AI4Agile Summary Report - Team Katara

Phong Bach, Emily Cawlfield, Nain Galvan, Aric Monary
School of Electrical Engineering and Computer Science
Washington State University
Everett, USA 98201
phong.bach, emily.cawlfield, nain.galvan, aric.monary@wsu.edu

I. Introduction

AI4Agile is an add-on application built for Jira Software Cloud [3], made to assist software development teams in agile enviroments with user story streamlining and refinement. The application utilizes Artificial Intelligence (AI) to power three major processes that refine user stories from epics into tasks. It offers AI-based smart suggestions of requirement work items with different levels of details: epics, user stories, and tasks. Finally, the application provides an integrated tool that allows development teams to visualize the explicit and implicit relationships among the requirement work items. Team Katara from Washington State University Everett submits this report to meet the requirement of the ICSE SCORE 2021 competition.

The rest of the report is organized as follows. Section II outlines preliminary background information of the app, the project stakeholders, and the team's processes working on the project. Section III summarizes the decomposition of requirements into goals, and further into functional and non-functional specifications. Section IV and V outline the design and implementation of the tool respectively, while the latter also expanded on detailed differences between final implementation and the initial design. Section VI demonstrates the primary functionalities of the tool via walkthroughs of two use cases, as well as the steps done for testing. Lastly, Section VII reflects on the outcome of the project, as well as explores future prospects of the project.

II. Background and Process

A. Project Scope

The team chose the project "AI4Agile: Developing AI-enabled tool support for user story refinement in JIRA" from SCORE 2021's project list to participate in the competition. The project is sponsored by Dr. Hoa Khanh Dam from Univeristy of Wollongong in Australia. It stemmed from Dr. Dam's research in the intersection of Software Engineering and Artificial Intelligence [16]–[18].

The project proposed four primary requirements [1]:

1) Recommend user stories that are decomposed from an epic.
2) Recommend smaller user stories to be 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, the application must be integrated and easily deployed into the platform of Atlassian's Jira software, a widely used software project management tool that provides a marketplace for add-on applications [2].

B. Teams and Stakeholders

We entered the SCORE competition upon recommendation by our faculty advisor Dr. Bolong Zeng. The four team members who participate in the competition are all senior students in the Software Engineering major at Washington State University, Everett, USA. We complete the project as part of our Software Requirement and Senior Design Capstone courses. [1]

Among the list of projects, the team chose AI4Agile since the team were interested in working with AI for the first time. The idea of intersecting AI with Software Engineering (SE) was also intriguing, particular for four Software Engineering major students. The project is essentially a meta-project on SE as we would be developing an SE tool that helps with the SE process, while conducting our own SE process.

The team enlisted the help of several faculty and industry mentors/advisors. Dr. Zeng is the faculty advisor who supervises the team's process and review our work products. As the team has no prior experience with AI and Natural Language Processing (NLP), we ask for advice from Prof. Jeremy Thompson, whose main expertise is in these fields. We also contacted Dr. Dam, the SCORE sponsor as soon as the team started our work. Dr. Dam provided us with his previous research and publications, from which AI4Agile was born, as well as sample data that he used in the past.

As the project is a tool for software developers, we were fortunate enough to have a senior project manager, Skip Baccus, from Microsoft who agreed to fulfill the role as a domain expert. Throughout the project, he works with us as both a mentor and a stand-in for potential client. All people mentioned above rounded up the stakeholders in the project. Team member Aric Monary served as a team liaison to coordinate communications among all involved parties.

C. Team Process

The team adopted the agile process for the project. The team met a minimum of 2 times per week to go over documentation and the implementation work on the project. These meetings led into the scrum meeting with our business mentor every Tuesday. Each time a new component was implemented, we debrief the mentor during the scrum to solicit feedback from his perspective. In between these meetings, the team worked on their own assigned tasks, while the faculty advisor reviewed our documentations. Figure 1 shows the major activities of the team on a weekly basis.

The team started by studying on Dr. Dam's paper, mainly [18], and brainstorming concepts that we think are important to the project. The team then split into doing research on various key issues, such as Jira API, NLP, etc. based on individual strengths and interests. Meanwhile, the entire team participated in regular requirement solicitation with the mentor to refine the intial scope in the project proposal. Throughout the process we obtain key understanding on agile process itself, and

[1] As part of this year's theme of our Senior Design Capstone projects (Avatar: the Last Airbender), the team is named Team Katara.

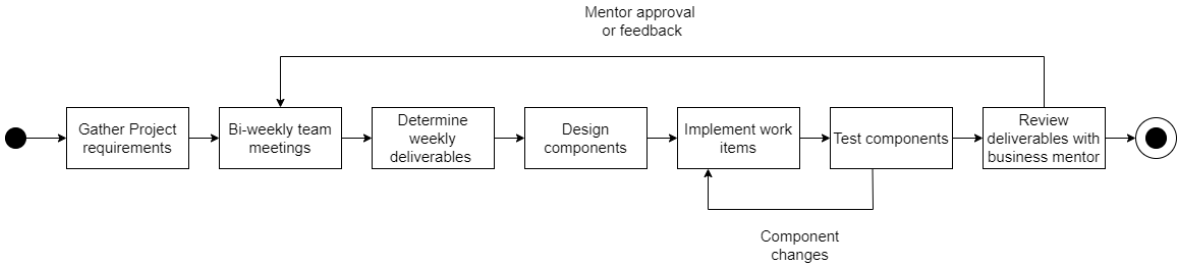


Fig. 1: Activity Diagram outlining the SE processes

how the tool can best assist within the project scope. The requirements are presented in more details in Section III. As we were finalizing major features to implement, the team simultaneously started early coding to experiment with implementation based on our earlier research.

The development effort mainly consists of two parts: AI and UI components. The AI components were of high priority through majority of the project. Three members each focused on one of the three primary requirements needed for the AI, while the fourth member focused on building UI for the visualization tool. As the project came to a close, we moved the UI portion as top priority while AI development was completed. As our final milestone, we demonstrated the functional prototype of the tool to our stakeholders by conducting walkthroughs of major use cases. Figure 2 presents the project timeline, with key work tasks and dates noted. Figure 3 provides an overview of responsibility assignments for functionalities of the add-on.

III. Requirements

A. Preliminary Analysis

We first performed analysis on the given proposal in Section II-A. On the infrastruture side, the proposal requires the project to be integrated into Jira. Among the different versions of Jira software available, the team chose Jira Cloud over Jira Server, for its cloud-based nature is more compatible in modern workplace.

We defined the key concepts as given in the primary requirements. For the purpose of our project, we consider **Epic** to be a collection or narrative of user stories; they contain more information as to the functional usage of a target project being documented. The next level is **User Story**, which is a compact set of requirements. These requirements can be furthered broken down into **Tasks**, actionable item to be completed as part of a user story. We consider Epic, User Story, and Tasks as the three levels of requirements items that our tool focuses on.

B. Requirement Modeling with KAOS

First, the team modeled the requirements through the use of KAOS models [21]. KAOS is a Goal-oriented requirement engineering [19] tool that emphasizes goals in requirement acquisition. The team built the KAOS models to build thorough traceabilities within project requirements, which includes describing the problem to be solved, identifying top level goals to address problems, the constraint it must be fulfilled by solutions, etc. Moreover, using the KAOS models, the team could

Section	Task		Start Date	Due Date	Assignee
Planning and Proof of Concept	Component Research		8/31	9/7	Phong, Emily, Nain, and Aric
	Prototype Development		9/7	9/20	
	Prototype Demo		9/20		
	Component Document Draft		8/31	9/20	
Development	Infrastructure	Component Vision Document	9/18	9/25	
		Testing Text Preprocessing	9/25	10/2	
		Compile Training/Test Data	9/25	10/2	Phong, Nain, and Aric
	AI	AI Model	10/2	10/16	Phong, Nain, and Aric
		AI Validation Testing Round 1	10/16	10/23	
		Component Vision Revision	10/16	10/23	
		Requirement Validation	10/16	10/23	
		AI Refinement	10/23	11/06	
		AI Validation Testing Round 2	11/06	11/13	
		Final AI Refinement/Testing	11/13	11/20	
	Visualization	UI/UX Setup	9/18	9/25	Emily
		Validation Component	9/25	10/2	
		UI Validation Walkthrough	10/6		
		UI Refinement	10/6	10/13	
		Suggestion System	10/20	11/3	
		Issue Relationship Clustering	10/09	10/30	
		Issue Clustering Demo	11/03		
		UI/UX Validation Round 2	11/03		
		Final UI/UX Refinement	12/04	12/08	
Component Testing	Unit Testing		11/20	11/27	Phong, Emily, Nain, and Aric
	Requirement Validation				
Deployment, Integration, and Systems Testing	Test AI Flow		11/27	12/04	Phong, Nain, and Aric
	Test AI output into Jira				
	AI Validity				
	Accessibility				Emily
	UI/UX Refinement				
Prototype	Alpha Prototype Presentation		12/11		Phong, Emily, Nain, and Aric

Functionality		Responsibility
Epic Decomposition		Aric
Story Optimization		Phong
Task Generation		Nain
Dependency Visualization		Emily and Phong
User Interface	User Experience	Emily
	Primary Button Layout	Nain
	Suggestion Panel	Aric

Fig. 3: Function Responsibility Breakdown

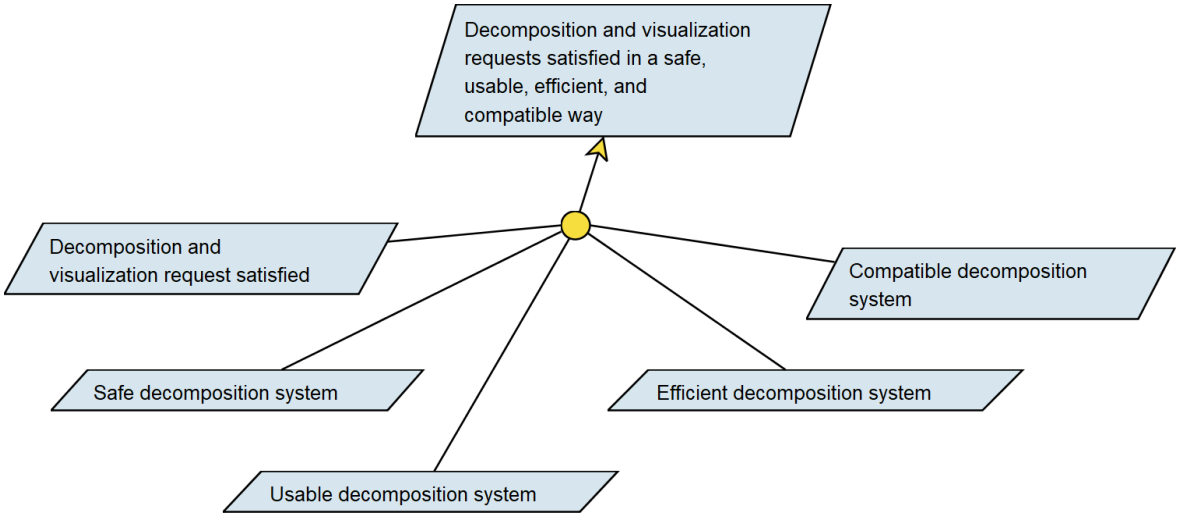


Fig. 4: Goal: "Decomposition requests satisfied in a secure, usable, efficient, compatible, and reliable way".

easily remove or add requirements as we encounter new information and changes. [21] provides universally applicable patterns for reference, and we specified our initial goals by adopting these generic patterns, which also guide the later requirement refinement.

We identify three primary processes to handle the requirements items defined above: 1) Epic Decomposition, 2) Story Optimization, and 3) Task Generation. Fig. 4 defines the top-level goal for the entire system, with subgoals that enumerate all the cases that must be covered to fulfill our main

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 generates important relationships
NFR12	Available on Atlassian Marketplace
NFR13	Available on Atlassian Partner Marketplace

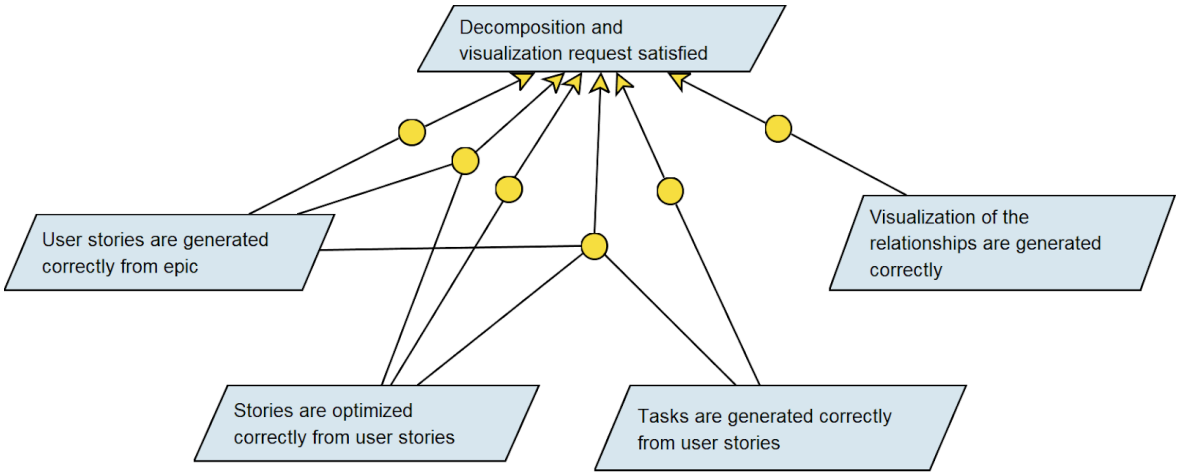


Fig. 5: Goal: "Decomposition request and visualization satisfied".

goal. The subgoals each encompasses a portion of the functional and nonfunctional requirements (FRs and NFRs) of the system. As shown in Fig. 4, the goals that our system should achieve are:

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

The goals for "Safe System", "Usable System", "Efficient System", and "Compatible System" covered the NFR portion of the system. By using respective generic patterns from [21], we generated the nonfunctional requirements for our system as shown in Table I.

Next, we broke down the "decomposition request and visualizations satisfied" goal into subgoals,

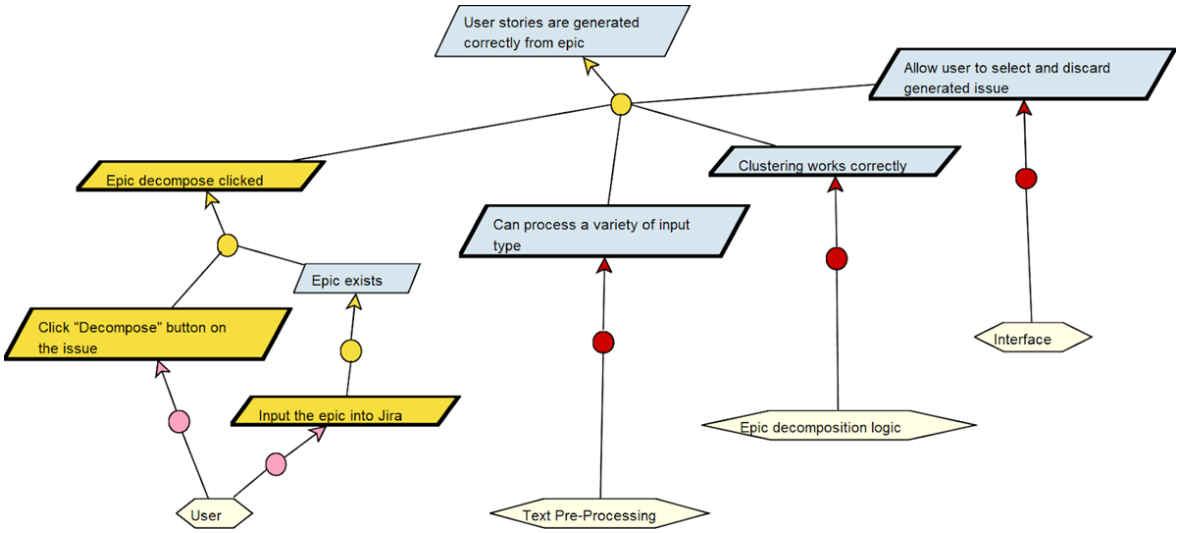


Fig. 6: Goal: "User stories are generated correctly from epic".

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 types
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

which correlated to the main FRs. Fig. 5 shows the first level of functional subgoals, also listed below:

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

Each subgoal in Fig. 5 has its own KAOS goal model to be further refined. Due to space constraint, we include one example in Fig. 6 to demonstrate the refinement from subgoal "User stories are generated correctly from epic" to the specification, as well as responsible agent for fulfillment. From these models, we concluded the functional requirements as in Table II.

C. Requirements Summary

The NFRs and FRs listed in Tables I and II are the scope of our implementation, each with their own specifications developed as well. To summarize the system's three major functional processes, we conclude that: 1) Epic decomposition consists of taking a templatized epic entered by the user and clustering requirements together into user stories based upon subject; 2) Story optimization further breaks down potentially large stories into smaller stories. Certain stories may not be subjected to change due to their intial small size; 3) Task generation is done by performing part of speech analysis on the user stories.

Furthermore, to achieve the NFRs such as integrity, for each process, the results are offered as suggestions to the user, allowing them to pick and choose which user stories and tasks to add to their Jira board. The user can further adjust settings to fine tune the granularity of generated results. i.e. number of tasks to be generated. Last but not least, epics, user stories, and tasks may be displayed in an interactive tree/cluster graph that shows the explicit and implicit relationships among them.

Overall, we consider the project's main motivation is to reduce the time needed on manual requirement engineering. While AI can play a number of different roles to achieve this goal, the team determined that the artificial intelligence component's purpose would be centered on processing text into increasingly refined portions to support an efficient RE process. Such an approach would increase the consistency of the outcome of the process, as the central process reduces the impact of human variables. At the meantime, we recognized that knowledge from an experienced project manager is still valuable, thus we make sure that the tool would not alter or discard the original input, avoiding unintended consequences.

Finally, the team combined the formatted requirement with direct feedback from our mentor and developed the user stories that we later used for demo and verification purposes, as presented in Section VI.

IV. Design

A. AI Selection Criteria

Performing an epic breakdown involves 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 the team might be able to generate a general data set of decomposed epics, we concluded that a typical epic might have too many subjects, and we would not be able to produce any accurately labeled model for training purposes.

Additionally, for the stated purpose of increasing consistencies, we made the assumption that the decomposition process would not rely upon historical project information, such as previously completed epics or sprints. As a result, unsupervised clustering techniques were chosen as it could be equally applicable to new and old projects alike.

The last criteria for the text processing approaches was that we would not attempt to generate "new" information based on given input. 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. Table III summarizes the decisions we made regarding the selection of AI techniques.

TABLE III: AI design choices

Component	Options	Choice	Reasoning
Epic Decomposition	• K-means • Mean shift • DBSCAN • OPTICS	Mean shift	All of the mentioned clustering algorithms, besides K-means, 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 subject. 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.

B. Epic Decomposition

Through manual decompositions of epics into stories it was identified that the process fundamentally involved the categorization of requirements and specifications. As such, text clustering techniques were 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 to perform comparisons.

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 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 process be applicable to a large range of epics, in terms of subject and size, dynamic clustering techniques, such as DBSCAN, OPTICS, and Mean shift, were explored as they do not require the number of clusters to be specified. DBSCAN/OPTICS are density-based 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.

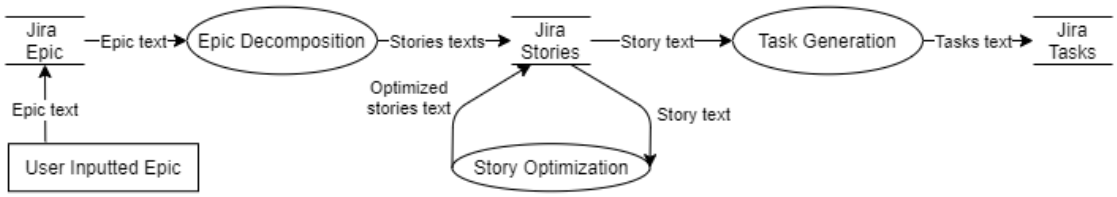


Fig. 7: Simplified Data Flow diagram outlining the flow of the decomposition processes

C. Story Optimization

Story optimizationfurther refines stories that are identified as containing more than one story or software feature. For story optimization, sentence similarity is high priority.

Google's pre-trained Word2Vec model [10] is used to extract the features out of sentences. A sentence embedding technique called SIF embeddings (Smooth Inverse Frequency) is used to compute sentence embeddings as a weighted average of word vectors. Using the features, a comparison is made by calculating the cosine similarity between sentences.

The result is the similarity coefficient between sentences and it could be used to group sentences together based on their degree of similarity. A function decision is used 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 criterion laid out in Section IV-A, simplification on the existing input of requirements can be completed by deconstructing complex sentences into simple sentences.

The first process to break down sentences was to use part 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, which words are connected to the subject can be determined. By ensuring the sentence contains one subject and verb, it is possible to create a complete simple sentence. Everytime a new subject is found, a new sentence can be created. Each simple sentence generated from the stories can then be suggested as a task.

E. Process Flow

There is an overall linear flow, as seen in Fig. 7 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.

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.

V. 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 [4], a frontend library, Atlassian User Interface (AUI) [5], 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.

Within Jira, all objects (epics, stories, issues) are rendered in the same way, each containing a shared issue panel as shown in Fig. 8. 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 panel, when opened, requests suggestions from the backend through jQuery AJAX [7] 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 handles 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 [8], a

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.

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 [6] through an open-source Python library, “atlassian-Python-api” [15], 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

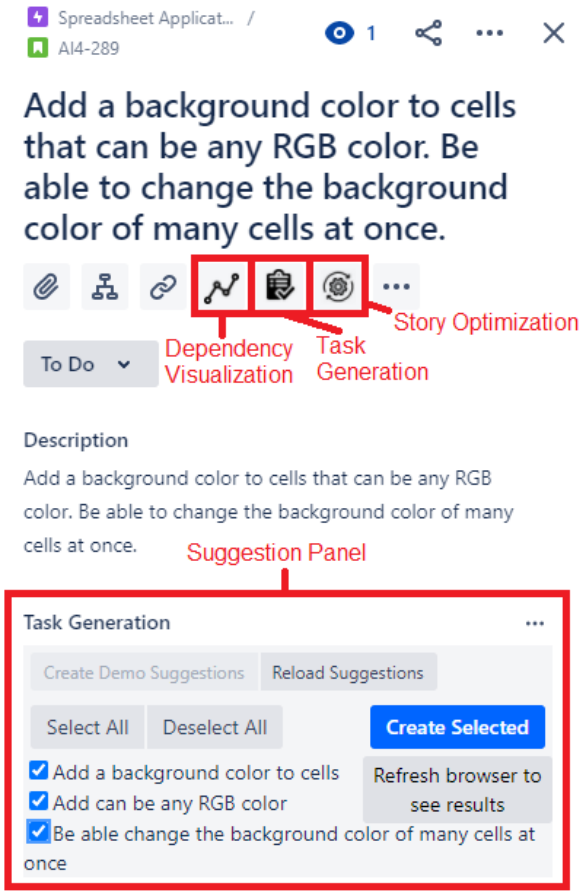


Fig. 8: Issue Panel within Jira while the Task Generation process shown after suggestions are recieved

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. 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 [20]. 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.

VI. Demonstration and Testing

A. Demonstration by Walkthrough

This section describes a walkthrough of two user stories the team developed based on our requirements, to demonstrate the primary features currently implemented in the finished prototype system.

1) Complete Decomposition Use Case: 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. 10).

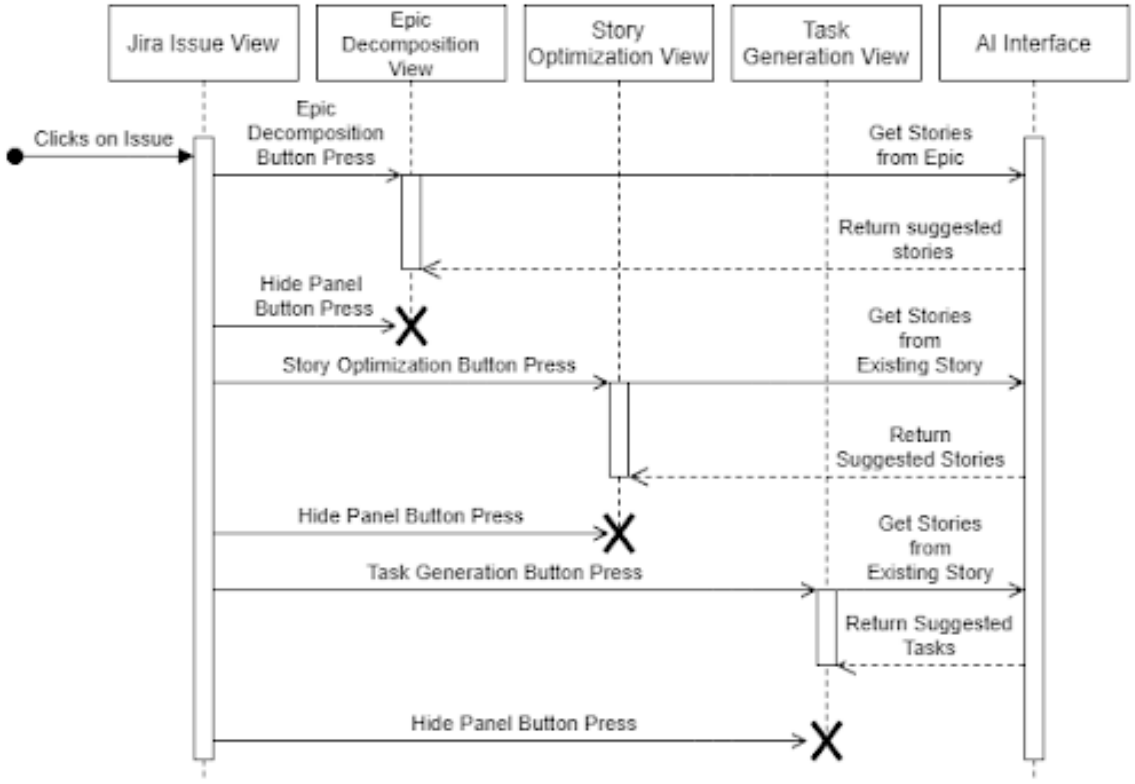


Fig. 9: Suggestion web panel sequence diagram representing the flow of messages between the frontend Jira web page and AI4Agile backend

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 [9]
Story Optimization	Word embedding	Google Word2Vec Pre-trained Model [10]
	Feature Extraction	scikit-learn: machine learning in Python [9]
	Cosine similarity	scikit-learn: machine learning in Python [9]
	Function decision	N/A
Task Generation	Sentence classification	Stanford CoreNLP [20]
	Part of speech tagging	Natural Language ToolKit [11]
UI	Tree relationship graph	Cytoscape [13]
	Clustered relationship graph	Cytoscape [13]
	Jira Cloud frontend	Atlassian Connect Express Modules [14]

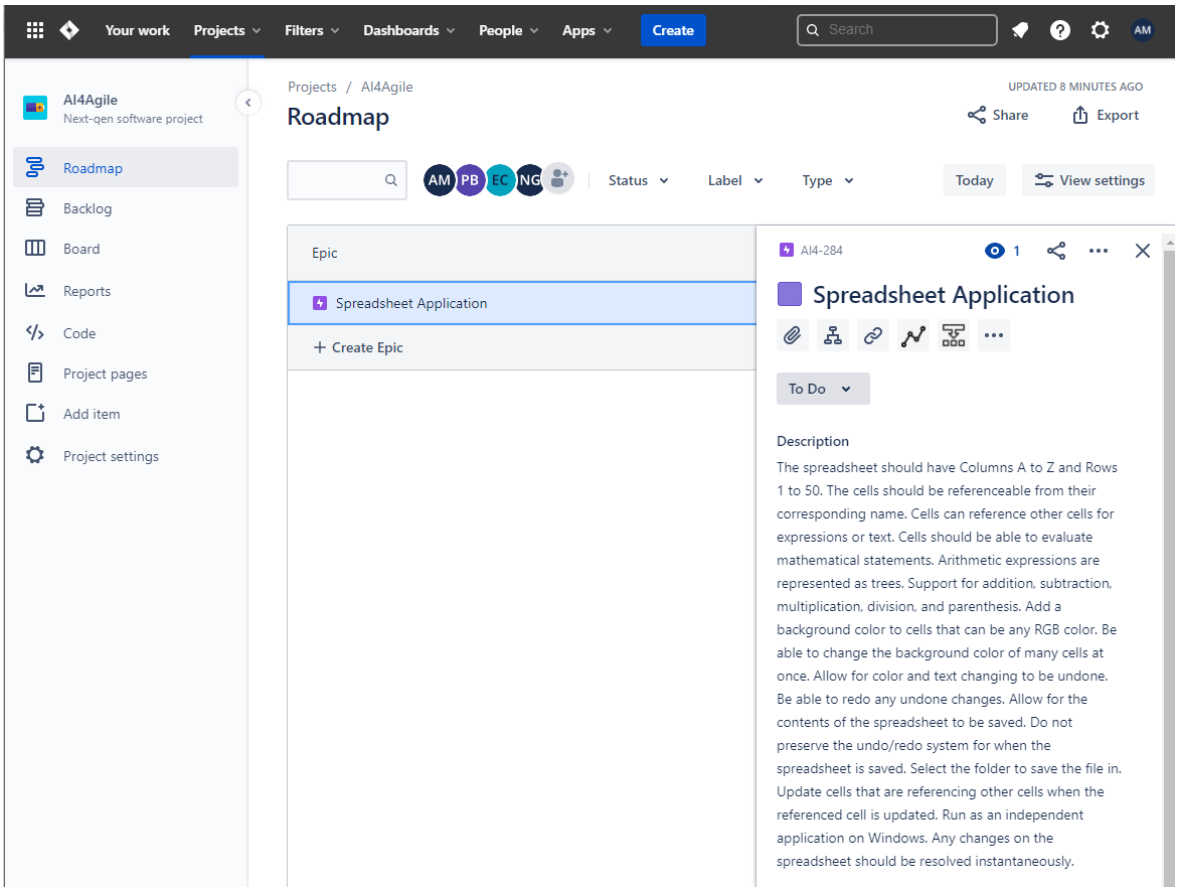


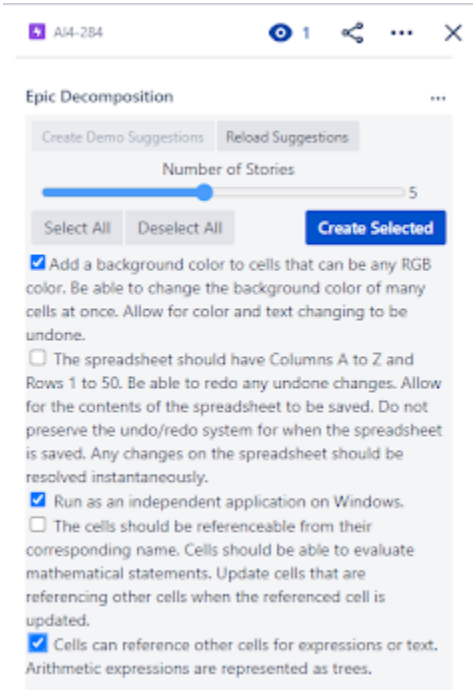
Fig. 10: Overall view of Jira Roadmap with Decompose Epic button in focus

Now, Frank looks at the suggestions the AI came up with (Fig. 11a). 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.

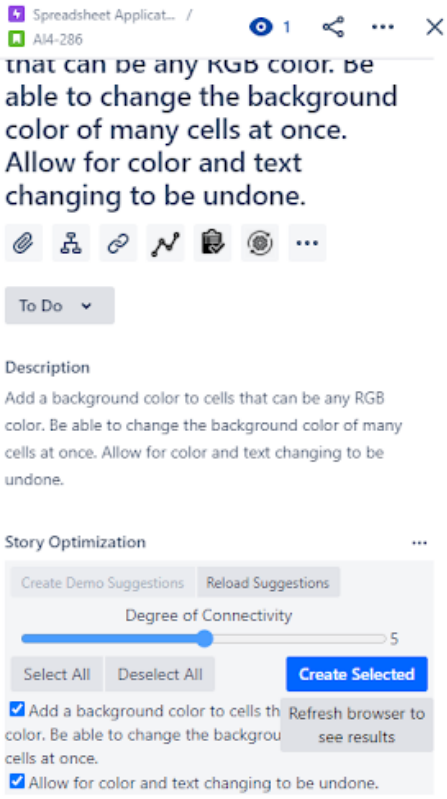
After refreshing the webpage, the newly created user stories that Frank approved are visible under the heading of the Spreadsheet Application epic.

If Frank wants to continue the process of breaking up epics, he can go into one of the user stories and click Optimize User Story (Fig. 11b). 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



(a) Generated User Story Suggestions



(b) User Story Optimization Suggestions

Fig. 11: User Story Suggestions Views

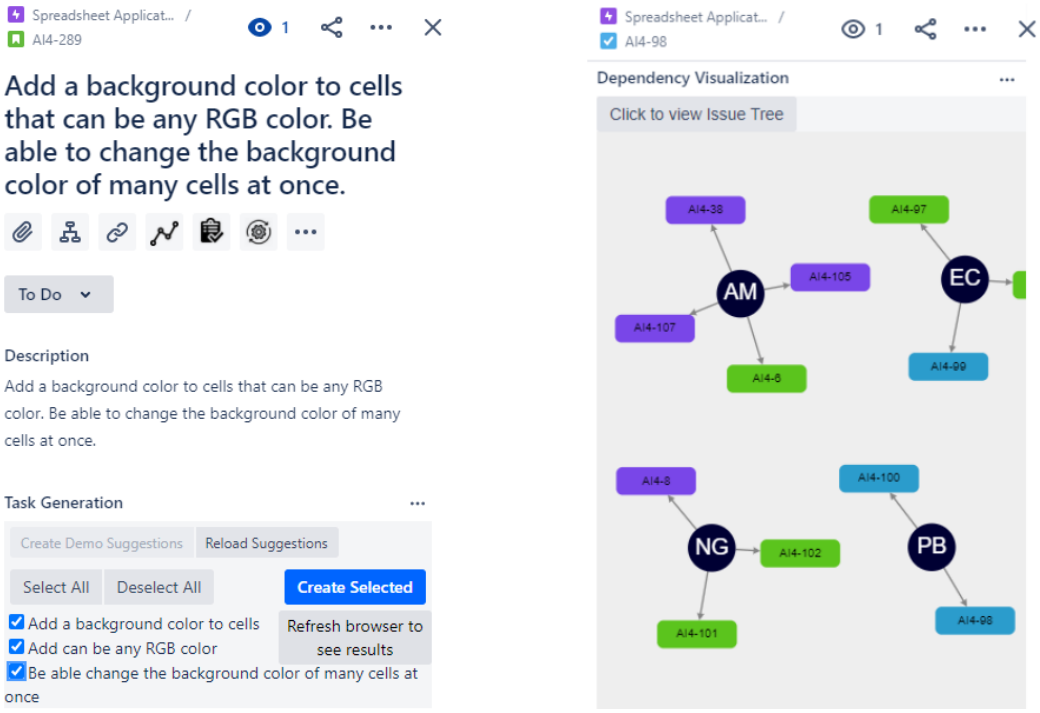
suggestions. In addition to those options, Frank can choose to adjust the connectivity slider to change how closely related items need to be in order to stay with the same story.

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. 11a).

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. 13).

2) Relationship Visualization Use Case: 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. 14). 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



(a) Task Suggestions page

(b) Developer cluster graph on a selected task

Fig. 12: Task suggestions and Cluster Graph

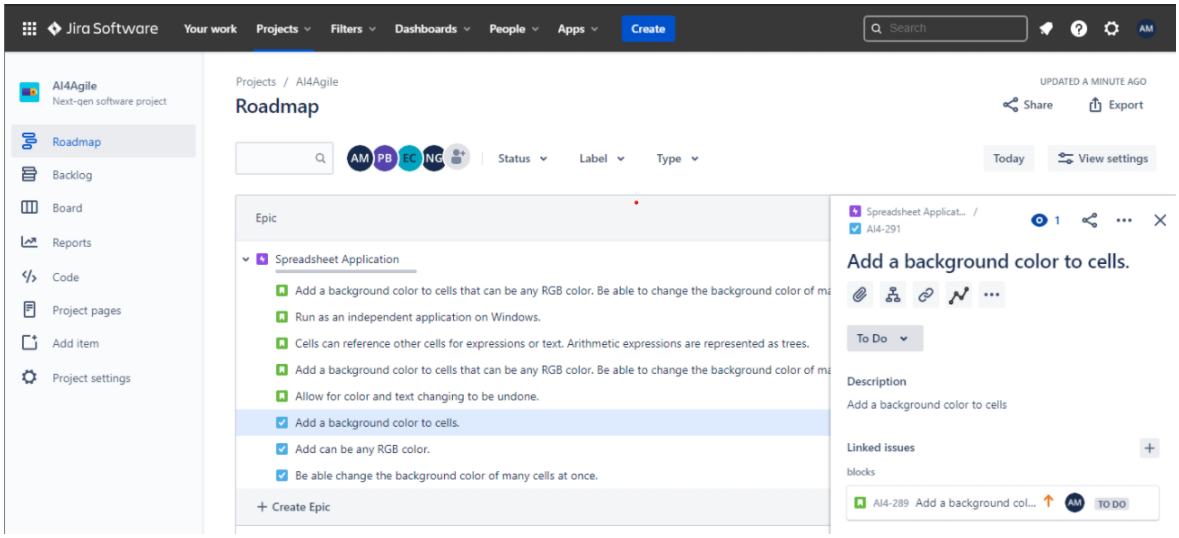


Fig. 13: View of newly created tasks for one fully decomposed user story

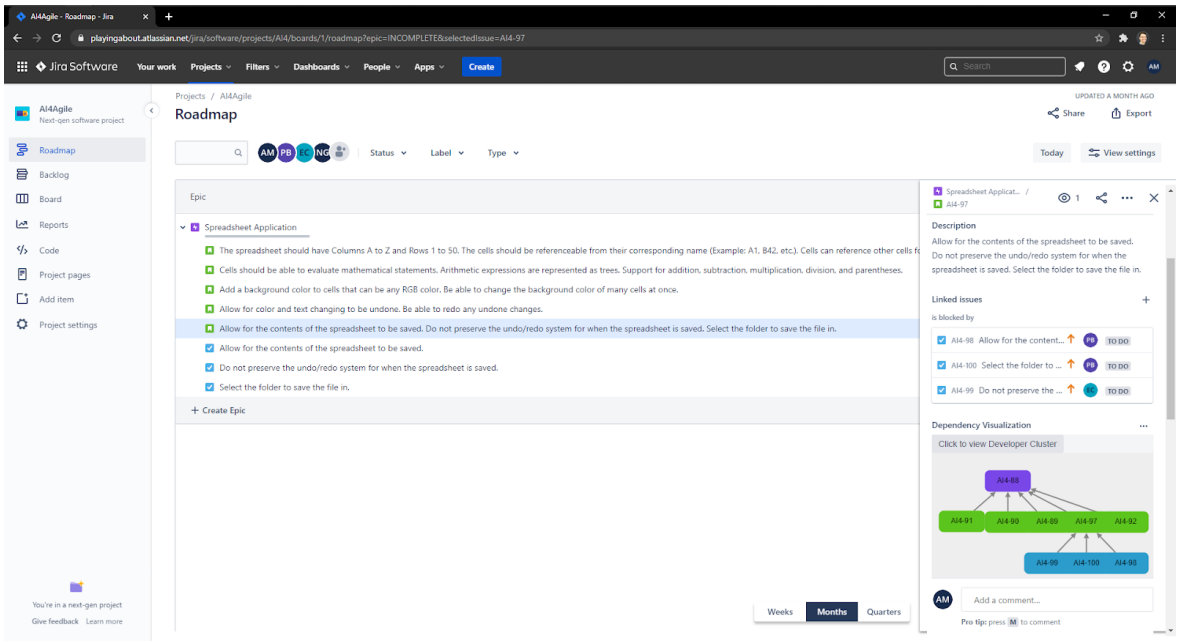


Fig. 14: A zoomed out view of navigating to the issue relationship graph

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. 14) to see what the current assignments and workload look like for her team (Fig. 12b). If she wants to go back to the previous graph view, the Click to view Issue Tree button is in the top left corner.

B. Testing

The application is primarily composed of individual web panels integrated into Jira's existing UI. As a result, each web panel was able to be created, tested, and reviewed independently 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.

The rest of functional testing was performed on component and integration levels. Component testing was separated into two categories: text processing and relationship visualization.

We based the testing of the text processing component 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 fell completely within 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 tool.

To represent the average case, semi-ideal inputs were used since natural language understanding from a vector perspective is ambiguous without context.

For integration testing, 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.

VII. Future Works and Conclusion

As part of an academic capstone project, regardless of whether the team would advance into the next stage with the SCORE competition, we plan to continue on with the project for the next few months. The team has so far defined a vision with loose deliverables in mind. For this next phase, we will deploy the project to the Atlassian Marketplace, hence also collecting feedback from potential actual customers, which will be helpful for further improvements. At the meantime, the team have noticed several possible improvements for our main features. For instance, in the epic decomposition process, we might implement dynamic clustering of stories, that automatically adjust the numbers of stories generated, instead of relying on user provided parameters. The main improvement for story optimization would be in optimizing the algorithm to be faster, as the current wait time is relatively long compared to our goal, as it caused a noticeable delay.

We are also looking into switching from the current supporting package in Java, to the Stanza Natural Language Processing (SNLP) [20] package in Python. The SNLP package has a word dependency feature that could improve the results of task generation, especially in generating complete sentences rather than fragments. The relationship visualization component also has multiple adjustments that could be made, such as filtering and ease-of-use controls for zooming and panning.

In conclusion, the team has completed the tasks within the scope as we defined based upon the AI4Agile project proposal. In this paper, we presented the team's structure and SE process, as well as all major stakeholders invovled. We then described in details the lifecycle throughout the project, covering requirements, design, implementation and final user story based testing and demonstration. The team have plans to continue the project in the near future as part of our capstone project, and will look into explore several directions where improvements on the main features could be done. The ultimate goal of the team is to deploy the project to be adopted by potential real clients.

Acknowledgment

Special thanks to Dr. Hoa Khanh Dam, Dr. Bolong Zeng, Prof. Jeremy Thompson, and Skip Baccus for their support, assistance, and time given to the team and this project.

References

[1] Ai4agile Project Proposal. https://conf.researchr.org/home/icse-2021/score-2021.
[2] Atlassain Jira Marketplace. https://marketplace.atlassian.com/addons/app/jira.
[3] Atlassain Jira Cloud. https://www.atlassian.com/software/jira/whats-new/cloud.

[4] Atlassian Design Guidelines https://atlassian.design/.
[5] Atlassian user interface https://aui.atlassian.com/.
[6] Atlassian REST API https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/
[7] JQuery AJAX https://api.jquery.com/jquery.ajax/.
[8] Flask https://flask.palletsprojects.com/en/1.1.x/.
[9] scikit-learn https://scikit-learn.org/stable/.
[10] Google word2vec https://code.google.com/archive/p/word2vec/.
[11] Natural Language Toolkit https://pythonprogramming.net/tokenizing-words-sentences-nltk-tutorial/.
[12] Stanford NLP stanza https://stanfordnlp.github.io/stanza/depparse.html
[13] Cytoscape https://cytoscape.org/
[14] Atlassian Connect Express Modules https://bitbucket.org/atlassian/atlassian-connect-express/src/master/
[15] atlassian-Python-api https://github.com/atlassian-api/atlassian-python-api
[16] Hoa Khanh Dam. Empowering software engineering with artificial intelligence. *Lecture Notes in Business Information Processing*, 367, 2018.
[17] Hoa Khanh Dam. Artificial intelligence for software engineering. In *XRDS: Crossroads*, volume 25, pages 34–37, 2019.
[18] Hoa Khanh Dam, Truyen Tran, John Grundy, Aditya Ghose, and Yasutaka Kamei. Towards effective ai-powered agile project management. In *Proceedings of the 41st International Conference on Software Engineering (ICSE 2019)*. New Ideas and Emerging Results, 2019.
[19] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20:3–50, 1993.
[20] Bidyut Das, Mukta Majumder, and Santanu Phadikar. A novel system for generating simple sentences from complex and compound sentences. *International Journal of Modern Education and Computer Science.*, 1:57–64, 2018.
[21] Respect-IT. A kaos tutorial. 2007.