AI4Agile Alpha Prototype

CptS 421 - Bolong Zeng
Authors: Phong Bach, Emily Cawlfield, Nain Galvan, and Aric Monary
Mentor: Skip Baccus

Date Submitted: 12/11/20

1. Alpha Prototype Description

The Alpha Prototype implements the following features:

1. Ability to edit generated suggestions
2. Iterative issue creation process
3. Ability to select or deselect suggestions
    3.1. Select All
    3.2. Deselect All
4. Epic Decomposition
    4.1. Slider to select amount of created stories
5. Story Optimization
    5.1. Slider to select connectivity degree of optimized stories
6. Task Generation
7. Relationship Graph
    7.1. Issue Tree
    7.2. Developer Clusters

A majority of these were core requirements defined in the initial project prompt from SCORE 2021. The additional items are the result of design choices or expanded interpretations of those requirements in order to achieve certain quality attributes.

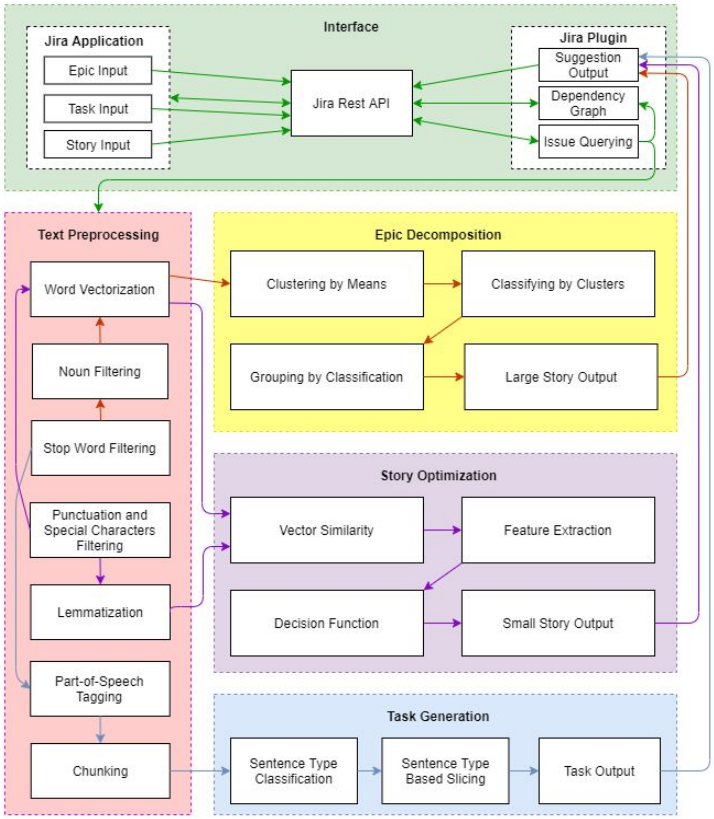

Figure 1.1: Block diagram showing the architecture and interactions between components, with a common path through text preprocessing before transitioning into individual AI processes.

Figure 1.1 shows the component interactions between collections of functions or sub-processes. The five main blocks of the alpha prototype application are the interface, text preprocessing, epic decomposition, story optimization, and task generation. The interface contains the front end of both the Jira platform and this application, which is integrated within the platform. The first step to each of the AI processes to prepare the text data, either stemming from user input or generation from the AI processes. Epic decomposition is a clustering process that groups together requirements and specifications, which are in the form of individual sentences, into blocks based on related wording. Story optimization is a process of breaking down user stories by grouping sentences that has high semantic similarity. The final block, task generation, is a process of generating tasks, which are deliverables, based on simple sentence generation.

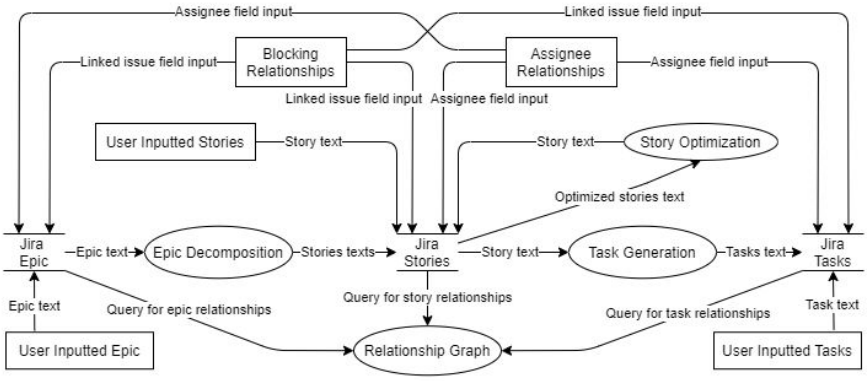

Figure 1.2: Dataflow diagram showing the various entry points into the application and the way data moves through the combined Jira and AI4Agile system.

From a high level viewpoint, the various AI processes appear to be linear and interdependent. In reality, each process can happen independently. A typical user starts by inputting an epic, decomposing it, optimizing all of the generated stories, and finally generating tasks from the optimized stories. A user may also choose to enter new stories or change existing stories between each process. As an example of independent execution, a user may choose to only use task generation from stories that have been manually entered.

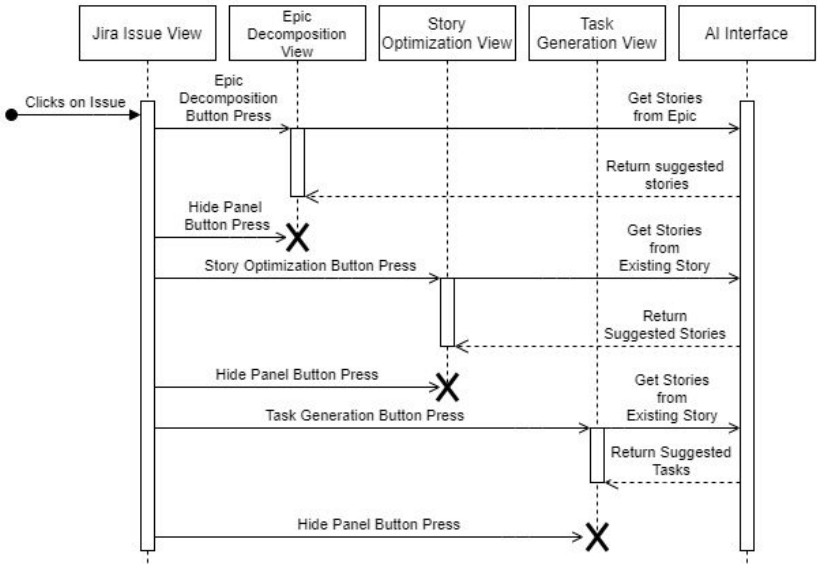

Figure 1.3: Suggestion web panel sequence diagram showing the message passing between actions and views.

The available action sequences for the alpha prototype are outlined in figure 1.3, which goes into details such as the loading and creation of web panels. This goes into more detail on the backend and frontend communications of the app, showing how the web panels rendered in Jira communicate with the backend to generate the suggestions for each AI.

2. Design Modifications Resulting from Alpha Prototype Testing

2.1 Updates to the planned overall solution approach

One major design modification located in the interface block of figure 1.1 was the use of the Flask API to map HTTP requests to Python functions that reside in the text preprocessing, epic decomposition, story optimization, and task generation processes. Flask is a web framework for Python, meaning that it provides functionality for building web applications, including managing HTTP requests and rendering templates. Flask applications tend to be written on a blank canvas, and so are more suited to a contained application such as prototype API.

The second design modification was the relationship graph algorithm. For tree graphs, the graph only shows the currently selected issue's parent issue and all the selected one's child issues. This eliminates unnecessary information and focuses on giving users an understanding of the child issues, which can be important in doing impact analysis. In the beginning, the cluster graphs were expected to behave similarly to the tree graphs. However, there is little to no benefit in viewing a single cluster graph with one developer. The updated primary focus of cluster graphs is to show the user the relationships between all developers and their issues. As a result, the user can view all developers and the issues that are assigned to them, assisting in high-level views where Jira's feature set is lacking.

There were some changes in the AI components. The main modification is the design choice for epic decomposition. Story optimization main components remain the same. Task generation had some complication with the current python implementation. The updates to the components are discussed in more detail in section 2.2.

2.2 Updates to the planned components

Epic Decomposition

The primary change to the Epic Decomposition process was the utilization of k-means clustering over a dynamic clustering technique, such as DBSCAN or Mean-shift. Dynamic clustering techniques were unable to be implemented in a manner that yielded consistent meaningful clusters across many epic inputs. This was a result of an inability to filter noise such that the clusters were neither completely disjoint or entirely overlapping. Thus, k-means, a centroid-based clustering algorithm, was used at the cost of selecting a default number of stories to generate. The default number of stories generated is five but the user can select between 2 and 10 stories using a newly added slider in the UI for the Epic Decomposition process.

Story Optimization

A new update to the story optimization is the slider integration. The user can choose to increase or decrease the degree of connectivity between sentences. The slider displays options from 0 to 10, which maps to values from 0.65 to 0.75 as threshold values on the backend.

The slider integration was a feature that was added based on mentor feedback. A parameter is exposed to give the user more control of the output. The exposed parameter is the degree of connectivity between the sentences. It was determined after impact analysis that it is better to implement it this way because it generates more meaningful results tailored to each user. This also eliminates the need for having a fixed threshold number, since the optimal threshold number can vary between stories.

Task Generation

The modification for task generation was in changing the python library. Initially the natural language toolkit (NLTK) library was going to be used, but it did not support the features that were needed to make task generation functional. The main feature that NLTK lacked was word dependency. Word dependency is important to this process, since this was the main feature used to create simple sentences. Word dependency returns a pair of words, identifying the dependency between the two. The new python library being used is Stanza. This library allows for part-of-speech tagging, word dependency, and tree parsing. By having part-of-speech tagging and word dependency, the creation of simple sentences from complex and compound sentences was easier to implement.

3. Future of the Project

The main scope of development was bounded by the requirements for the SCORE 2021 competition during the first semester of the project. During the second semester, the scope will be based upon the successful deployment of the app to the Atlassian Marketplace along with input from an actual customer. As a result of the app becoming customer-oriented, the specific features of the application will be dependent on the guidance of the customer, which will be identified at the beginning.

The schedule is geared towards the eventual deployment of the app on the marketplace. As a result, the main epics throughout the coming semester will be as follows:

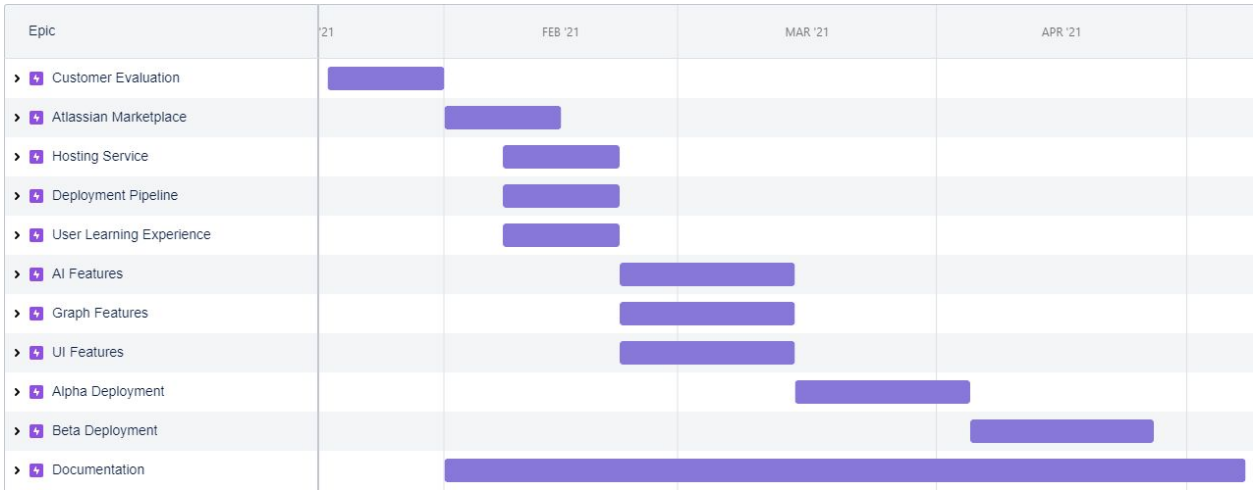

Figure 3.1: Gantt Chart of the major epics for Spring Semester

Link to Gantt Chart to view Tasks per Epic

Each epic has a set of sub-tasks that will be completed within the timeframe of their parent epic. The currently identified sub-tasks within each epic are found within Table 3.1.

Epic	Task	Point Person	Start Date	End Date
Customer Evaluation	Identify Real Customer(s)	Aric Monary	1/18	1/31
	Evaluate Customer needs			
	Get Customer feedback on existing software			
Atlassian Marketplace	Review Marketplace Guidelines	Emily Cawlfield	2/1	2/14
	Review Existing App's Compliance with Guidelines	Phong Bach		
	Evaluate Security Compliance	Aric Monary		
	Integrating Authentication	Nain Galvan		
Hosting Service	Research Hosting Options	Nain Galvan	2/8	2/21
	Evaluate Software Hosting Needs			
	Retrofit Software to be Hosted			
Deployment Pipeline	Evaluate DevOps services based on Hosting Service	Phong Bach	2/8	2/21
	Deploy the Pipeline			
	Integrate Pipeline into development			
	Link Pipeline to Jira			
User Learning Experience	Prioritize use cases from customer	Emily Cawlfield	2/8	2/21
	Develop Written User Guide			
	Develop Integrated Instructions into app			
AI Features	Caching AI results	Aric Monary	2/22	3/14
	Filtering "bad" generated Tasks	Nain Galvan		
	Improve Noise Handling	Phong Bach		
Graph Features	User settings for Rendered Fields	Emily Cawlfield	2/22	3/14
	Additional Relations			
	Additional Fields			

Table 3.1: Task Table for Spring Semester

Epic	Task	Point Person	Start Date	End Date
UI Features	Additional Warnings for Edge Cases	Aric Monary	2/22	3/14
	Revise Layout by User Feedback			
Alpha Deployment	Evaluate Initial User Feedback	Nain Galvan	3/15	4/4
	Fix Critical Bugs from Alpha	Depends on Bug		
	Create New Features per User Feedback	Depends on Feature		
Beta Deployment	Deploy new Alpha Features	Depends on Feature	4/5	4/26
	Fix Critical Bugs from Beta	Depends on Bug		
	Evaluate Beta User Feedback	Phong Bach		
Documentation	Beta Prototype Updates	N/A	2/1	2/5
	Midterm Progress Report		3/7	3/12
	Final Report		4/26	5/7

Table 3.1 (continued): Task Table for Spring Semester