Team Katara - AI4Agile Initial Report

Phong Bach, Emily Cawlfield, Nain Galvan, Aric Monary
Washington State University Everett

I. Introduction

AI4Agile is an application for Jira Cloud, made to assist software development teams who use agile methods in user story streamlining. The application consists of three processes to refine user stories from an epic into tasks. An additional visualization component, integrated into Jira, offers suggestions to users and visualizes both explicit and implicit relationships amongst epics, user stories, and tasks.

The primary three processing can be encompassed as epic decomposition, story optimization, and task generation. Epic decomposition consists of taking a templatized epic inputted by the user and clustering requirements together into user stories based upon subject. Story optimization further breaks down potentially large stories into smaller stories. Certain stories may not be subject to change depending on their initial size. Lastly, the user stories will be broken into the tasks based on parts-of-speech in the stories description. At each process, the results will be offered as suggestions to the user, allowing them to select which user stories and tasks they'd like to add to their Jira board. The user can further change any suggestions along with regenerating new suggestions based upon new parameters. The input for each process will be based on the user's selected suggestions and inputted stories. Epics, user stories, and tasks may be selected and a graph field will populate showing a selectable explicit and implicit relationships which may be rendered in a tree or cluster form.

The report is composed of ten sections. The second section outlines our initial understanding of the application of AI to the agile development process and our vision of the app . The third section covers the structure of the team and methods of collaborating with stakeholders. The fourth section summarizes the decomposition of requirements into functional and non-functional categories. Section five outlines the design and section six explains the implementation along with the differences between it and the initial design. Section seven goes through two use cases that encapsulate the primary functionalities that were implemented. Section eight describes the methods for evaluating the user interface and experience in addition to testing the main processes. Section nine explores the future prospects of the project. Lastly, section 10 offers reflection on the outcome of the project.

II. Preliminary Understanding

What follows is a discussion of the background of this project, as well as the team's interpretations of the undertaking from the start. This project originated from the SCORE 2021 competition prompt titled "AI4Agile: Developing AI-enabled tool support for user story refinement in JIRA". This prompt was sponsored by professor Hoa Khanh Dam, and tied into previous research he was involved with, one of the papers of which was linked directly within [1]. The prompt further went into the reasoning for AI usage in this setting and the vision for such an application, which along the way this team has expanded upon and further detailed as the project progressed in order to better understand the vision.

The prompt given for this project listed four primary requirements towards the goal of aiding in agile user story refinement. These four were:

1) Recommend user stories that are decomposed from an epic.
2) Recommend smaller user stories are [sic] split from bigger user stories.
3) Recommend subtasks derived from a user story.
4) A visualization (e.g. a graph) of epics, user stories and tasks, and their relationships.

In addition to these requirements, it was specified that the application was to be integrated into the platform of Atlassian's Jira software. While there are different versions of Jira available to develop for, Jira Cloud was chosen over Jira Server, since the team viewed it as being more relevant to future trends in the workplace due to its focus not being on hosting on-premises for work environments. An abridged version of the vision for this project as it connects to the scheme of artificial intelligence (AI) or machine learning usage for software development was also given. A more complete vision was detailed in the linked research paper, as it contained proposals and persuasions towards a grander project for AI use in agile assistance.

The grander project's intent focused primarily on risk management via AI, while this competition project's scope, which was defined from analysis of the project prompt and input from an early on defined customer, had a primary focus of providing a time savings for users refining user stories. The risk management project involved the user of historical data and subsequently an increased utilization of AI to process that data and provide more personalized feedback and tools to teams. In that context, the necessity of AI and the form it would take was clearer, while in the prompt its role was left more open to interpretation. In the end, the team determined that the artificial intelligence component's purpose would be centered on processing text into increasingly smaller portions in order to support user story refinement.

While the project's goal on a functional level was described as supporting user story refinement, the motivation behind that given in the prompt involved reducing reliance of the process on team members' experience levels, and supporting that process so that fewer differences are introduced by people refining via their own methods. This also ties back to the ideal of the parent project for risk management, as it would improve the consistency of the refinement process, since the overlap or contradiction of rules for refining backlog items when done by hand is likely to cause inconsistency in resolution methods used, making for less predictable outcomes and backlog item sizes.

III. Collaboration

This project was used as a capstone project. It was given as one of various options that the team could choose for their capstone. The reason this specific project was chosen was because it was intriguing, since none of us had any previous experience in AI and could therefore learn something new and relevant to current industry trends.

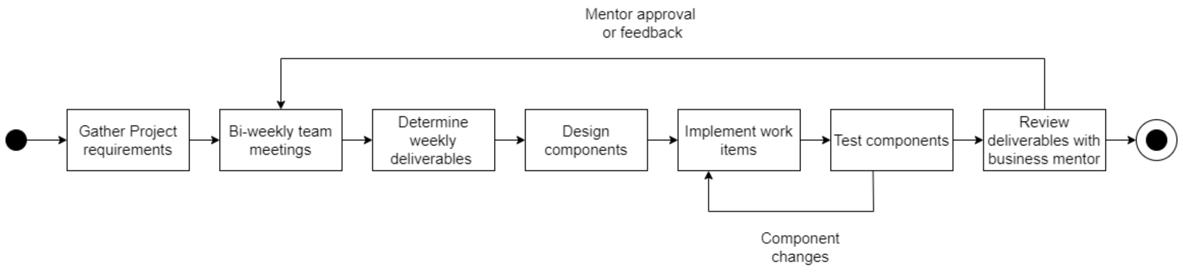

Mentor approval or feedback
Gather Project requirements
Bi-weekly team meetings
Determine weekly deliverables
Design components
Implement work items
Test components
Review deliverables with business mentor
Component changes

Fig. 1. Activity Diagram outlining the development processes

For this project we had various people collaborating. The team consisted of 1 faculty advisor, 1 business mentor, 1 faculty mentor, SCORE 2021 prompt sponsor, and 4 undergrads. It was set up to where the business mentor would be the customer, since he already had industry experience. Since this is used as a capstone there are course assignments that needed to be completed this is where the faculty mentor was used for. He provided feedback to make sure the documentation for the project would satisfy course assignments. The faculty advisor assisted with AI components design questions, since he has a background in AI. He was able to provide insightful information on which possible implementations may be needed for the AI components. The prompt sponsor was used to clarify project requirements. One undergrad was assigned the role of team liaison for easier communication between the different people involved.

In the beginning the team was split into two parts AI and UI where the AI components were considered priority. Three undergrads focused on the three requirements needed for the AI which are epic to user story, user story optimization, user stories to tasks and the last one focused on the UI requirements of visualization of their relationships. Later into the project the UI was moved as top priority. The UI consisted of visualizations between the relationships, showing the user suggestions and integrating the new buttons to the Main UI. By breaking up the AI and UI into parts allowed for more efficiency. Each component the undergrads chose was based on what intrigued them the most.

The team met a minimum of 2 times per week to go over documentation that was needed to complete the course and work on implementing the AI components and UI for the project. These meetings would feed into the business mentor meetings that were held once a week. Each time a new implementation piece was completed, it was discussed during the business mentor meetings to receive feedback from a customer perspective.

In the following sections, we will describe more in detail what work items and deliverables we performed during the project lifecycle.

IV. Requirements

First, the team modeled the requirements through the use of KAOS models [2]. By building the KAOS models, the team could describe the problem to be solved and the constraint it must be

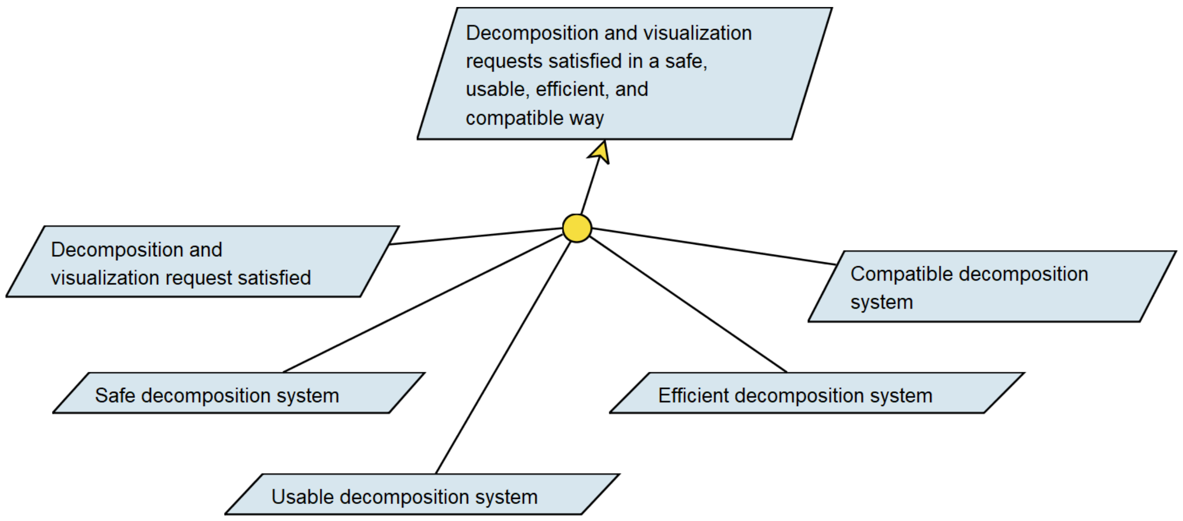

Decomposition and visualization requests satisfied in a safe, usable, efficient, and compatible way
Decomposition and visualization request satisfied
Compatible decomposition system
Safe decomposition system
Efficient decomposition system
Usable decomposition system

Fig. 2. Goal: “Decomposition requests satisfied in a secure, usable, efficient, compatible, and reliable way”.

fulfilled by any solution provider. Moreover, using the KAOS models, the team could remove or add new requirements easily with new changes. To gather requirements, there were various means but an efficient way was to reuse requirement patterns that are universal to software application. We specified the goals starting with the generic pattern of requirements and used it on new cases to guide the refinement of requirements.

The tactic that we used here is case-driven decomposition. We specified the subgoals that enumerate all the cases that must be covered to fulfill our main goal. The subgoals covered all the functional and nonfunctional that we wanted to include in our system. From the generic pattern, the goals that our application want to achieve are:

- Decomposition and visualization request satisfied
- Safe decomposition system
- Usable decomposition system
- Efficient decomposition system
- Compatible decomposition system

Continuing with generic patterns, we reused the goal generic pattern for “Safe System”, “Usable System”, “Efficient System”, and “Compatible System”. From these models, we generated the nonfunctional requirements for our system.

Next, we broke down the “decomposition request and visualizations satisfied” goal into subgoals correlated to the main functionalities that the project proposal included. The subgoals for functional requirements are:

- User stories are generated correctly from epic

TABLE I
NON-FUNCTIONAL REQUIREMENTS

ID	Description
NFR1	Show resources about Agile development
NFR2	Follow Agile development process
NFR3	Have presets for user without domain expertise
NFR4	Show notification about decomposition status
NFR5	Do not store information on third-party database
NFR6	Only query Jira the information that is needed
NFR7	Software is secure
NFR8	Retain original sentences in epics with minimal modification
NFR9	The response time for AI processing is less than five seconds
NFR10	The response time for visualization is less than two seconds
NFR11	Visualization only generate important relationships
NFR12	Available on Atlassian Marketplace
NFR13	Available on Atlassian Partner Marketplace

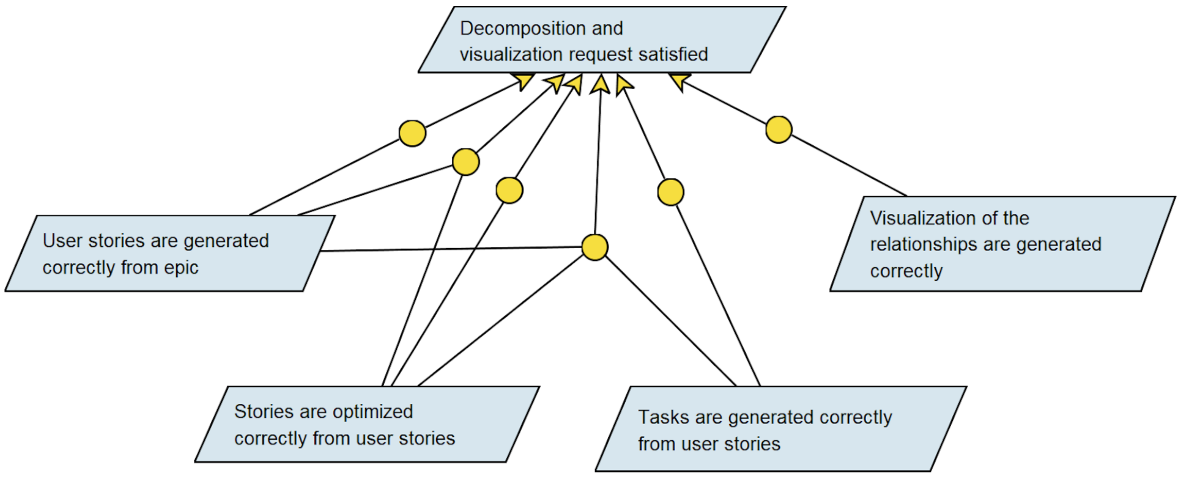

Decomposition and visualization request satisfied
User stories are generated correctly from epic
Visualization of the relationships are generated correctly
Stories are optimized correctly from user stories
Tasks are generated correctly from user stories

Fig. 3. Goal: “Decomposition request and visualization satisfied”.

- Stories are optimized correctly from user stories
- Tasks are generated correctly from user stories
- Visualization of the relationships are generated correctly

The next step is to generate the requirements from all the subgoals. Each subgoal from the “Decomposition request and visualization satisfied” goal has its own KAOS model. From these models, we generated the functional requirements for our system as shown in the following table.

TABLE II
FUNCTIONAL REQUIREMENTS

ID	Description
FR1	Clustering works correctly
FR2	Allow user to select and discard generated issue
FR3	Can process a variety of input type
FR4	User stories extraction works correctly
FR5	Retain user stories if those are small enough
FR6	Sentence building works correctly
FR7	Show the explicit relationship between issues as a tree
FR8	Show the implicit relationship of developers to the issues as clusters
FR9	Allow user to show and edit relationship
FR10	Show a customizable type and depth to relationship between issues
FR11	The graph should render as the object is selected

V. Design

A. AI Criteria

The overall process that was found when performing an epic breakdown is a combination of clustering related requirements, specifications, and details along with the further breakdown of complex info into smaller stories and tasks. Clustering can be performed in either supervised or unsupervised learning. Supervised learning requires the existence of a data set that has been correctly labeled. Although a general data set of decomposed epics can be generated, it was deemed that the range of topics an epic can encompass was too large to have a trained model that would accurately cluster the content of an epic.

Additionally, an early assumption made about the decomposition process is that the text processing techniques should not rely upon the existence of historical project information, whether that be in the form of previously completed epics or sprints. As a result, unsupervised clustering techniques were only considered as they allowed for the decomposition process to be applicable to new projects, which lack history, and projects that have not been topically available to train a clustering model off of.

The last criteria for the text processing approaches under consideration was that no additional information should be generated through AI. This was due to the previously discussed lack of data, specific to the project domain, that could be used to potentially generate stories and tasks.

B. Epic Decomposition

Through manual decompositions of epics into stories it was identified that the process fundamentally involved the categorization of requirements and specifications. From such, text clustering techniques based upon the vectorization of the contents of epics. Each requirement, assumed to be in the form of a complete sentence or distinguishable section, is vectorized and normalized so a comparison can be performed.

Based on a series of tests on a simpler text clustering problem, K-means, a non-deterministic centroid-based clustering technique, yielded the best results when it came to creating consistent

TABLE III
AI DESIGN CHOICES

Component	Options	Choice	Reasoning
Epic Decomposition	• KMeans • Meanshift • DBSCAN • OPTICS	Mean-shift	All of the mentioned clustering algorithms are dynamic as they do not require the number of clusters as input. Mean-shift only requires the size of the region to search through (bandwidth), which can be estimated, where all other options require arbitrary values to create clusters.
Story Optimization	• TF-IDF • SIF • Word2vec • Cosine simularity • Word mover's distance	SIF with word2vec and Cosine similarity	TF-IDF is the most simple for word embedding. SIF with word2vec can achieve a higher accuracy than TF-IDF with the custom word2vec model that is relevant to the topic. WMD is far more expensive to calculate, especially on longer texts. Cosine similarity can give the same performance without losing too much semantic similarity.
Task Generation	• Sentence classification	Sentence classification	There is a need to split up the different types of sentences, such as complex and compound, into simple sentences, in order for them to be manipulated into tasks. To do this, sentence classification is the necessary first step.

clusters of requirements. The downside to K-means is that it requires the number of clusters to be identified, forcing there to be user input or an estimation to be made. Since it was a goal to have the app be applicable to a large range of epics, in terms of topic and size, issues such as DBSCAN, OPTICS, and Mean shift were explored as they are dynamic and do not require the number of clusters to be specified. DBSCAN/OPTICS are density-based clustering techniques and Mean shift is a centroid-based clustering technique. Ultimately, it was decided to go with Mean shift since the implementation of the technique included an estimation of the cluster sizes, thus requiring no external input.

C. Story Optimization

The process of story optimization is further breaking down a story to if it contains more than one story or software feature that needed to be achieved. For story optimization, sentence similarity is high priority.

A pre-trained Word2Vec model was used to extract the features out of sentences. A sentence embedding technique called SIF embeddings (Smooth Inverse Frequency) was used to compute

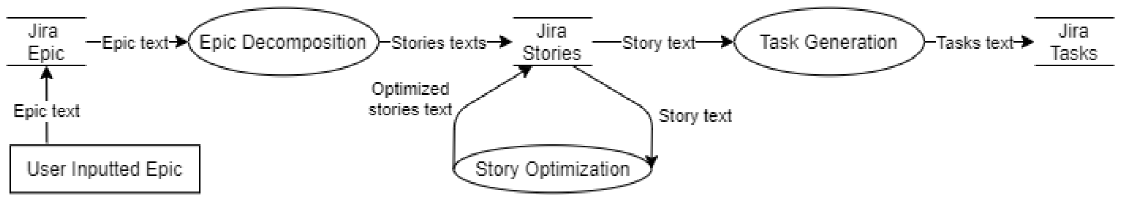


Fig. 4. Simplified Data Flow diagram outlining the flow of the decomposition processes

sentence embeddings as a weighted average of word vectors. Using the features, a comparison was made by calculating the cosine similarity between sentences.

The result was the similarity coefficient between sentences and it could be used to group sentences together based on their degree of similarity. A function decision was implemented to correctly group sentences from the similarity coefficient using a specified threshold level.

D. *Task Generation*

Task generation was identified as the process of breaking down a requirement into its most simplest form. Based on the criteria laid out in V-A, simplification on the existing input of requirements could be completed by deconstructing complex sentences into simple sentences.

The first process to break down sentences was to use parts-of-speech tagging to remove any unnecessary words from the sentence; this was done by identifying stopwords, since simple sentences contain only one subject and predicate. Therefore, using the word dependency we were able to determine which words were connected to the subject. By making sure the sentence contained one subject and verb we were able to create a complete simple sentence. Everytime a new subject was found that meant a new sentence was needed. Each simple sentence generated from the stories can then be suggested as tasks.

E. *Process Flow*

There is an overall linear flow, as seen in figure 4 to AI4Agile's main processing: Epic Decomposition, Story Optimization, and then Task Generation. Additionally, the relationship graph may be generated at any point and is independent of the previously mentioned flow. It is possible that the starting point in the flow may vary between uses. For example, a user may have manually entered in all their overarching user stories for their epic, forgoing the use of the Epic Decomposition process. From these entered stories, either story optimization or task generation may be performed.

Placeholder cause I need to reference figure 5

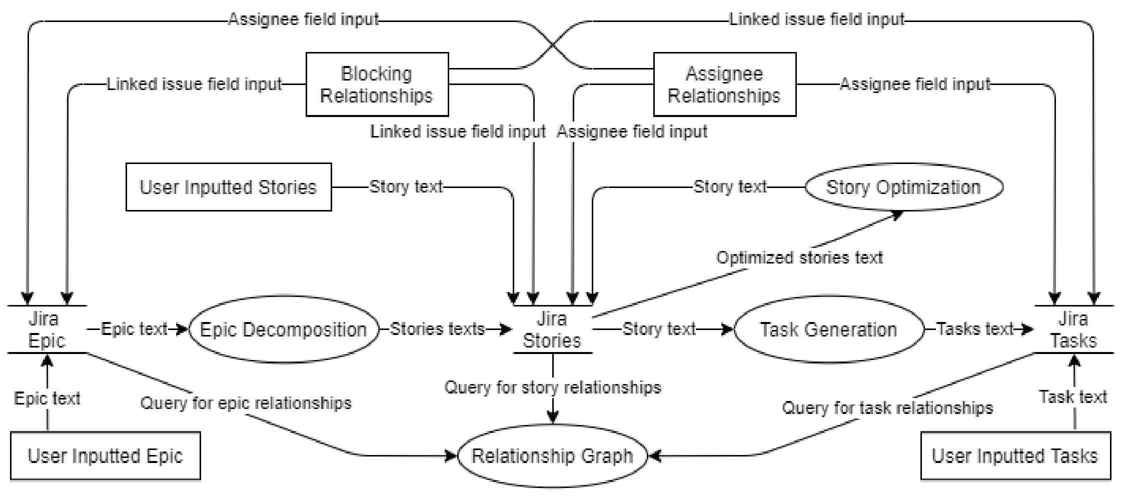


Fig. 5. Data Flow diagram showing the various entry points, sources of data, and processes within Jira and AI4Agile

F. *Relationship visualization*

The purpose of the relationship graph is to give users a visual representation of the dependencies between a selected issue in Jira and the issues that are related to it. Currently in Jira, hierarchies are shown minimally in such ways as dropdown lists from epics, or lists within an issue selection pane of either children or explicitly specified blocking/cloning/similarity relationships. This graph provides a way to view those relationships in one place at a glance, for ease of visual understanding.

G. *User Interface and Experience*

For the user interface and experience, a combination of prototyping and stakeholder meetings went into the decisions displayed in Table IV. The highest priority was to make the interface as intuitive and easy to use as possible, to fall in line with the larger project goal of saving users time throughout the agile development process.

VI. Implementation

The AI4Agile app was implemented within Jira Cloud as an app and connected to a Python-based backend.

A. *Frontend*

As part of the Atlassian Design Guidelines, a frontend library, Atlassian User Interface (AUI), is provided to create new UI elements to match Jira's style and user experience. Needed components that were missing from the AUI were crafted by the provided Design Guidelines from Atlassian.

TABLE IV
UI AND UX DESIGN CHOICES

Options	Choice	Reasoning
• Suggestion Format • Chatbot	Suggestion format	A conversation element can slow down the process and ease of use. A suggestion format was chosen to avoid the case where using the plugin as one who is familiar with the project and decomposition methods would slow down a decomposition process instead of accelerating it.
• Templated epic input • Unrestricted epic input • Semi-structured epic input via guidelines	Semi-structured epic input via guidelines	A template was deemed too restrictive and time consuming for the user, but unrestricted input was infeasible for AI processing. Thus, the choice was to present the user with guidelines in the user manual for what the optimal structure of an epic would be for use with this plugin.
• External database use • Strictly Jira database use	Strictly Jira database use	There is a need to split up the different types of sentences, such as complex and compound, into simple sentences, in order for them to be manipulated into tasks. To do this, sentence classification is the necessary first step.
• Relationship Graph as tree • Relationship graph as clusters • Relationship graph as tree and clusters	Relationship graph as tree and clusters	While the original concept had the relationships as clusters, it was determined from stakeholder input that a tree format would likely be more intuitive to the user base. At the same time, if the relationships to be shown are only developer assignments, the lack of parent-child relationships suggested clustering was still the sensible choice.
• Undo functionality for story/task creation • Preview button for story/task creation • Iterative creation process	Iterative creation process	With undo functionality, there were issues such as where it would make sense to have such a thing, how far after the action it should be able to be undone, and how much additional effort would be needed to implement it. The reasoning behind this feature in general was for new users to be able to experiment with the plugin without fear of consequences. It was determined that switching to an iterative process, where users can create one story or task at a time from the suggestions, would serve this same purpose to lessen potential consequences and still allow users to experiment. As a result of the iterative creation process, the need for a preview system was eliminated.

Within Jira, all objects (epics, stories, issues) are rendered in the same way, each containing a shared issue panel as shown in figure 6. Our app is implemented within the issue panel in the form of additional panels that can be opened through newly introduced buttons to the primary button set. When a process button or visualization button is pressed, a new panel, built with HTML and javascript, will render and be focused on.

Each of the processes use the same suggestion panel, with variations depending on the need for user inputs in the case of epic decomposition and story optimization. The suggestion module, when opened, requests suggestions from the backend through jQuery AJAX and displays a loading animation until the suggestions are received. Stories can be changed and once checked, they can be created which is done by passing the selected stories via an AJAX call to the backend. The responsibility to populate selected stories to the Jira board is by the backend.

B. Backend

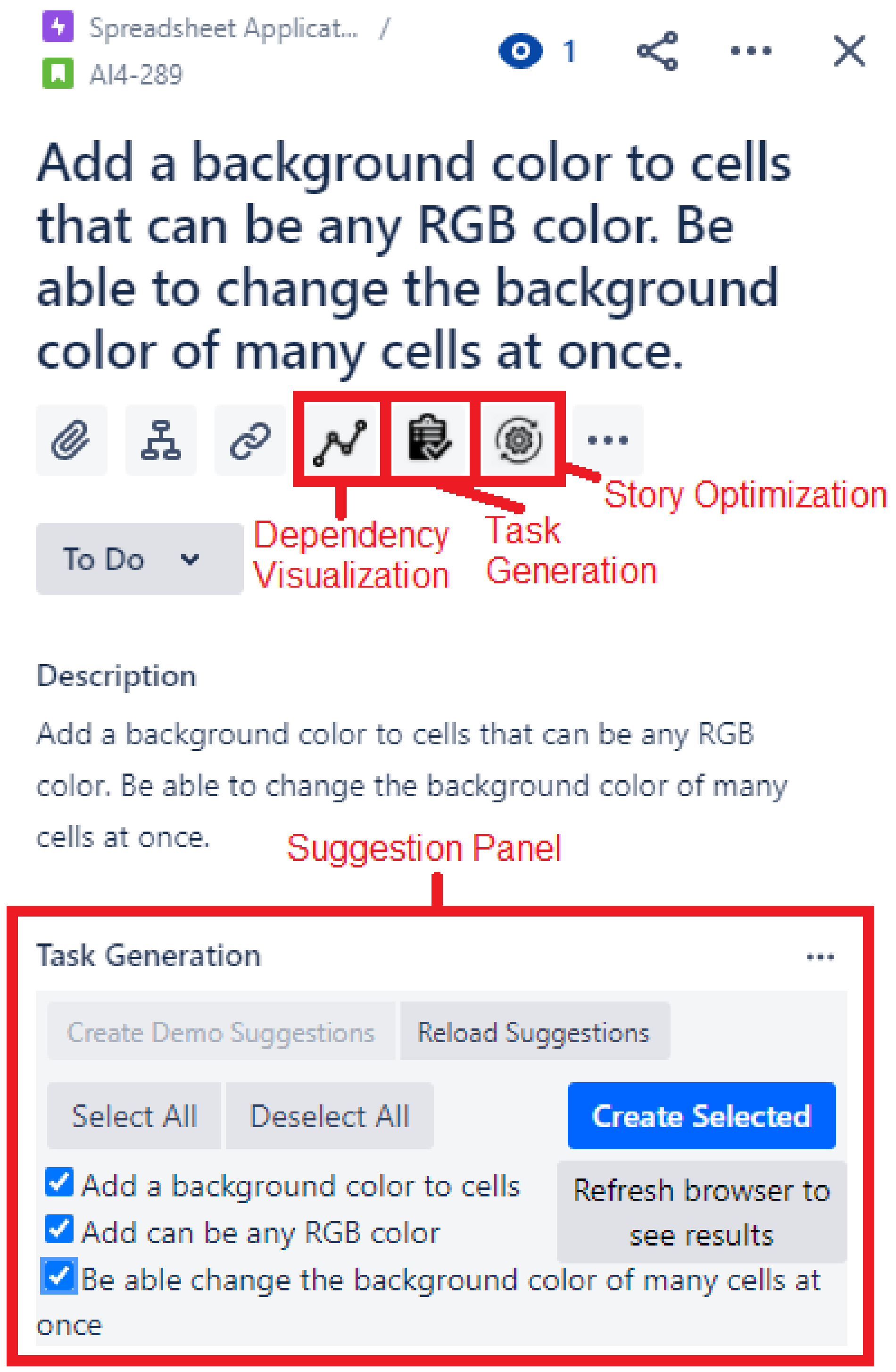
The backend served two primary functionalities: generating suggested issues and populating selected suggestions into Jira. Each process has its corresponding suggestion generator and selected suggestion creator since the specifics of each type of issue varies. Jira treats all issue types (i.e. epics, stories, and tasks) the same, each having the same set of shared fields with the addition of custom fields depending on the type of issue.

All three processes were implemented using Python due to the available libraries and as a result the primary interface was also implemented using Python. The interface uses Flask, a Python web framework, to receive the messages sent by AJAX in the frontend. The information needed from the issues are gathered by querying the Atlassian Rest API through an open-source Python library, "atlassian-Python-api", which simplifies Atlassian API calls. When generating suggestions, the description is queried for and passed as a list of individual requirements to the text processor. Suggestions are then passed back through a reply to the original message. Once a user selects, and possibly edits, which issues they would like to have populated in their Jira board, then the original issue is queried again to percolate existing fields, such as assignee, reporter, due date, tags, or sprints, to the newly created child issues. Each newly created issue is given a blocking relationship to the issue they were derived from. For example, all of the created stories from an Epic Decomposition will "block" the epic from being completed. Any further tasks created from Task Generation on a story will "block" the story from completion.

1) Epic Decomposition: The Epic Decomposition process was implemented using 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.

2) Story Optimization: For Story Optimization, the process maintained the initial implementation without any significant changes to the algorithm. The algorithm utilizes a pre-trained Word2Vec model, along with SIF and cosine similarity functions from the scikit-learn python library. Initially, the degree of connectivity between sentences was specified by choosing the optimal threshold level. However, it was changed so that the user can choose to increase or decrease the degree of connectivity between sentences offering more flexibility in the generated results.

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.



Spreadsheet Applicat... /

AI4-289

Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once.

Description

Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once.

Task Generation

- [x] Add a background color to cells
- [x] Add can be any RGB color
- [x] Be able change the background color of many cells at once

Refresh browser to see results

Fig. 6. Issue View within Jira while the Task Generation process shown after suggestions are recieved

TABLE V
COMPONENTS, SUBCOMPONENTS, AND SOURCES USED FOR IMPLEMENTATION OF SUBCOMPONENTS

Component	Subcomponent	Source
Epic Decomposition	K-means clustering	scikit-learn: machine learning in Python
Story Optimization	Word embedding	Google Word2Vec Pre-trained Model
	Feature Extraction	scikit-learn: machine learning in Python
	Cosine similarity	scikit-learn: machine learning in Python
	Function decision	N/A
Task Generation	Sentence classification	Stanford CoreNLP
	Part of speech tagging	Natural Language ToolKit
UI	Tree relationship graph	Cytoscape
	Clustered relationship graph	Cytoscape
	Jira Cloud frontend	Atlassian Connect Express Modules

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 inputs.

3) Task Generation: The approach for Task Generation was based off a decomposition approach by Das, Majumder, and Phadikar in 2018 [3]. Initially the natural language toolkit (NLTK) library was selected for implementing task generation, but it lacked the need for parts-of-speech tagging. 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 parts-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.

C. Communication

The plugin communicates from Jira to the backend interface via a listener through AJAX calls which are received by Flask. When a web panel is opened for any of the three processes, a message is sent to generate and return the suggestion issues. The Jira panel will function asynchronously with a loading animation appearing until the resulting suggestions are received. Once the results are received, populated in the suggestion box, and the desired suggestions are selected by the user, then a message is sent containing which suggestions to then create within Jira.

VII. VERIFICATION AND VALIDATION

A. Demonstration

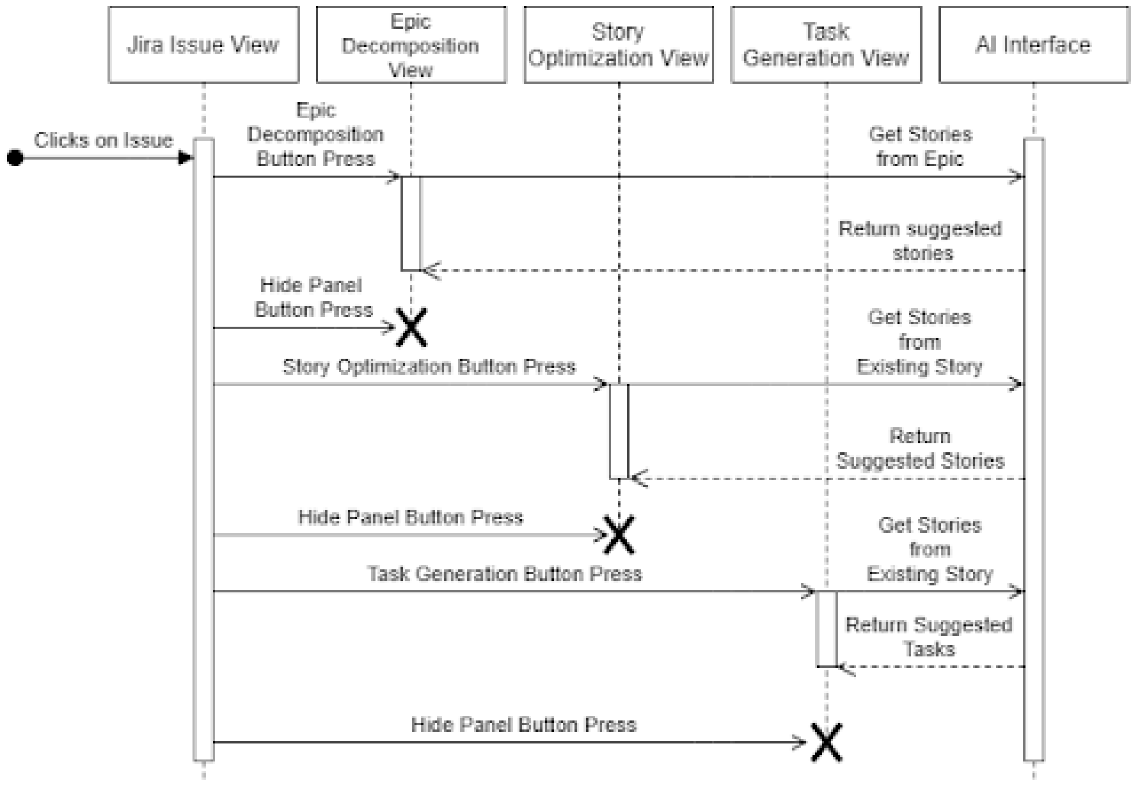


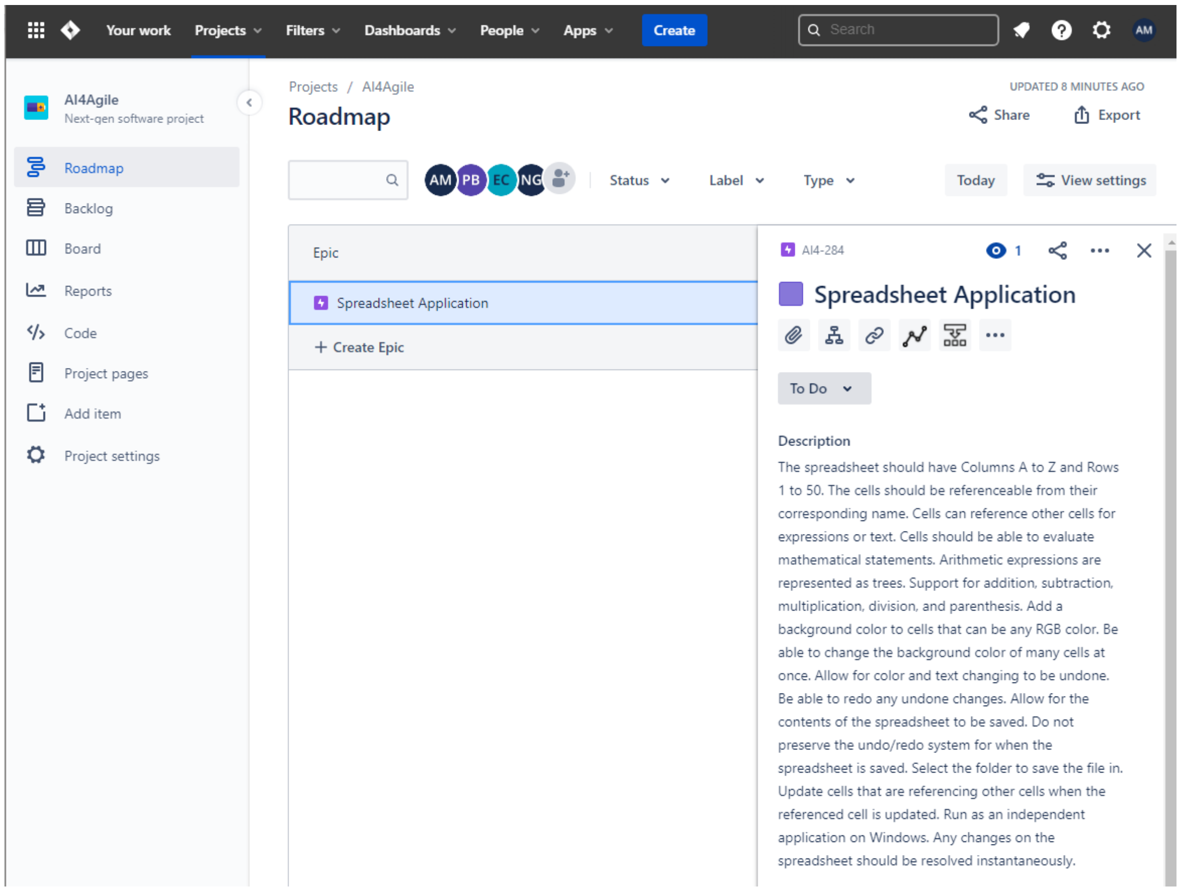
Fig. 7. Suggestion web panel sequence diagram representing the flow of messages between the frontend Jira web page and AI4Agile backend

1) Complete Decomposition Scenario: Frank is a software requirements analyst, and he wants to speed up the process of breaking an epic full of requirements into smaller user stories. For this purpose, he installs the AI4Agile plugin to Jira. Frank creates a new epic, puts the requirements for his Spreadsheet Application in as plaintext sentences, and clicks the Decompose Epic button (fig. 8).

Now, Frank looks at the suggestions the AI came up with (fig. 9). If he decides he doesn't like any of these suggestions, he can click the Ignore All button to go back to his Spreadsheet Application epic. If Frank wants more or fewer stories, he can adjust the value on the slider and it will refresh the results. To edit a story, he can click on its text box, then save or cancel those edits. To ignore a story suggestion entirely, he can leave its box unchecked. Once Frank is happy with his resulting user story or the set of stories he wants, he clicks Create Selected.

After refreshing the webpage, the newly created user stories that Frank approved are visible under the heading of the Spreadsheet Application epic.

If Frank wants to continue the process of breaking up epics, he can go into one of the user



The spreadsheet should have Columns A to Z and Rows 1 to 50. The cells should be referenceable from their corresponding name. Cells can reference other cells for expressions or text. Cells should be able to evaluate mathematical statements. Arithmetic expressions are represented as trees. Support for addition, subtraction, multiplication, division, and parenthesis. Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once. Allow for color and text changing to be undone. Be able to redo any undone changes. Allow for the contents of the spreadsheet to be saved. Do not preserve the undo/redo system for when the spreadsheet is saved. Select the folder to save the file in. Update cells that are referencing other cells when the referenced cell is updated. Run as an independent application on Windows. Any changes on the spreadsheet should be resolved instantaneously.

Fig. 8. Overall view of Jira Roadmap with Decompose Epic button in focus

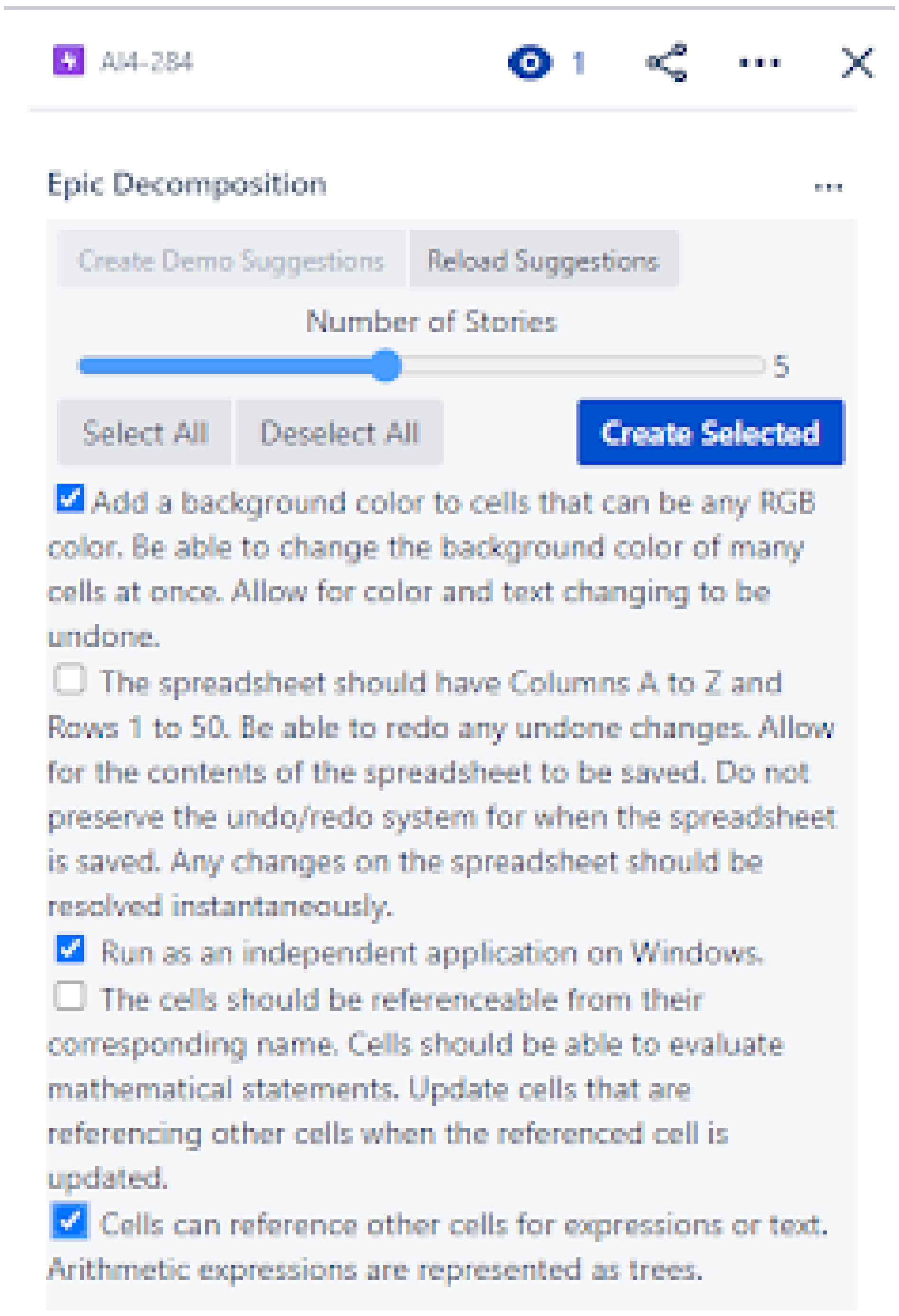
stories and click Optimize User Story (fig. 10). Depending on the size of the user story already, it might be broken down into multiple smaller stories, or left alone if it's already optimized.

Once the Story Optimization Suggestions are generated, Frank has the same options as with the previous stories: to select or deselect stories via the checkboxes, make edits, or ignore all suggestions. In addition to those options, Frank can choose to adjust the connectivity slider to change how closely related items need to be in order to stay with the same story.

Now there are five user stories, since Story 4 was optimized into two separate stories. To continue the decomposition process, Frank opens a story and clicks Generate Tasks (fig. 9).

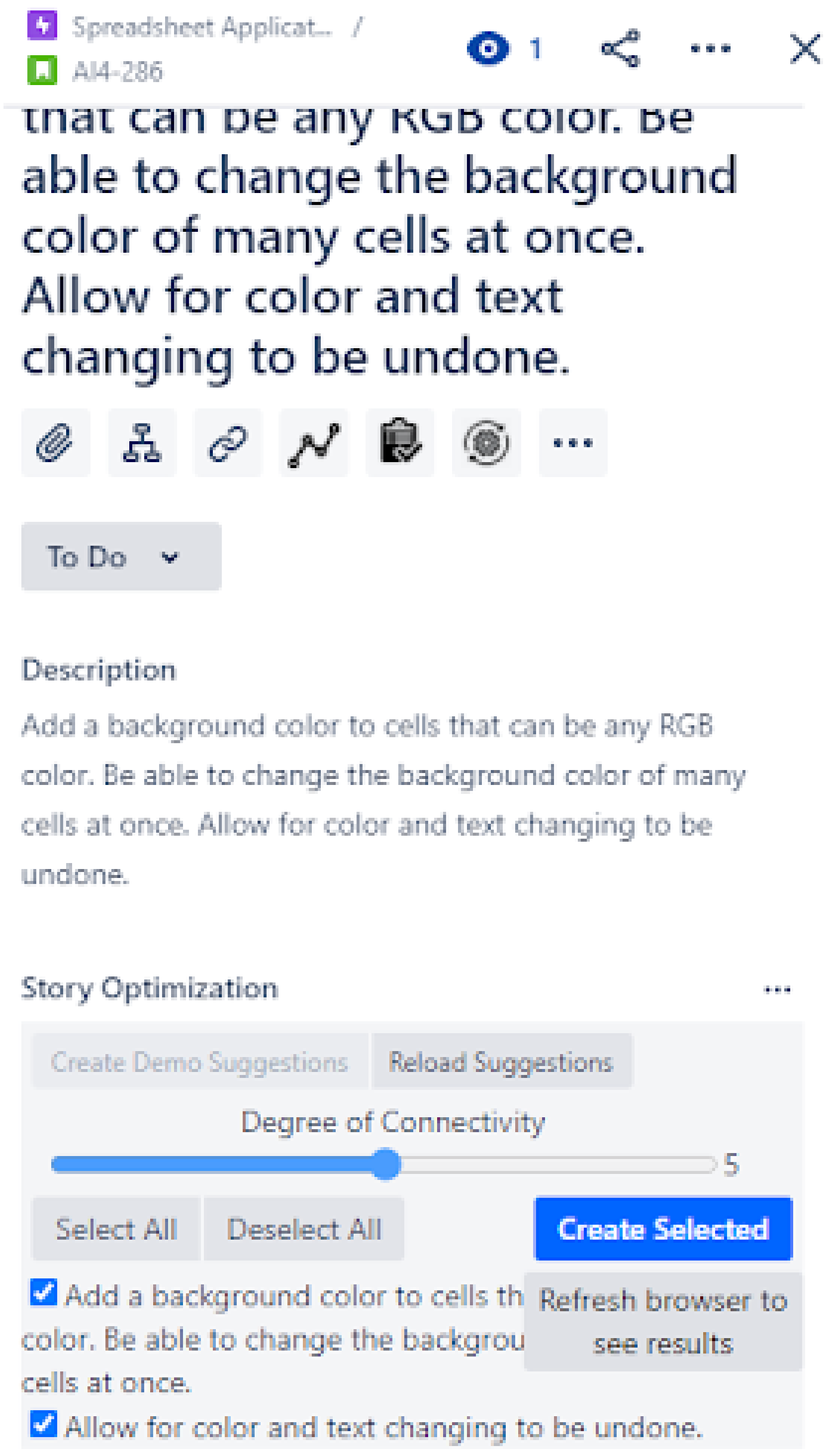
As before, the options include selecting or deselecting individual suggestions, editing, and ignoring all suggestions. Once he's happy with the tasks, he clicks Create Selected.

The tasks have now been created, and linked to their parent story to indicate a blocking relationship. All Frank had to do was make some decisions and maybe edits, and now he's got one story fully decomposed (fig.).



- [✓] Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once. Allow for color and text changing to be undone.
- [ ] The spreadsheet should have Columns A to Z and Rows 1 to 50. Be able to redo any undone changes. Allow for the contents of the spreadsheet to be saved. Do not preserve the undo/redo system for when the spreadsheet is saved. Any changes on the spreadsheet should be resolved instantaneously.
- [✓] Run as an independent application on Windows.
- [ ] The cells should be referenceable from their corresponding name. Cells should be able to evaluate mathematical statements. Update cells that are referencing other cells when the referenced cell is updated.
- [✓] Cells can reference other cells for expressions or text. Arithmetic expressions are represented as trees.

Fig. 9. View of generated User Story Suggestions



Spreadsheet Applicat... /
AI4-286

that can be any RGB color. Be able to change the background color of many cells at once. Allow for color and text changing to be undone.

To Do

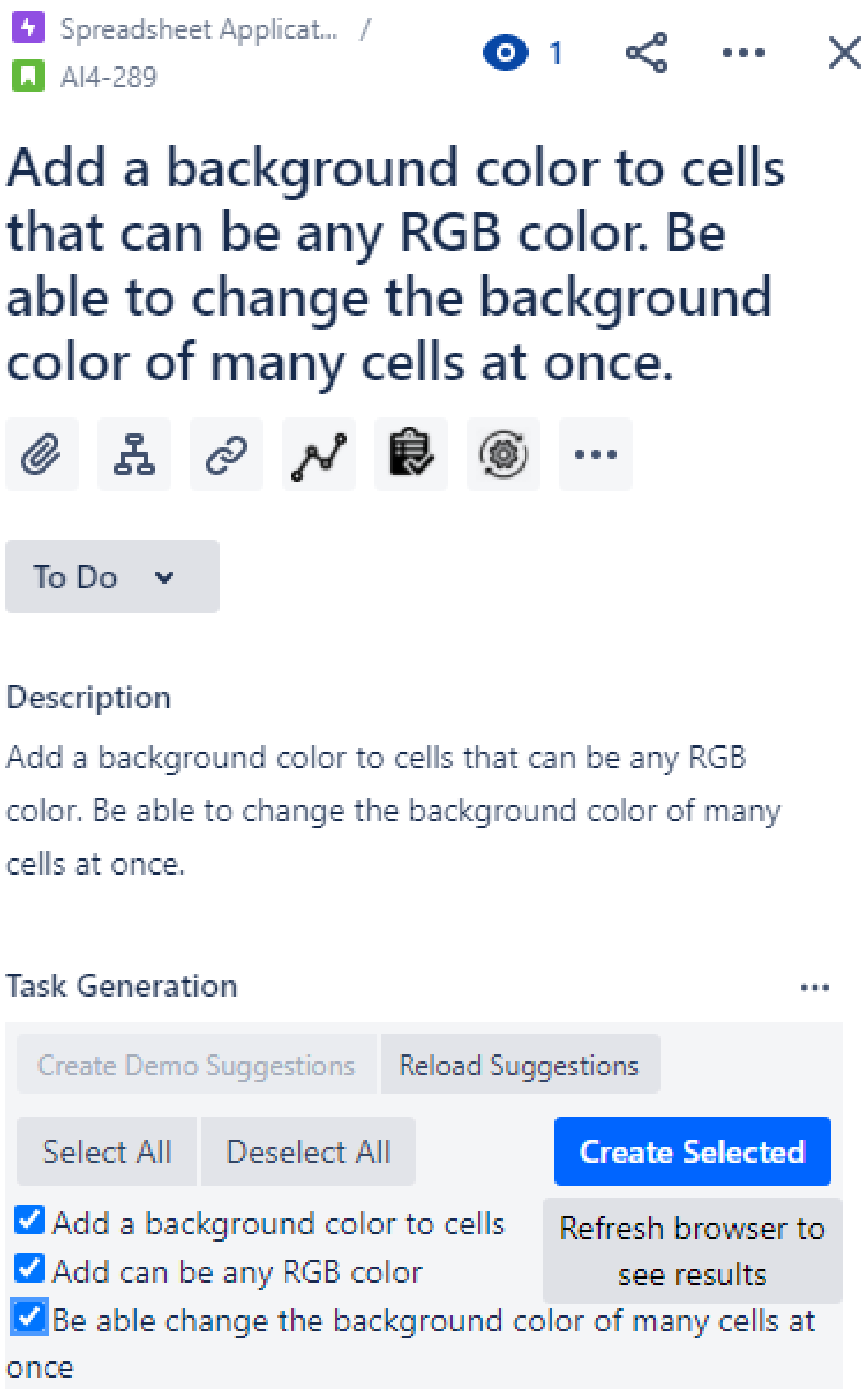
Description

Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once. Allow for color and text changing to be undone.

Story Optimization

☑ Add a background color to cells th color. Be able to change the backgrou cells at once.
☑ Allow for color and text changing to be undone.

Fig. 10. User Story Optimization Suggestions page


Spreadsheet Applicat... /
AI4-289

1

Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once.

To Do

Description

Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once.

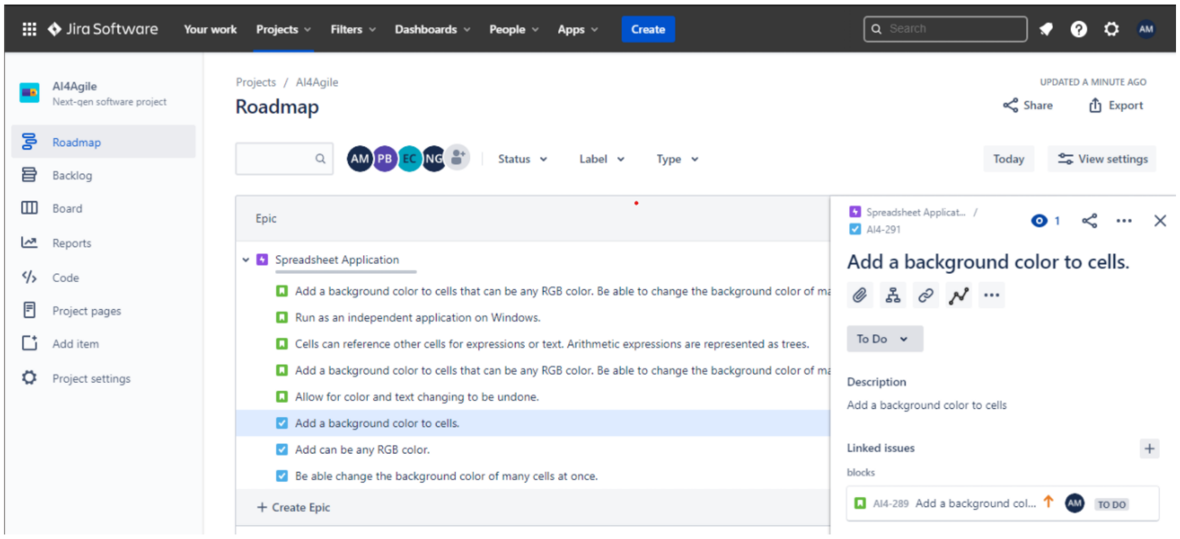
Task Generation

Create Demo Suggestions
Reload Suggestions
Select All
Deselect All
Create Selected

- [x] Add a background color to cells
- [x] Add can be any RGB color
- [x] Be able change the background color of many cells at once

Refresh browser to see results

Fig. 11. Task Suggestions page



Projects / AI4Agile

Roadmap

Epic

Spreadsheet Application

- Add a background color to cells that can be any RGB color. Be able to change the background color of ma
- Run as an independent application on Windows.
- Cells can reference other cells for expressions or text. Arithmetic expressions are represented as trees.
- Add a background color to cells that can be any RGB color. Be able to change the background color of ma
- Allow for color and text changing to be undone.
- Add a background color to cells.
- Add can be any RGB color.
- Be able change the background color of many cells at once.

Spreadsheet Applicat... / AI4-291

Add a background color to cells.

To Do

Description

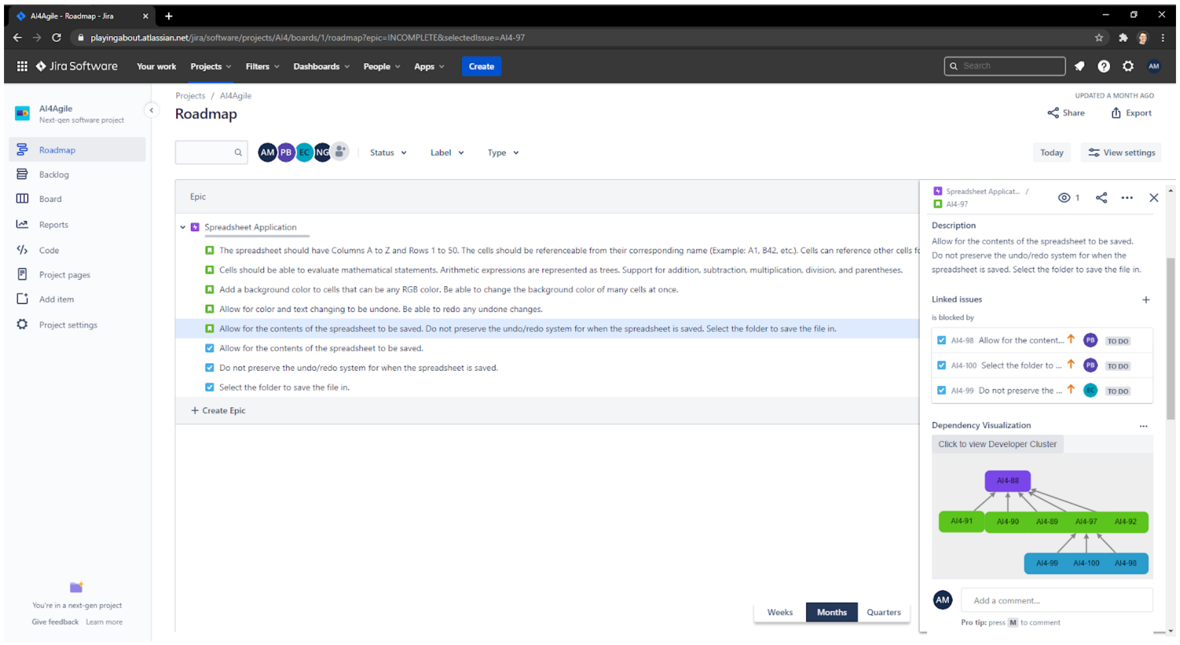
Add a background color to cells

Linked issues

blocks

AI4-289 Add a background col... TO DO

Fig. 12. View of newly created tasks for one fully decomposed user story



playingabout.atlassian.net/jira/software/projects/AI4/boards/1/roadmap?epic=INCOMPLETE&selectedIssue=AI4-97

Projects / AI4Agile

Roadmap

Epic

Spreadsheet Application

- The spreadsheet should have Columns A to Z and Rows 1 to 50. The cells should be referenceable from their corresponding name (Example: A1, B42, etc.). Cells can reference other cells f
- Cells should be able to evaluate mathematical statements. Arithmetic expressions are represented as trees. Support for addition, subtraction, multiplication, division, and parentheses.
- Add a background color to cells that can be any RGB color. Be able to change the background color of many cells at once.
- Allow for color and text changing to be undone. Be able to redo any undone changes.
- Allow for the contents of the spreadsheet to be saved. Do not preserve the undo/redo system for when the spreadsheet is saved. Select the folder to save the file in.
- Allow for the contents of the spreadsheet to be saved.
- Do not preserve the undo/redo system for when the spreadsheet is saved.
- Select the folder to save the file in.

Spreadsheet Applicat... / AI4-97

Description

Allow for the contents of the spreadsheet to be saved. Do not preserve the undo/redo system for when the spreadsheet is saved. Select the folder to save the file in.

Linked issues

is blocked by

AI4-98 Allow for the content... TO DO

AI4-100 Select the folder to ... TO DO

AI4-99 Do not preserve the ... TO DO

Dependency Visualization

$$\begin{tikzcd}
 & \text{AI4-88} & & & \\
\text{AI4-91} \arrow[ur] & \text{AI4-90} \arrow[u] & \text{AI4-89} \arrow[ul] & \text{AI4-97} \arrow[ull] & \text{AI4-92} \arrow[ulll] \\
 & & \text{AI4-99} \arrow[ur] & \text{AI4-100} \arrow[u] & \text{AI4-98} \arrow[ul]
\end{tikzcd}$$

Fig. 13. A zoomed out view of navigating to the issue relationship graph

2) Relationship Visualization Scenario: Madison is a scrum master, and she wants to take a closer look at some of the tasks to make sure the developer assignments make sense. To do this, she uses the AI4Agile Issue Relationship Graph feature by navigating to a certain epic, user story, or task, and finding the graph portion (fig. 13). Now, Madison can see all the available relationships between the task she has selected and other tasks, stories, and the epic they came from. She is better able to decide whether the team members she assigned to these pieces make sense given the relations between them.

For an alternate view of the relationships, Madison can use the Click to view Developer Cluster button (fig. 13) to see what the current assignments and workload look like for her team (fig. 14). If she wants to go back to the previous graph view, the Click to view Issue Tree button is in the top left corner.

VIII. Verification and Validation

A. UI/UX Feedback

The application is primarily composed of individual web panels integrated into Jira's existing User Interface. As a result, each web panel was able to be created, tested, and reviewed independent from Jira Cloud. The isolation was used to conduct input and output testing but the modules were generally tested within the context of Jira to ensure that the user experience is seamless between our integrated app and the existing Jira Cloud platform.

B. Testing

Testing was performed on a component and integration level. Component testing was separated into two categories: text processing and relationship visualization.

Text processing component level testing was based upon results from three categories of inputs: ideal, disjoint, and semi-ideal. Ideal inputs were those where all information belonged to clear categories. Disjoint inputs were those where all information was completely separate each with separate categories. Semi-ideal inputs had both outlier pieces of information and information that distinctly can be clustered with other info. Disjoint inputs were used to evaluate edge case usages of the app. To represent the average case, semi-ideal inputs were used since natural language understanding from a vector perspective is ambiguous without context.

At an integration level, all possible user paths were explored as each text process could be done independently or sequentially. For example, a user can complete the entire process by entering by decomposing an epic, optimizing the generated stories, and then generate tasks from the optimized stories. However, a user may choose to only optimize stories that were manually entered and then go on to generate tasks from the optimized stories.

IX. Future Works

Now that this project is beyond the initial competition's timeline and restraints, the team can explore a new scope as the focus shifts towards viewing the project more heavily through the lens of being an academic capstone, without the prompt as the primary guide. The team has so far

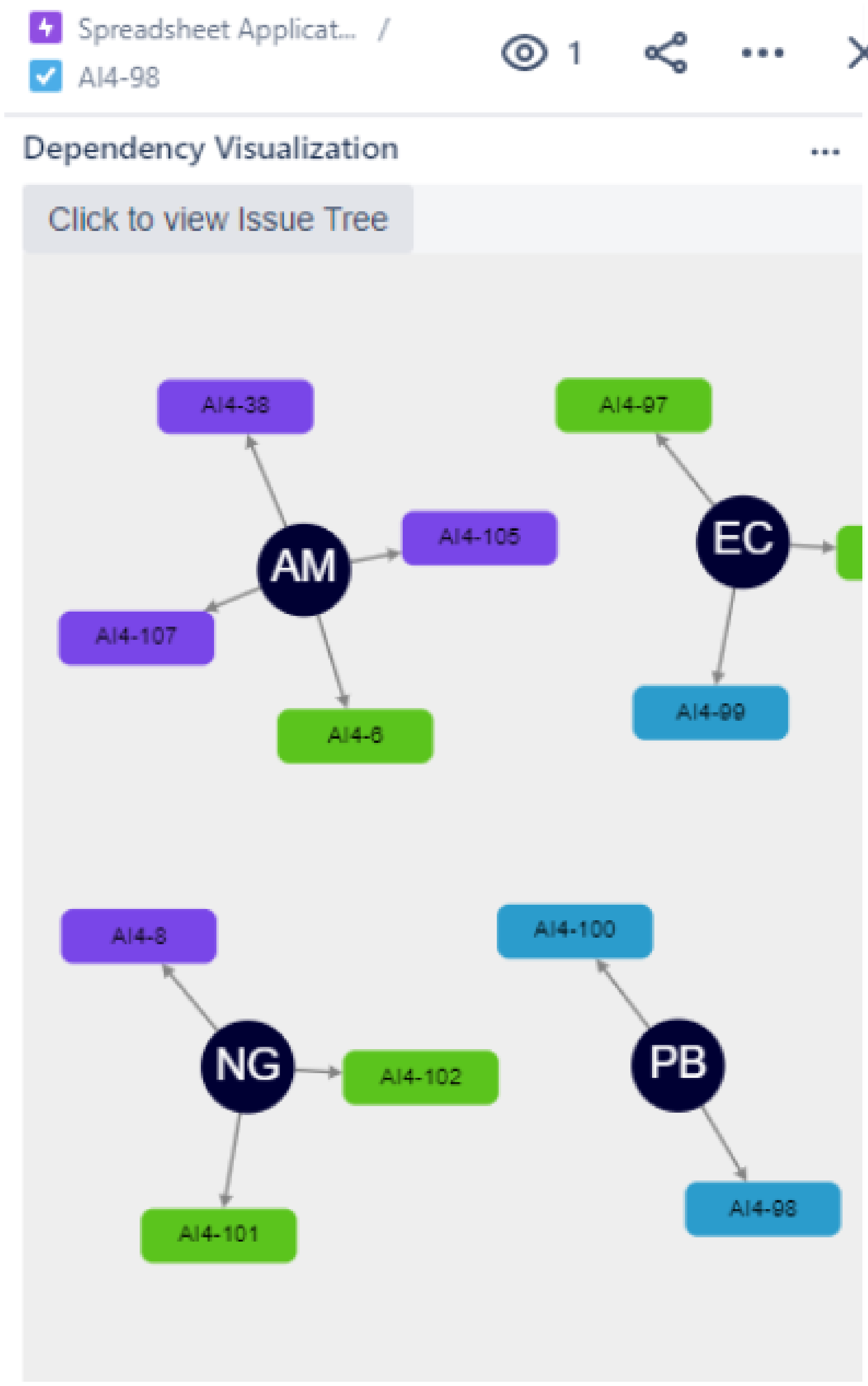


Fig. 14. Developer cluster graph on a selected task

defined a vision with loose deliverables in mind for this next time period, with the intent that these things be better defined upon return from the winter development break and with the finding of real customers. In further detail below are the plans for this, including the specific deliverables.

During the second phase of development, the project 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 as elicited at the beginning of the second phase. Despite the dependence on customer inputs, general guidelines have been predetermined for areas or features that would most benefit from improvement or creation. For epic decomposition, that improvement is in the form of aiming for dynamic clustering of stories, as opposed to the current clustering with a slider for the user to adjust if the amount of stories doesn't fit well with the size of the given epic. The main improvement for story optimization would be in optimizing the algorithm to yield results faster, as the current wait time is high relative to the team's goal times. The final AI component improvement would be for task generation, in the form of switching from the current supporting package in Java to the Stanza Natural Language Processing package in Python. This switch would aid in improving the results from task generation, especially in lessening the outputting of sentence fragments instead of complete sentences, due to a word dependency feature within the Python package. The switch comes as a result of research presented in "A Novel System for Generating Simple Sentences from Complex and Compound Sentences." These AI improvements are centered on the user experience, as long wait times or results that need multiple adjustments to be used can decrease the time benefits of using the application's refinement support over the manual method. The relationship visualization component has a variety of ways it could be improved, in ways such as providing more relationship options to be shown, filtering options to cover those choices so they don't all need to be displayed at once, and ease-of-use controls for zooming and panning in the form of on-screen buttons instead of implicit mouse controls. All of these deliverables are potential plans for the upcoming project period, but their approval or further details will depend heavily on the results of the customer focus shift.

X. Conclusion

This document is used to describe the actions taken to complete the SCORE 2021 AI4Agile prompt on the Jira Cloud platform by building a plugin using the interface and development tools that the parent company, Atlassian, provided. The new plugin performed epic decomposition, story optimization, task generation, and relationship visualization for various forms of user stories. To implement these new features on Jira Cloud, research was done on how Jira Cloud worked, and how the AI components should function and interface with Jira. Determining the best area to implement the new features was important, because the team wanted somewhere the user frequently visits and which can be seen easily. Section 6 of this paper went into detail on the specific area chosen and its reasoning.

The team for this project consisted of 7 people, including mentors and advisors, where only the faculty advisor had previous knowledge of AI, therefore the rest of the team members had a steep learning curve. Due to the wide range of AI types, research needed to be done to determine