5. SOFTWARE PROCESS MODEL

5.1 Methodology

The Study in Woods project follows an **Agile** software development methodology, implementing a hybrid approach combining Scrum for sprint management and Kanban for continuous feature flow. This methodology was chosen to enable rapid iteration and the ability to adapt to changing requirements during the academic project timeline.

Development spans 12 phases over 6 months (June - December 2024), with two-week sprints targeting 20-25 story points each. Phases overlap with continuous integration and testing running throughout.

5.2 Sprint Structure

Activity	Frequency	Duration	Deliverables
Sprint Planning	Every 2 weeks	2 hours	Sprint backlog, Story estimates
Sprint Review	Every 2 weeks	1 hour	Working software demo
Backlog Refinement	Weekly	1 hour	Refined user stories

5.3 Development Phases

Phase	Weeks	Key Deliverables	LOC
1: Project Setup	1-2	Monorepo structure, Docker Compose, initial DB schema	~3,500
2: Authentication	3-4	JWT auth, login/register, password reset	
3: Academic Hierarchy	5-6	University/Course/Semester/Subject CRUD	
4: Document Upload	7-8	DigitalOcean Spaces integration, multipart upload	~4,200
5: AI Knowledge Base	9-10	GradientAI KB integration, document indexing	
6: Syllabus Extraction	11-12	Llama 3.3 70B extraction, structured JSON output	
7: Chat Sessions	13-14	Session CRUD, message history	~3,800
8: AI Chat	15-16	SSE streaming, KB-context responses	
9: Citations	17-18	Source document references in responses	
10: PYQ Extraction	19-20	Question extraction from exam papers	~3,000
11: Analytics	21-22	Usage tracking, dashboard charts	
12: Admin Panel	23-24	User management, system settings, deployment	

5.4 CI/CD Pipeline

Automated via GitHub Actions on every push:

- **Linting:** golangci-lint (Go), ESLint (TypeScript)
- **Testing:** Unit tests (70% coverage required), integration tests via Docker Compose

- **Build:** Multi-stage Docker images
- **Deploy:** Zero-downtime deployment to DigitalOcean Droplet on main branch merge

5.5 Version Control

Git Flow branching with Conventional Commits:

- **main:** Production-ready code
- **develop:** Integration branch
- **feature/*:** Individual features
- Semantic versioning (MAJOR.MINOR.PATCH) for releases