International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

Agile Scrum for Dual-Artefact Climate Technology Software Development: Methodology, Risk Management, and Retrospective Evaluation

Chinoso Job[1], Onwe, Festus Chijioke[2]
[1]*University of Greater Manchester, United Kingdom*
[2]*Information Technology Department, University of Port Harcourt, Rivers State, Nigeria.*

***Abstract*—Agile software development methodologies have demonstrated broad applicability across industries beyond traditional software engineering, yet their application to climate technology systems with dual interdependent software artefacts remains understudied in the empirical literature. This paper presents a comprehensive account of the application of the Scrum framework to the concurrent development of two integrated software artefacts for BlueWave Solutions—a climatetech company deploying solar-powered IoT desalination buoys: (i) a Flask-based RESTful API for ingesting and serving realtime environmental telemetry data; and (ii) a Django-based eCommerce platform for desalination unit sales, subscription management, and environmental data access. The development was executed across three Sprints by a four-member MSc Software Engineering team. The paper details the requirements engineering process (User Stories, MoSCoW prioritisation, Planning Poker estimation), Scrum ceremony execution, Definition of Done (DoD) enforcement, INVEST criteria application to backlog refinement, and an integrated risk management strategy. Testing strategy encompasses Test-Driven Development (TDD), JUnit/Pytest unit testing, integration testing with Postman, and security testing of JWT authentication. Sprint velocity tracking, retrospective insights, and a structured evaluation against original requirements demonstrate that Agile Scrum enabled adaptive, quality-assured delivery of both artefacts within a 12-week timeboxed schedule. The paper contributes empirical evidence for the effectiveness of Scrum in multi-component, climate-oriented software development contexts.**

***Index Terms*—Agile, Scrum, software development, IoT, Flask API, Django, MoSCoW, TDD, risk management, climate technology, JWT, user stories**

I. Introduction

The Agile Manifesto [1] articulates a set of values prioritising individuals and interactions, working software, customer collaboration, and responding to change—principles particularly relevant to climate technology projects where requirements evolve as environmental data models are refined and stakeholder understanding deepens. Scrum [2], the most widely adopted Agile framework, operationalises these values through time-boxed Sprints, self-organising teams, and continuous feedback mechanisms.

BlueWave Solutions is a climate-tech startup addressing freshwater scarcity through a fleet of solar-powered IoT buoys that perform micro-desalination and continuously monitor environmental parameters (salinity, pH, pollutants). To support this mission, two integrated software artefacts were required:

1) A Flask API to ingest, store, and serve IoT telemetry data with JWT-secured access;
2) A Django eCommerce Platform to sell desalination units, display environmental metrics, and manage data subscriptions.

These artefacts present an interesting Agile test case: they are architecturally distinct (API vs. web application), developed in parallel with shared authentication infrastructure, and serve different user populations (researchers via API, consumers via web). Conforming to the observations of Conformito et al. [8] that Agile's applicability extends beyond software to technically complex multi-component systems, this project provides a concrete empirical case of cross-artefact Agile coordination.

This paper contributes: (i) a detailed account of Scrum ceremony execution for dual-artefact development; (ii) empirical evidence on MoSCoW and Planning Poker effectiveness in technical estimation; (iii) an integrated risk management strategy for multi-component Agile projects; and (iv) a retrospective evaluation of Sprint velocity and delivery against requirements.

II. Background

A. Agile and Scrum in Software Engineering

Agile development practices have been shown to improve software quality, team productivity, and customer satisfaction compared to traditional plan-driven approaches, particularly in environments with uncertain or evolving requirements [7]. Scrum specifically provides a lightweight framework with three roles (Product Owner, Scrum Master, Development Team), five events (Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective, Sprint itself), and three artefacts (Product Backlog, Sprint Backlog, Increment) [2].

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

B. User Stories and Requirements Engineering

Cohn [3] defines User Stories as the primary requirements capture mechanism in Agile, formatted as: "As a [role], I want [feature] so that [benefit]." Wake [5] introduces the INVEST criteria—Independent, Negotiable, Valuable, Estimable, Small, Testable—as quality criteria for User Stories. MoSCoW prioritisation (Must Have, Should Have, Could Have, Won't Have) enables rational backlog ordering under time constraints [6]. Planning Poker, using Fibonacci sequence story point cards, provides a structured approach to collaborative estimation that surfaces assumption mismatches among team members [4].

C. Test-Driven Development

Beck [9] defines TDD as a development discipline where test code is written before production code, cycling through Red (failing test) → Green (passing implementation) → Refactor phases. TDD promotes design clarity, reduces defect rates, and provides executable documentation of system behaviour.

III. Project Context And Team Structure

A. Development Team

The Scrum Team comprised four MSc Software Engineering students with differentiated technical specialisations:

- Ifesinachi Nwankwo — Backend development lead; Flask API, database design, JWT implementation, Django ORM;
- Peter Ikweagwa — UI/UX design, Django frontend (dashboard.html, metrics app), API estimation;
- Mbam Bridget — DevOps, CRUD operations, SQLite3 development, TDD contribution;
- Nweze Isaiah — Flask API setup, JWT security, Swagger specification, Django accounts app.

The module tutor served as Product Owner, providing Sprint Review feedback and requirement clarification.

B. Technology Stack

TABLE I
BLUEWAVE SOLUTIONS TECHNOLOGY STACK

Component	Technology
Flask API	Python 3.11, Flask, SQLAlchemy, JWT, Swagger/OpenAPI
Django Platform	Python 3.11, Django 4.2, Bootstrap 5, ORM
Database (Dev)	SQLite3 (local development)
Database (Prod)	MySQL Community Server
Testing	Pytest, Django TestCase, Postman
Collaboration	Trello (backlog), GitHub (version control)
Deployment	AWS (EC2 + RDS)
Documentation	Swagger UI, GitHub Wiki

IV. Requirements Engineering

A. User Story Specification

Fig. 1 illustrates the User Story format applied across both artefacts, with DoD and acceptance criteria for US-13 (JWT Authentication).

US-13: JWT Authentication

As a Product Owner, *I want* the API secured with JWT authentication *so that* only authorised users can access telemetry data.

DoD: JWT generation and validation implemented; unit tests passing; Postman test demonstrates 401 on invalid token.

Acceptance Criteria: (1) Valid JWT returns 200 with data; (2) Missing JWT returns 401 Unauthorized; (3) Expired JWT returns 403 Forbidden.

Priority: Should Have (MoSCoW)Estimate: 8 Story Points

Fig. 1. User Story specification template (US-13 example) with DoD and acceptance criteria.

B. MoSCoW Prioritisation

Fig. 2 illustrates the distribution of 24 User Stories across MoSCoW priority categories for both artefacts.

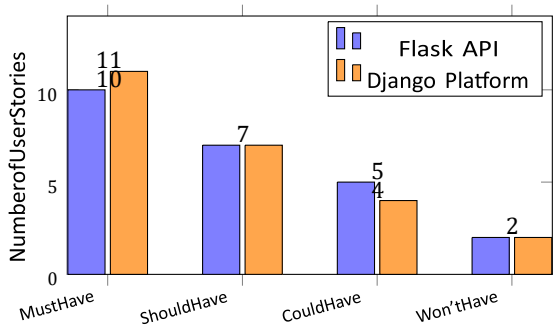

Fig. 2. MoSCoW priority distribution across Flask API and Django Platform User Stories.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

C. Planning Poker Estimation

Planning Poker sessions were conducted at Sprint Planning using a modified Fibonacci sequence (1, 2, 3, 5, 8, 13) via an online Planning Poker tool. When team estimates diverged by more than one sequence point, structured discussion was facilitated by the Scrum Master to surface assumption differences. Table II shows the story point velocity achieved across Sprints.

TABLE II
SPRINT VELOCITY AND KEY DELIVERABLES

Sprint	Focus	Planned SP	Completed SP
Sprint 1	Flask API (core)	42	39
Sprint 2	Django Platform	44	42
Sprint 3	Refinement & Secu-rity	28	28
Total		114	109

V. Scrum Ceremonies And Sprint Execution

A. Sprint Planning

Sprint Planning allocated User Stories from the prioritised Product Backlog to each Sprint's Sprint Backlog, with story point capacity calculated from team velocity. Cross-artefact dependencies (JWT token must be available in Flask before Django integration) were explicitly mapped to Sprint dependencies, ensuring Sprint 1 Flask API JWT output was available as Sprint 2 input.

B. Daily Scrum

Daily Scrums (15-minute timebox) were conducted asynchronously via a shared Slack channel, with each member posting: (i) completed yesterday; (ii) planned today; and (iii) blockers. Blockers were escalated to a Zoom call with the Scrum Master within 24 hours.

C. Sprint Review and Retrospective

Sprint Reviews demonstrated working software increments to the Product Owner (module tutor). Sprint Retrospectives used the Start/Stop/Continue format documented on Mural digital boards. Key retrospective actions included: (i) adopting GitHub Pull Request templates to enforce peer code review; and (ii) standardising API error response formats to reduce integration friction between Flask and Django.

D. Definition of Done

The team's DoD applied to all User Stories:

1) Code reviewed and approved by at least one peer;
2) Unit tests written and passing (Pytest / Django TestCase);
3) API endpoints validated in Postman with documented response examples;
4) Feature merged to main branch via Pull Request;
5) Swagger documentation updated for API changes;
6) Acceptance criteria from User Story verified by team member.

VI. Risk Management

A. Risk Identification and Categorisation

Risks were identified through a structured team risk workshop at Sprint Planning 1, categorised as Technical, Operational, or Scheduling, and rated by probability (1–5) and impact (1–5). Fig. 3 presents the risk heat map.

B. Risk Mitigation Strategies

Table III details mitigation strategies for the five highestpriority identified risks.

VII. Testing Strategy

A. Test-Driven Development (TDD)

TDD was applied to all Flask API endpoint implementations, following the Red-Green-Refactor cycle [9]. The process ensured that every endpoint had a failing test before implementation began, guaranteeing test coverage parity with production code. Fig. 4 illustrates the TDD workflow applied.

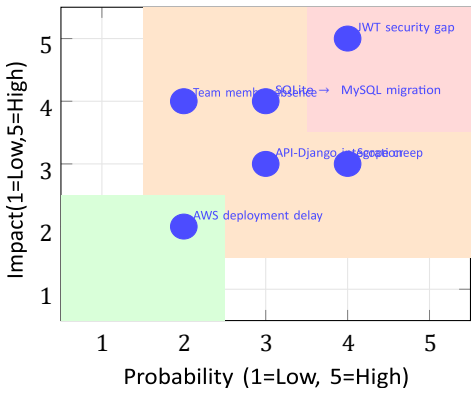

Fig. 3. Risk heat map: probability vs. impact. Red zone = high priority; green = monitor.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

TABLE III
TOP-5 RISK REGISTER WITH MITIGATION ACTIONS

Risk	Rating	Mitigation
JWT security gap (incomplete auth)	P4/I5	Prioritised Sprint 1 backlog; TDD security tests first
SQLite → MySQL migration failure	P3/I4	Documented schema; migration scripts tested in Sprint 3
Team member absence	P2/I4	GitHub task documentation; cross-training across artefacts
Scope creep (feature expansion)	P4/I3	MoSCoW enforcement; Product Owner approval required
API/Django integration mismatch	P3/I3	Shared Swagger contract; integration tests in Sprint 2

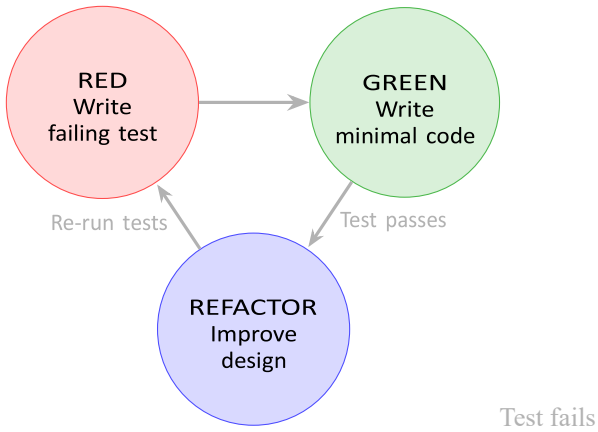

Fig. 4. TDD Red-Green-Refactor cycle applied to Flask API development.

B. Test Coverage by Module

VIII. Design And Iterative Development

A. Design Artefacts

Design artefacts evolved iteratively across Sprints following Agile's inspect-and-adapt principle [10]:

- UML Class Diagrams: Modelled User, Telemetry, Subscription, and Product entities; Test Coverage (%)

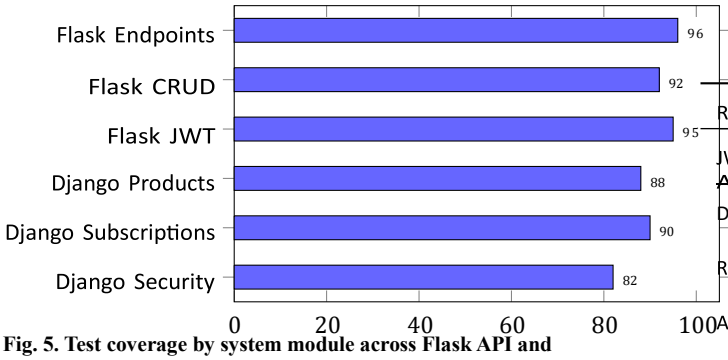

Fig. 5. Test coverage by system module across Flask API and Django Platform

TABLE V
Requirements Delivery Evaluation

Requirement	Status	Notes
JWT-secured Flask API	Delivered	All endpoints protected; 95% test coverage
Django eCommerce	Delivered	Products, subscriptions, responsive design
Role-based access	Delivered	Admin/Customer roles enforced
AWS deployment	Delivered	EC2 + RDS MySQL in pro-

- Sequence Diagrams: JWT authentication flow between Django client, Flask API, and SQLite/MySQL;
- Entity-Relationship Diagrams (ERD): Extended from Sprint 1 (Telemetry only) to Sprint 3 (full multi-entity schema);
- Wireframes: Evolved from paper prototypes [14] to Bootstrap-responsive HTML layouts;
- 3NF Normalisation: Applied to eliminate redundancy and ensure referential integrity [13].

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

B. Product Increments

TABLE IV
SHIPPABLE PRODUCT INCREMENTS BY SPRINT

Sprint	Shippable Increment
Sprint 1	Flask API: CRUD endpoints, JWT auth, Swagger docs, SQLite persistence
Sprint 2	Django: accounts, products, subscriptions, dashboard, responsive UI
Sprint 3	SQLAlchemy ORM integration, MySQL migration, Bootstrap refinement, GitHub workflow

IX. Retrospective Evaluation

A. Delivery Against Requirements

All Must Have requirements were delivered. Two Should Have features (advanced analytics dashboard, automated email notifications) were deferred to future Sprints based on velocity capacity analysis. Table V summarises delivery completeness.

B. Key Retrospective Insights

Three principal lessons emerged from the project retrospective:

1) Self-organisation value: Autonomous task negotiation and flexible role assignment enabled the team to absorb unexpected absences and capitalise on individual technical strengths;
2) Iterative design efficacy: Maintaining living design artefacts (ERDs, wireframes) updated at each Sprint duction Advanced analytics Deferred Moved to future Sprint; out of capacity boundary prevented design drift and integration surprises;
3) TDD confidence multiplier: TDD reduced debugging time in Sprint 3 integration testing, as individual component correctness was pre-verified—consistent with Beck's [9] findings on defect rate reduction.

X. Conclusion

This paper has presented an empirical account of Scrum framework application to the concurrent development of two interdependent climate technology software artefacts.

The MoSCoW/Planning Poker requirements engineering process successfully structured 24 User Stories into deliverable Sprint backlogs. The integrated risk management strategy, particularly the prioritisation of JWT security testing in Sprint 1, prevented the highest-rated risk from materialising as a delivery blocker. TDD produced high test coverage (>88% across all modules) with manageable test maintenance overhead. Sprint velocity analysis demonstrates stable team performance across three Sprints with 95.6% story point completion rate. All Must Have requirements were delivered within the 12-week schedule. The project confirms the applicability of Scrum to multi-component, climate-oriented software development, with the cross-artefact dependency mapping emerging as a critical coordination mechanism specific to this context.

REFERENCES

[1] K. Beck et al., Manifesto for Agile Software Development. 2001. [Online]. Available: https://agilemanifesto.org/
[2] K. Schwaber and J. Sutherland, The Scrum Guide. 2020. [Online]. Available: https://scrumguides.org/
[3] M. Cohn, User Stories Applied: For Agile Software Development. Boston: Addison-Wesley, 2004.
[4] M. Cohn, Agile Estimating and Planning. Upper Saddle River: Prentice Hall, 2005.
[5] B. Wake, "INVEST in good stories, and SMART tasks," XP123, 2003.
[6] J. Highsmith, Agile Software Development Ecosystems. Boston: Addison-Wesley, 2002.
[7] D. K. Rigby, J. Sutherland, and H. Takeuchi, "Embracing agile," Harvard Bus. Rev., vol. 94, no. 5, pp. 40–50, 2016.
[8] E. C. Conformito et al., "Can agile project management be adopted by industries other than software development?," Proj. Manag. J., vol. 47, no. 3, pp. 21–34, 2016.
[9] K. Beck, Test-Driven Development: By Example. Boston: AddisonWesley, 2003.
[10] S. W. Ambler, Agile Modeling: Effective Practices for XP and the Unified Process. New York: Wiley, 2002.
[11] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process. Boston: Addison-Wesley, 2012.
[12] I. Sommerville, Software Engineering, 10th ed. Boston: Pearson, 2016.
[13] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 7th ed. Boston: Pearson, 2016.
[14] C. Snyder, Paper Prototyping. San Francisco: Morgan Kaufmann, 2003.