit and integration tests
  - Docker image builds
  - PDF generation from LaTeX docs

Deployment

- Docker for containerized environments
- Git and GitHub projects for version control and project management

11 Coding Standard

The most important thing that we will be striving for is consistency throughout our project. This can only be possible by highlighting a clear set of rules and standards for development. These rules and standards have been listed below.

General Rules

- Code reviews: 1+ review approval required before merge
  - Approval depends on style as well as correctness of code
- Commits and PRs must follow the standardized template established in the workflow plan
- No sensitive data will be committed to the repository
  - Credentials
  - Datasets with restrictions (fragment images)
- Decisions regarding style, tech stack and others may evolve over time but must be documented in this document (Development Plan)

Python Rules and Standards

- Follow PEP8 guides for formatting and structure
- Use Ruff for linting and rule enforcement
- MyPy for typechecking
- Write NumPy/SciPy-style docstrings for public functions, including data shapes and types
- Implement tests with Pytest and measure coverage with coverage.py
  - Aim to reach 80% coverage

JavaScript/TypeScript Rules and Standards

- Follow Google JavaScript Style Guide with React/JSX convention
- Use ESLint for static analysis and Prettier for consistent formatting
- Standardize styling with Tailwind CSS
- Test frontend code with Jest and aim to reach 80% coverage

API and Documentation Standards

- Define request and response schemas using `pydantic` models and TypeScript interfaces
- Use proper HTTP status codes as well as versioned endpoints
- Use consistent routing for endpoints
- Generate API documentation

CI Enforcement

- Run Ruff, MyPy and Pytest for backend code
- Run ESLint, Prettier and Jest for frontend code
- Block merges on failed lint, typecheck or test results

Appendix — Reflection

[Not required for CAS 741 —SS]

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. Why is it important to create a development plan prior to starting the project?

2. In your opinion, what are the advantages and disadvantages of using CI/CD?

3. What disagreements did your group have in this deliverable, if any, and how did you resolve them?

Appendix — Team Charter

[borrows from University of Portland Team Charter —SS]

External Goals

[What are your team's external goals for this project? These are not the goals related to the functionality or quality fo the project. These are the goals on what the team wishes to achieve with the project. Potential goals are to win a prize at the Capstone EXPO, or to have something to talk about in interviews, or to get an A+, etc. —SS]

Attendance

Expectations

[What are your team's expectations regarding meeting attendance (being on time, leaving early, missing meetings, etc.)? —SS]

Acceptable Excuse

[What constitutes an acceptable excuse for missing a meeting or a deadline? What types of excuses will not be considered acceptable? —SS]

In Case of Emergency

[What process will team members follow if they have an emergency and cannot attend a team meeting or complete their individual work promised for a team deliverable? —SS]

Accountability and Teamwork

Quality

[What are your team's expectations regarding the quality of team members' preparation for team meetings and the quality of the deliverables that members bring to the team? —SS]

Attitude

[What are your team's expectations regarding team members' ideas, interactions with the team, cooperation, attitudes, and anything else regarding team member contributions? Do you want to introduce a code of conduct? Do you want a conflict resolution plan? Can adopt existing codes of conduct. —SS]

Stay on Track

[What methods will be used to keep the team on track? How will your team ensure that members contribute as expected to the team and that the team performs as expected? How will your team reward members who do well and manage members whose performance is below expectations? What are the consequences for someone not contributing their fair share? —SS]

[You may wish to use the project management metrics collected for the TA and instructor for this. —SS]

[You can set target metrics for attendance, commits, etc. What are the consequences if someone doesn't hit their targets? Do they need to bring the coffee to the next team meeting? Does the team need to make an appointment with their TA, or the instructor? Are there incentives for reaching targets early? —SS]

Team Building

[How will you build team cohesion (fun time, group rituals, etc.)? —SS]

Decision Making

[How will you make decisions in your group? Consensus? Vote? How will you handle disagreements? —SS]