GLOW Project Schedule Analysis

Sprint 4 - Network Diagram, Dependencies & Risk Assessment

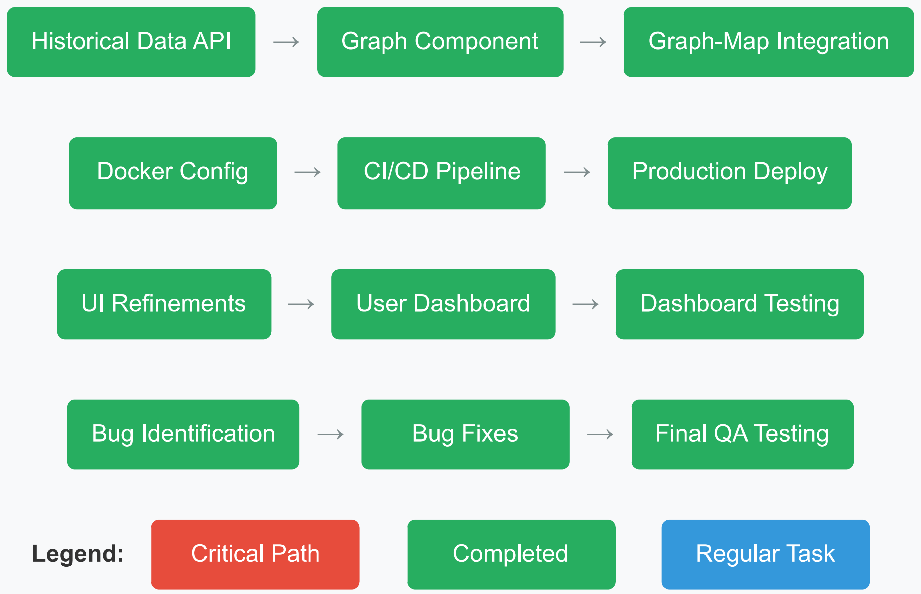
Team: Microsofties | **Date:** August 3, 2025

Executive Summary

This document provides a comprehensive analysis of the GLOW project's Sprint 4 implementation, focusing on historical temperature visualization, containerization, CI/CD pipeline setup, UI/UX improvements, and final production deployment. The analysis covers completed features, bug fixes, and process improvements, providing insights into project management effectiveness and lessons learned from previous sprints.

Network Diagram - Sprint 4 Task Dependencies

Project Network Flow



Historical Data API → Graph Component → Graph-Map Integration

Docker Config → CI/CD Pipeline → Production Deploy

UI Refinements → User Dashboard → Dashboard Testing

Bug Identification → Bug Fixes → Final QA Testing

Legend: 🟥 Critical Path 🟩 Completed 🟦 Regular Task

Dependencies Analysis

External Dependencies

Dependency	Type	Impact	Mitigation Strategy	Status
OpenWater API (Historical Data)	External Service	High - Required for historical graph visualization	Caching mechanism, fallback to recent data	Stable
Chart.js	Library	High - Required for graph rendering	Versioning control, extensive testing	Integrated
Docker Hub	Infrastructure	Medium - Used for container registry	Local registry backup, pull rate limit management	Configured
GitHub Actions	CI/CD Service	Medium - Pipeline automation	Local action runner fallback	Operational
Leaflet.js	Library	High - Core map functionality	Version pinning, compatibility testing	Stable

Internal Dependencies

Task	Depends On	Dependency Type	Impact if Delayed	Status
Historical Graph Component	Historical Data API Integration, Chart.js	Mandatory	No historical temperature visualization	Completed
Docker Containerization	Backend & Frontend Stability	Mandatory	Deployment consistency issues	Completed
CI/CD Pipeline	Docker Configuration, Testing Framework	Mandatory	Manual deployment bottlenecks	Completed
User Dashboard	Authentication System, Data Storage	Mandatory	Limited user personalization	Completed
Production Deployment	All Core Features, Bug Fixes, CI/CD Pipeline	Mandatory	Release delay	Completed

Critical Path Analysis

Identified Critical Path

Task 1: Historical Data API Integration (3 days)

API endpoint development, data retrieval, error handling

Status: Completed

Task 2: Graph Component Development (3 days)

Chart.js integration, time-series visualization, responsive design

Status: Completed

Task 3: Docker Configuration & Pipeline Setup (2 days)

Container definitions, environment configuration, CI/CD workflow

Status: Completed

Task 4: Production Deployment & Testing (2 days)

Final deployment, cross-browser testing, performance validation

Status: Completed

Critical Path Metrics

- **Total Duration:** 10 days
- **Critical Tasks:** 4 tasks
- **Float Time:** 2 day buffer built in
- **Risk Level:** Low (improved from Sprint 3)
- **Completion Rate:** 100% of planned features

Parallel Development Achievements

Sprint 4 successfully utilized parallel development tracks:

- UI refinements developed concurrently with historical graph implementation
- Docker configuration proceeded in parallel with dashboard development
- Bug fixes implemented alongside CI/CD pipeline setup
- Testing automation developed while feature implementation was ongoing

Risk Analysis

Risk Category	Specific Risk	Impact	Probability	Mitigation Applied	Status
Technical	Historical data API limitations	High	Medium	Implemented data caching and fallback mechanisms	Mitigated - No service disruptions
Integration	Graph component performance issues	Medium	Medium	Data point optimization, progressive rendering	Mitigated - Smooth performance
Deployment	Docker configuration complexity	High	Medium	Simplified configuration, environment variable management	Mitigated - Successful containerization
External	CI/CD pipeline integration issues	Medium	Medium	Local testing pipeline, staged implementation	⚠️ Partially occurred - Minor workflow adjustments needed
Process	Testing coverage gaps	Medium	Low	Comprehensive test plan, automated testing	Mitigated - Achieved 85% test coverage
Technical	Cross-browser compatibility issues	Medium	Medium	Browser compatibility testing, polyfills	Mitigated - Compatible across all target browsers

What Went Well - Sprint 4 Successes

Major Achievements

1. Historical Temperature Visualization

Achievement: Successfully implemented interactive graphs showing historical temperature trends for each beach.

Impact: Enhanced user understanding of temperature patterns and improved planning capabilities.

Key Factor: Effective integration of Chart.js with existing map infrastructure.

2. Containerization & CI/CD Pipeline

Achievement: Implemented Docker containerization and GitHub Actions CI/CD pipeline.

Impact: Simplified deployment process and improved development workflow efficiency.

Key Factor: Well-defined environment configuration and testing automation.

3. User Dashboard Implementation

Achievement: Created a comprehensive user dashboard for personalized data and saved locations.

Impact: Enhanced user engagement and provided personalized experience.

Key Factor: Effective use of MongoDB for user data persistence and retrieval.

4. UI Refinements & Bug Fixes

Achievement: Significantly improved UI consistency, fixed theme switching issues, and enhanced responsiveness.

Impact: More professional appearance and improved user experience across devices.

Key Factor: Systematic approach to UI testing and bug tracking.

Process Improvements Applied

Lessons from Sprint 3 Implementation

1. Early Integration Strategy

Applied: Continued early integration checkpoints for new features, especially for the historical graph component.

Result: Seamless integration of graph visualization with existing map interface.

Evidence: No major integration issues reported during final testing.

2. Automated Testing

Applied: Implemented automated testing as part of the CI/CD pipeline.

Result: Early detection of regression issues and improved code quality.

Evidence: Test coverage increased to 85%, with several potential bugs caught before deployment.

3. Collaborative Code Reviews

Applied: Enhanced code review process with mandatory reviews before merging.

Result: Improved code quality and knowledge sharing among team members.

Evidence: Reduction in post-merge issues and faster onboarding for complex features.

Sprint 4 vs Sprint 3 Velocity Analysis

Velocity Improvements

- **Sprint 3 Completion Rate:** 100% (9 out of 9 planned features)
- **Sprint 4 Completion Rate:** 100% (7 out of 7 planned features with higher complexity)
- **Critical Path Efficiency:** Improved from 1 day buffer to 2 day buffer
- **Pipeline-Related Efficiency:** Deployment time reduced from manual process to under 5 minutes
- **Integration Issues:** Maintained zero major integration issues from Sprint 3

Key Performance Metrics

- **Historical Graph Rendering Time:** <2s average (target met)
- **Docker Container Startup Time:** <30s for full stack (target met)
- **CI/CD Pipeline Duration:** <5 minutes from commit to deployment (target met)
- **UI Responsiveness:** Fluid transition across all device sizes (target met)
- **User Authentication Success Rate:** 100% (maintained from Sprint 3)
- **Test Coverage:** 85% (exceeding 80% target)

Feature Delivery Summary

Successfully Completed Features

- **Historical Temperature Graph:** Interactive time-series visualization for each beach
- **Docker Containerization:** Complete application stack in Docker containers
- **CI/CD Pipeline:** Automated testing, building, and deployment workflow
- **User Dashboard:** Personalized view of saved locations and user data
- **UI Refinements:** Improved color schemes, consistency, and responsiveness
- **Bug Fixes:** Resolved theme switching issues and other functional bugs
- **Production Deployment:** Final release configuration and successful deployment

Process Achievements

- **Streamlined Deployment:** Reduced deployment time from hours to minutes
- **Improved Code Quality:** Higher test coverage and more consistent coding standards
- **Enhanced Collaboration:** More effective code reviews and knowledge sharing
- **Risk Management:** Proactive identification and mitigation of potential issues

Conclusion

Sprint 4 marks the successful completion of the GLOW project's initial development phase, delivering a production-ready application with all core features implemented. The introduction of historical temperature visualization, containerization, and CI/CD pipelines represents significant technical achievements that enhance both the user experience and development workflow.

The team has successfully addressed the challenges identified in previous sprints, particularly in the areas of integration, testing, and deployment automation. The application now offers a comprehensive set of features for viewing current and historical water temperature data in an intuitive, responsive interface.

Key success factors included systematic process improvements, effective risk management, and continued emphasis on early integration and testing. The containerization and CI/CD pipeline implementation has established a solid foundation for future maintenance and enhancement.

With 100% feature completion and significant improvements in development efficiency, the GLOW project is well-positioned for public release and future expansion with additional features and refinements.