

Development Approach Recommendation Memo

Lindsey A. Monyelle

DAT 445 Predictive Analytics and Machine Learning (WI)

Dr. Jess Schwartz

October 16, 2026

Project Overview

The proposed system is a separate, searchable catalog for combat items in Kingdom of Loathing (KoL). Right now, information about these items is spread across wiki pages, item descriptions, and player knowledge, which can make it difficult to compare items and figure out which ones are useful for a specific situation. The catalog would bring this information together in one place, with details about damage, healing, combat effects, and other unique behaviors. It would also work with a script that provides information about inventory quantities and Mall prices. The main goal is to make it easier for players to find items that meet their needs and give script authors access to more consistent item information.

Development Approach Options

Agile Development

Agile development organizes work into smaller increments that are implemented, tested, and refined throughout the project. Rather than defining every requirement upfront, developers establish core requirements and adjust the system based on what they learn during development. This approach allows large projects to be divided into manageable phases, with each increment delivering functional improvements. One advantage of Agile is that developers can respond to new discoveries, changing requirements, and technical problems without redesigning the entire system. Early testing can also identify missing or inconsistent data before they become larger problems. However, frequent changes can lead to scope creep and inconsistent results if requirements, documentation, and development priorities are not clearly managed.

Prototyping-Based Development

Prototyping-based development uses an early model of a system to evaluate its design and functionality before committing to a complete implementation. Prototypes can range from simple interface mockups to working versions that use sample data, allowing developers to test whether the design and underlying data model support the intended functionality. This approach is useful for validating search and comparison features before expanding the system into a fully usable product. Building a proof of concept can reveal design flaws and technical limitations early, reducing the risk of investing time in an approach that does not work. However, a prototype may rely on carefully selected example data that does not represent the complexity of the full dataset, and an interface that appears functional does not guarantee that the underlying system is accurate or reliable. Prototyping is therefore valuable for validating specific aspects of a system but does not necessarily provide a complete development process on its own.

Traditional SDLC / Waterfall

The traditional Systems Development Life Cycle (SDLC), often associated with the waterfall approach, organizes development into defined phases, including planning, requirements analysis, design, implementation, testing, and maintenance. Each phase is substantially completed before the next begins. A major strength of this approach is its emphasis on upfront planning and documentation, which helps establish clear requirements and a consistent system design before implementation. A defined data dictionary, entity-relationship design, and process models can guide how information is organized and how system components interact. However, waterfall works best when requirements are stable and well understood from the beginning. If new behaviors or exceptions are discovered during implementation, changes to the data model or search functionality may require significant rework. Delaying testing until later phases can also allow design problems to go undetected for longer than they would under an iterative approach.

Comparison of Approaches

Agile Development	Prototyping Model	Waterfall
Project size		
Medium to large projects with evolving requirements..	Small to medium projects with uncertain requirements or designs.	Projects of any size with stable requirements and clearly defined phases.
Requirement clarity		
Requirements are refined throughout development.	Requirements are explored and clarified through evaluation.	Requirements are fully defined before implementation.
Changing requirements		
Accommodates changes between development iterations.	Supports design changes based on prototype evaluation.	Changes to completed phases may require significant rework.
User involvement		
Supports ongoing feedback, although participation can vary.	Focuses feedback on evaluating the prototype.	Emphasizes feedback during requirements and final review
Documentation		

Requires deliberate effort to keep design decisions and requirements current.	Documents and tests selected aspects of the design but may not cover the full system.	Emphasizes formal documentation throughout the development phases.
Technical complexity		
Tests data structures, search logic, and integrations incrementally.	Evaluates selected technical features before full implementation.	Plans the entirety of the system upfront.
Risk		
Requires strong attention to scope management	Underlying technical issues may be excluded from the prototype model.	May leave design or implementation problems undiscovered until later testing.

Recommended Development Approach and Implementation Considerations

Agile iterative development is the best fit for the KoL Combat Item Catalog, with prototyping used to validate designs and functionality. Unlike Waterfall, Agile allows requirements to evolve as research reveals additional item behaviors or exceptions. Prototyping helps test specific features, but Agile provides the broader process needed to develop, test, and maintain the complete system.

Development should begin with the existing requirements, data dictionary, ERD, and process models, followed by a small set of verified combat items and basic effect-based searches. The Search, Results, and Compare prototypes can then be evaluated for usability before expanding the catalog and adding filtering and live inventory and Mail price retrieval. Feedback from players or script authors, when available, can help identify confusing searches or missing functionality.

Progress should be tracked through completed features and testing milestones. Testing should verify catalog accuracy, search results, and items with multiple or unusual effects. Requirements, data definitions, and test results should be documented and updated as the system evolves. The main risks are scope creep, inconsistent item data, and limited external feedback. Keeping the initial scope focused on core catalog and search functionality will help control these risks while leaving room for future improvements.

References

- A KoL Wiki. (2026, July 8). *Banishing*. A KoL Wiki. Retrieved August 30, 2026, from <https://wiki.kingdomfloathing.com/Banishing>
- A KoL Wiki. (2024, January 18). *Combat Items*. A KoL Wiki. Retrieved September 1, 2026, from https://wiki.kingdomfloathing.com/Combat_Items
- gausie. (2026, August 22). *kolmafia/src/data/modifiers.txt*. Github. Retrieved August 30, 2026, from <https://github.com/kolmafia/kolmafia/blob/4b19681c000308d08c4e55452c246883847a4d6c/src/data/modifiers.txt>
- KoLmafia Wiki. (2026). CLI reference. KoLmafia Wiki. Retrieved September 1, 2026, from https://wiki.kolmafia.us/index.php?title=CLI_Reference
- KoLmafia Wiki. (2021, March 15). *Custom Combat Script - Kolmafia*. KoLmafia Wiki. Retrieved September 1, from https://wiki.kolmafia.us/index.php?title=Custom_Combat_Script
- KoLmafia Wiki. (2023, March 10). Modifier Maximizer - Kolmafia. KoLmafia Wiki. Retrieved September 1, 2026, from https://wiki.kolmafia.us/index.php?title=Modifier_Maximizer