ACADEMIC ANNEX

Spec Driven Development

An AI-Native Software Development Program

Beyond vibe-coding, building reliable systems.

A day-wise academic plan for mixed-skill college cohorts: fundamentals of SDD, BDD and TDD; guided AI-native engineering practice; and a 12-hour evidence-driven hackathon.

06	36	08
DAYS	TOTAL HOURS	PLANNED LABS

REDRUFF • SAI PAVAN VELIDANDLA

sdd.redruff.in

Curriculum proposal / November 2026

PROGRAM ARCHITECTURE

Accessible starting point. Shared finish line.

Who the program is for

B.Tech, BCA, MCA and related computing students who have mixed programming skills. Students without programming fundamentals complete optional preparatory Python, Git and terminal practice before Day 1.

Structure and teaching rhythm

Days 1–4: 24 hours of instruction, demonstrations, guided practice and feedback, including eight planned labs.
Days 5–6: 12-hour team hackathon with explicit specification approval, implementation, change challenge, tests and jury demonstration.

The three support lanes

A single project and common core rubric prevent an uneven experience. Foundation students use scaffolds; Builder students implement independently; Advanced students tackle extensions. All must demonstrate essential individual competencies.

Shared student competencies

Write a verifiable requirement; explain BDD acceptance behavior; create or interpret a TDD test; modify code with an AI assistant; inspect a diff; run tests; handle a change request; and defend a working software feature.

THE CORE LOOP

Understand → Specify → Plan → Build → Test → Review → Adapt → Deliver

The loop is iterative. Specifications are maintained when expectations change, not treated as static Waterfall documents.

DAY 01

Think before you code

Learners recognize the difference between prompt-led experimentation and engineering a feature against explicit requirements.

TOPIC COVERAGE

AI-native engineering

Vibe coding, coding agents, unreliable outputs, missing requirements, review and human responsibility.

SDD + BDD + TDD

Specifications guide intent; scenarios describe behavior; tests constrain and verify implementation.

Problem discovery

Users, problem scope, stakeholders, non-goals, user stories and constraints.

Acceptance thinking

Given / When / Then; negative behavior; failure cases; what “done” actually means.

LAB 01 • SAME TASK, TWO APPROACHES

Build a small feature with an informal prompt and compare it with specification-first work. Record omissions and correction effort.

Evidence: Comparison notes and two implementation attempts.

LAB 02 • TURN A PROBLEM INTO A SPEC

For a College Event Registration System, write user stories, scope and BDD acceptance scenarios.

Evidence: Problem brief, five acceptance criteria and three scenarios.

DIFFERENTIATED PRACTICE

FOUNDATION	Complete a guided requirement template.
BUILDER	Draft a specification independently.
ADVANCED	Resolve contradictory rules and non-functional requirements.

DAILY READINESS GATE

Explain one acceptance criterion and how its correctness can be observed.

DAY 02

Make intent actionable

Convert a set of requirements into a reviewable specification, project context, technical plan and task list.

TOPIC COVERAGE

SDD workflow

Specify → clarify → plan → tasks → implement → verify; GitHub Spec Kit fundamentals.

Precise requirements

EARS-style behavior statements, assumptions, input validation and edge cases.

Context engineering

Project rules, AGENTS.md, bounded instructions, context relevance and avoiding document bloat.

Engineering plan

Simple system architecture, SQLite data model, dependencies, priorities and traceability.

LAB 03 • SPEC KIT PROJECT

Set up the starter repository; create project rules; refine the specification with an agent.

Evidence: Version-controlled specification and rules.

LAB 04 • DESIGN AND TASK PLAN

Create and review architecture, data model, implementation tasks and acceptance checklist.

Evidence: Reviewed plan and task-to-requirement map.

DIFFERENTIATED PRACTICE

FOUNDATION	Review AI-produced plans with a checklist.
BUILDER	Generate and edit the design and task list.
ADVANCED	Find dependencies, risks and architecture trade-offs.

DAILY READINESS GATE

Trace a requirement to a planned implementation task and test.

DAY 03

Build, test, verify

Implement a vertical slice with an AI coding assistant, use acceptance scenarios and practice the TDD feedback loop.

TOPIC COVERAGE

Controlled agent execution

Small tasks, readable diffs, meaningful commits, human review and dependency checks.

BDD into testing

Translate Given / When / Then into executable or manually verified acceptance scenarios.

TDD fundamentals

Red → Green → Refactor, pytest, unit tests, negative cases and basic integration.

Trust through verification

Read failures, fix generated defects, guard against regressions, introduce invariants.

LAB 05 • SPEC-TO-CODE

Build the registration workflow from an approved specification and review code changes.

Evidence: A working user journey and Git commits.

LAB 06 • TEST-FIRST DEFECT CHALLENGE

Write a failing rule test, implement it, then find a deliberately hidden defect.

Evidence: Test log, correction and requirement-to-test mapping.

DIFFERENTIATED PRACTICE

FOUNDATION	Run tests and implement a guided small change.
BUILDER	Build and test a feature independently.
ADVANCED	Use parameterized tests and stronger edge cases.

DAILY READINESS GATE

Demonstrate a code change and explain its test evidence.

DAY 04

Make changes safely

Practice SDD in an existing codebase, updating requirements and tests while protecting previously verified behavior.

TOPIC COVERAGE

Brownfield development

Read an unfamiliar app, locate affected behavior, extract intent and scope a change.

Change specifications

Incremental specs, change impact, brief OpenSpec demonstration and constrained agent tasks.

Engineering quality

Input validation, secrets, privacy, security basics, regression and review checklists.

Release readiness

Integration verification, reproducible demos, known limitations and change traceability.

LAB 07 • CHANGE AN EXISTING APP

Receive a feature request against an imperfect existing application; update spec, code and tests.

Evidence: Change specification and regression evidence.

LAB 08 • RELEASE REVIEW

Review another implementation; identify a risk and fix or document it before a demo.

Evidence: Review checklist and verified fix / limitation.

DIFFERENTIATED PRACTICE

FOUNDATION	Modify one well-contained existing function.
BUILDER	Implement a change and add regression tests.
ADVANCED	Conduct impact analysis and peer review.

DAILY READINESS GATE

Demonstrate preserved old behavior plus the new requirement.

DAYS 05 AND 06

The build-and-defend hackathon.

The hackathon tests engineering judgment as much as coding speed. Teams work on bounded authentic problems using synthetic data.

DAY 05 / DISCOVER → SPECIFY → BUILD

- Form balanced teams and select an approved problem brief.
- Define users, MVP scope, exclusions and observable success criteria.
- Create acceptance scenarios and obtain mentor specification approval.
- Plan architecture, assign tasks, build core user journey and begin tests.

DAY 05 GATE

Approved specification + functioning core workflow + initial test evidence.

DAY 06 / VERIFY → ADAPT → DELIVER

- Complete essential user journeys; integrate components; prioritize defects.
- Receive a surprise requirements change and assess impact before coding.
- Update the specification and tests; implement safely or document deferral.
- Run acceptance and regression checks, prepare repository, deliver live demo and Q&A.;

FINAL GATE

Working app + versioned spec + tests + change evidence + individual explanation.

ASSESSMENT MODEL

Judge the engineering. Not just the screen.

The proposed certification approach combines 70% team project evaluation with 30% independent competency assessment. Set the individual passing threshold in the delivery agreement.

TEAM HACKATHON RUBRIC

Problem / specification	20%
Design / task planning	15%
Working features / correctness	25%
BDD / TDD / verification	20%
Change / code quality	10%
Demo / collaboration	10%

MANDATORY INDIVIDUAL DEMONSTRATIONS

- Explain an acceptance criterion and its associated test.
- Inspect an agent-created diff and describe what changed.
- Modify a small function or business rule, then rerun the relevant tests.
- State a known limitation and the risk created by a new requirement.

Fairness: Advanced extensions cannot replace missing core competencies. Certificates depend on demonstrated work, not attendance alone.

DELIVERY CHECKLIST

Make the program work on the ground.

RECOMMENDED COHORT AND SUPPORT

- Starting model: 30–36 students, balanced teams of 3–4, one lead instructor and technical mentor support agreed by cohort size.
- Pre-course readiness check: working laptop, editor, Git, Python, GitHub, AI-agent access and basic code comprehension.
- Common stack: Python + Streamlit + SQLite + pytest + GitHub; primary SDD workflow: GitHub Spec Kit.
- Internet, electrical power, projector, local demo option, synthetic sample data and fallback plans for AI usage limits.
- RedRuff provides training assets, mentor guidance, sample repositories, evaluation materials and a cohort outcomes summary.

SOURCE NOTES AND PROVENANCE

Industry context

Stack Overflow Developer Survey 2026, AI section; 2025 DORA AI-assisted software development report.

Framework workflow

GitHub Spec Kit documentation; tool details to be checked against the installed release before delivery.

Facilitator credentials

Sai Pavan Velidandla, professional-profile.pdf in supplied portfolio; some totals are self-reported.

Engagement examples

Organizer post (Geethanjali), student-team social post (ANITS), and LBRCE institutional event report.

PLAN A CAMPUS COHORT / pavan@redruff.in | +91 95024 44362

sdd.redruff.in pavan.redruff.in