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 environments in streamlining user story refinement. The application utilizes Artificial Intelligence (AI) to power 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 users to visualize the explicit and implicit relationships among the requirements. Team Katara from Washington State University, Everett USA submits this report for the ICSE SCORE 2021 competition.

The rest of the report is organized as follows. Section II outlines preliminary background information of the app, the stakeholders, and the team's development process. Section III summarizes the requirements modeling, goal-oriented acquistion and decomposition. Section IV and V outline the design and implementation of the tool respectively, while the latter expanded on detailed changes made transitioning from initial design to the final implementation. Section VI demonstrates the primary functionalities of the tool via text and video walkthroughs of use cases, as well as the work done for testing. Lastly, Section VII reflects on the outcome of the project and explores its potential future prospects.

II. Background and Process

A. Project Scope

The team chose the project "AI4Agile: Developing AI-enabled tool support for user story refinement in JIRA" for our participation in the SCORE 2021competition. The project is sponsored by Dr. Hoa Khanh Dam from University 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 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, and we name ourselves Team Katara. [1]

The team chose AI4Agile from the list since the team were most interested in working with AI. The idea of intersecting AI with Software Engineering (SE) was also intriguing, particularly 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 had no prior experience with AI and Natural Language Processing (NLP), we asked for advice from Prof. Jeremy Thompson, whose main expertise is in these fields. We also contacted the SCORE sponsor for AI4Agile Dr. Dam as soon as the team started working on the project. 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 to have Skip Baccus, a senior project manager from Microsoft, who agreed to fulfill the role as a domain expert. Throughout the project, he worked with us both as a mentor and as a stand-in for potential client. All people mentioned above rounded up the stakeholders in the project. Team member Aric Monary served as the team liaison to coordinate communications among all involved parties.

C. Team Process

We adopted an agile process for ourselves throughout the project. The members met a minimum of 2 times per week, working on both documentation and implementation tasks. The meetings led into the scrum meeting with our business mentor every Tuesday, where we debriefed the mentor on our progress on features, and solicited feedback from his perspective. In between these meetings, the team worked on their own assignments, while the faculty advisor reviewed our documentations. Figure 1 shows the major activities of the team on a regular basis. Our approach emphasizes heavily on the quick feedback turn-arounds and frequent iteration loops.

The team started by studying Dr. Dam's papers, mainly [18], and brainstorming concepts that we considered vital to the project. We then split into doing research on various key issues, such as Jira API, NLP, and more according to individual strengths and preferences. Meanwhile, the entire team participated in regular requirement gathering sessions with the mentor to refine the initial scope of the project. Throughout the process we have obtained key understanding on the agile process itself, and how the tool can best assist within the project scope. The requirements are presented in more details in Section III. As we were finalizing the necessary major features, the team simultaneously started early coding to experiment with implementation based on our earlier research.

[1] As part of this year's theme of our Senior Design Capstone projects (*Avatar: the Last Airbender*).

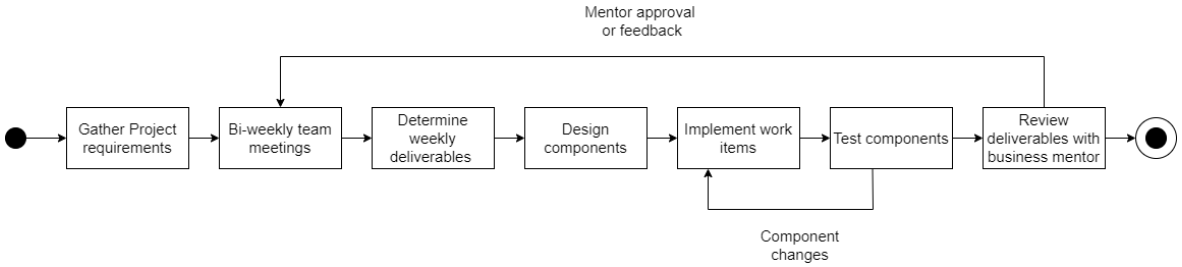


Fig. 1: Activity Diagram outlining the SE process

The development effort mainly consists of two parts: AI and UI components. The AI components were of high priority for the majority of the project. Three members each focused on one of the three primary features, 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 as 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. Fig. 2 presents the project timeline, with key work tasks and dates noted. Fig. 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 infrastructure 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.

For the purpose of our project, we consider the following key concepts. **Epic** is defined as 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 further broken down into **Tasks**, actionable items to be completed as part of a user story. Epic, User Story, and Tasks are the three levels of requirements items that our tool focuses on.

B. Requirement Modeling with KAOS

First, the team modeled the requirements using 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 guarantee thorough traceability across all requirements, which includes describing the problems, identifying top level goals to address problems and stating the constraint for solutions, etc. Moreover, using the KAOS models, the team could easily remove or add requirements as we encounter new information and changes. The KAOS tutorial handbook [21] also provides universally applicable generic patterns for reference, and we specified our initial goals by adopting these patterns, with selected models presented below.

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

Fig. 2: Project timeline

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: Team member responsibility breakdown

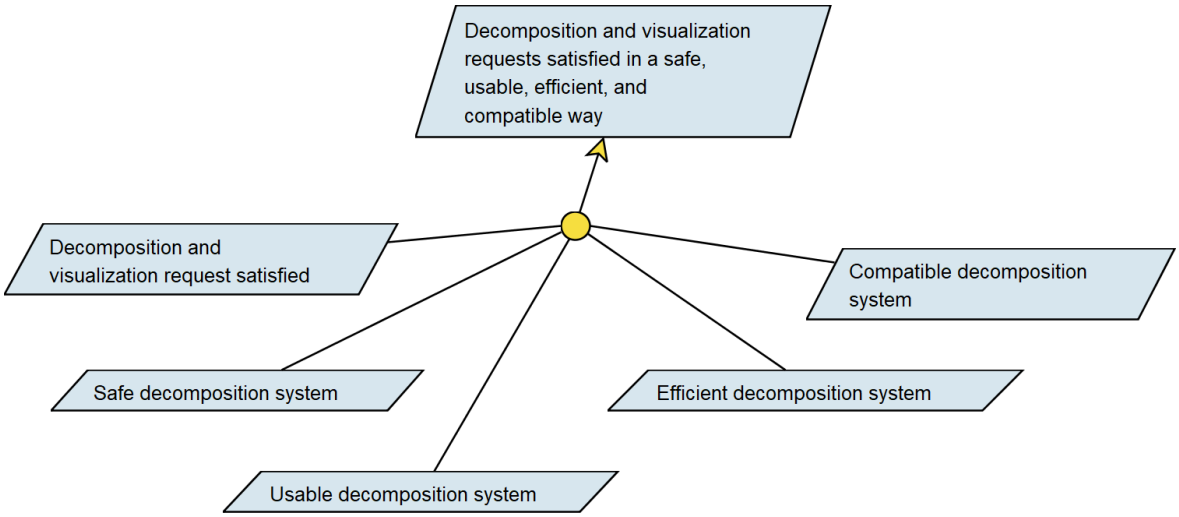


Fig. 4: Top level goal model of AI4Agile

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 cases that must be covered to fulfill our main 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

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

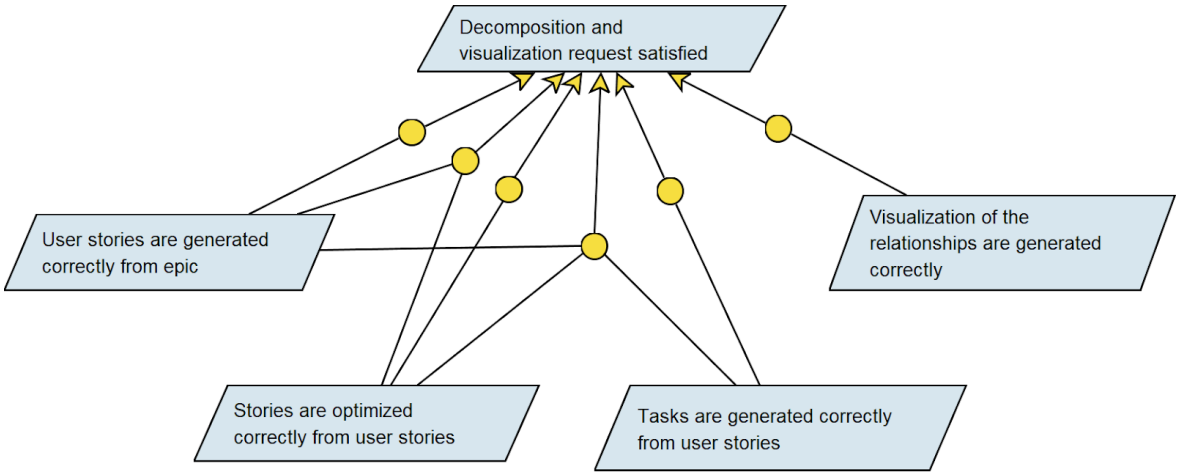


Fig. 5: Functional goal decomposition continued from Fig. 4

The goals for “Safe”, “Usable”, “Efficient”, 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, which correlated to the main FRs. Fig. 5 shows the first level of functional subgoals, 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 models that were further refined. Due to space constraint, we include one example in Fig. 6. Fig 6 shows how we refined the subgoal “User stories are generated correctly from epic” into the specification. It also identifies the agents responsible

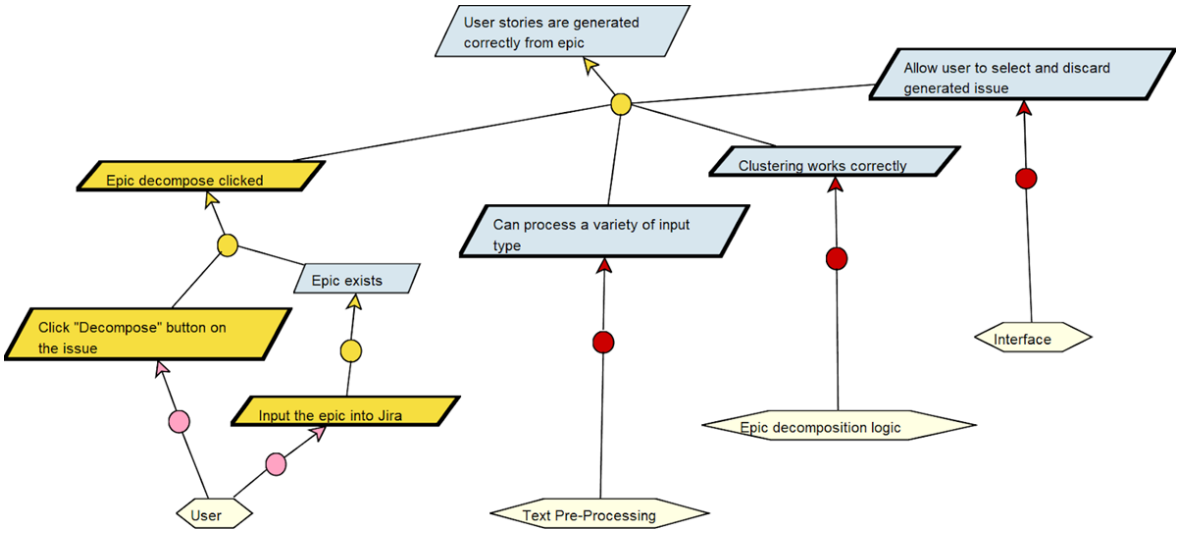


Fig. 6: Further refinement of one subgoal in Fig. 5

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

for fulfilling this subgoal. From these models, we obtained the functional requirements as shown in Table II.

C. Requirements Summary

The NFRs and FRs in Tables I and II defined the initial scope for our implementation, each with their own specifications developed. To summarize the system's three major functional processes, we concluded that: 1) Epic decomposition means that, given a semi-structured, manually entered epic, the app would refine it into user stories in the form of subject-based clusters; 2) Story optimization further breaks down large stories into smaller stories; 3) Task generation is done by performing part-of-speech analysis on the user stories.

Furthermore, to satisfy the NFRs such as integrity, for each process, the app would offer suggestion results 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, i.e. number of tasks to be generated, to fine tune the granularity of generated results. Finally, epics, user stories, and tasks can be displayed in an interactive tree/cluster graph that shows the explicit and implicit relationships among them.

The project's main motivation is to save time for manual Requirement Engineering (RE). While AI could be used in different ways to achieve this goal, our app predominantly utilized the AI component in text processing, generating increasingly refined results to support an efficient RE process. This approach could also increase the consistency of the RE outcomes, as the central process reduces the impact of human factors. In the meantime, we recognized that knowledge and experiences from a skilled project manager is still crucial, thus we make sure that the tool would not alter or discard any original input, preventing unintended consequences. Last but not least, with direct feedback from our mentor, the team formatted sample requirements we have encountered in practice, and developed the user stories used for demo and verification purposes, as presented in Section VI.

IV. Design

A. AI Selection Criteria

Performing an epic breakdown involves the task of 1) clustering related requirements/specifications, and 2) breaking complex information into smaller stories and tasks. Clustering can be performed with 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 contain too many subjects, and we would not be able to produce accurately labeled model for training purposes. In addition, for the stated purpose of increasing consistencies, we assumed 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 final consideration for choosing the text processing approach was that we would not attempt to generate "new" information based on given input. This was due to lack of specific project domain data that could be used to generate stories and tasks. Table III summarizes the rationales regarding the selection of AI techniques.

B. Epic Decomposition

Through manual decompositions of epics into stories, we discovered that the process was fundamentally about categorizing requirements and specifications. Each requirement, assumed to be in the form of a complete sentence or distinguishable section, is vectorized and normalized to perform comparisons. Then, text clustering may be performed based upon such vectors.

After a series of tests on a simpler text clustering problem, K-means, a non-deterministic centroid-based clustering technique, yielded the best results for creating consistent clusters of requirements. The downside to K-means is that it requires the number of clusters to be identified, forcing a user to provide an input or an estimation. Since we want the process to be useful for large epics with many subjects, we also investigated dynamic clustering techniques such as DBSCAN, OPTICS, and Mean

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 similarity • 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.

shift []. Such techniques 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, we chose Mean shift as the implementation included an estimation of cluster sizes, thus requiring no external input. Upon implementation, it was discovered that Mean shift could not sufficiently filter out noise and it was decided that k-means would be used since it allowed the user to select the number of stories created from an epic.

C. Story Optimization

Story optimization further refines stories that might contain more than one story or software feature. Our tool uses sentence similarity to power the optimization process. We used Google's pre-trained Word2Vec model [10] to extract features out of sentences. We applied a sentence embedding technique called SIF embeddings (Smooth Inverse Frequency) to compute the sentence embeddings as a weighted average of word vectors. Using the features, we can deicde how similar two sentences are by calculating the cosine similarity. The result is the similarity coefficient between sentences and it is then used as a basis to group sentences together. A function decision is used to correctly group sentences from the similarity coefficient using a specified threshold level.

D. Task Generation

Task generation is the process of breaking down a requirement into its simplest form. Based on the criterion laid out in Section IV-A, simplifying existing input of requirements can be completed by deconstructing complex sentences into simple sentences. The first step to break down sentences was to use part-of-speech tagging to remove unnecessary stop words from the sentence so that

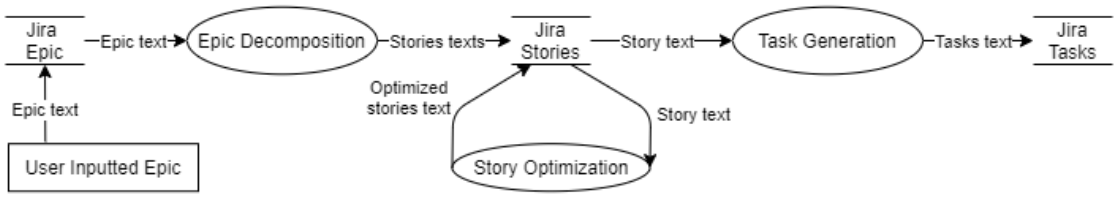


Fig. 7: Simplified Data Flow diagram outlining the flow of the decomposition processes

each simple sentence contains only one subject and predicate. By ensuring so, it is possible to create a complete simple sentence. Every time a new subject is found, a new sentence is created. Each simple sentence generated from the stories is then suggested as a task.

E. Process Flow

Fig. 7 models the overall flow of AI4Agile's main processing. A relationship graph may be generated at any point and is independent of the processes in Fig. 7. The starting point in the flow may also vary between uses. For example, a user may have manually entered all their overarching user stories for their epic, skipping the Epic Decomposition process entirely, and go directly to perform either story optimization or task generation.

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 relevant issues. This graph provides a way to view those relationships in one place at a glance, for ease of understanding. Having such a view can assist users in agile processes such as grouping tasks for sprints, or assigning tasks to developers. The usefulness comes by showing which tasks depend on other issues and need to be finished in series, and which tasks have already been assigned to certain developers, or which developers already have a large task load and shouldn't be the first pick for additional assignments.

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.

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	For the scope of the initial release, it was determined that a database would only add an unnecessary level of complexity. For later features that may include the use of historical project data, this will be reassessed.
• 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 was for new users to be able to experiment with the plugin without fear of unintended consequences such as data losses. 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.

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. The team coded our own UI components adhering 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 added buttons. When a process button or visualization button is pressed, a new panel appears.

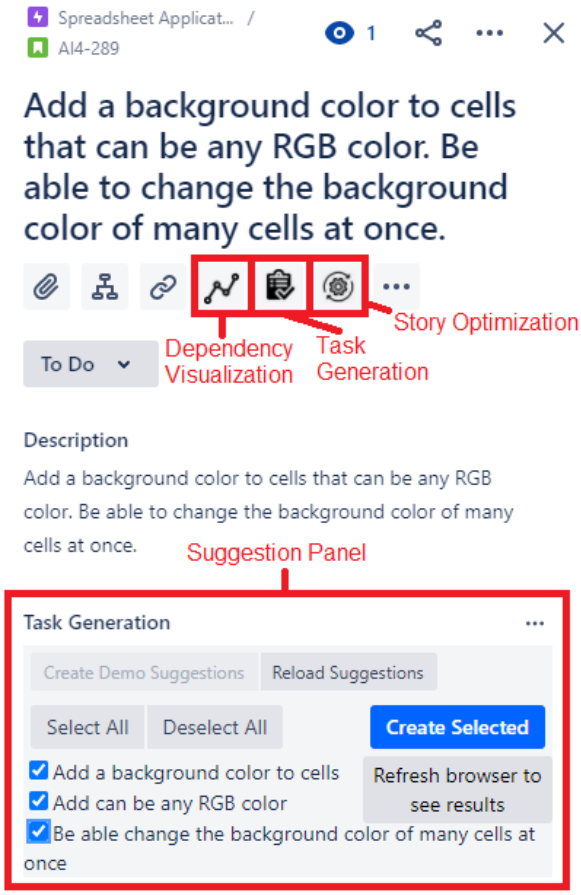


Fig. 8: Issue Panel within Jira with Task Generation suggestions

Each of the processes uses 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 handled by the backend.

B. Backend

The backend served two primary purposes: 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) in the same manner, each having an identical set of shared fields with the addition of customized fields depending on the type of issue.

All three processes were implemented using available Python libraries, with the primary interface also written in Python. The interface uses Flask [8], a Python web framework, to receive messages sent by AJAX in the frontend. The AI interface component uses atlassian-Python-api [15], which simplifies Atlassian Rest API [6] calls, to gather necessary information from the Jira issues. 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 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 noises such that the clusters were neither completely disjoint nor 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 two and ten stories using a newly added slider in the UI for the Epic Decomposition process.

2) Story Optimization: For Story Optimization, in the beginning, 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 sentence embeddings as a weighted average of word vectors. However, these techniques were extremely computationally intensive. The story optimization also required every sentence to be compared against one another, which led to a long waiting time. To overcome the obstacle, the implementation was changed to use TF-IDF from the scikit-learn python library. The technique was also able to extract features from sentences with a faster process time.

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

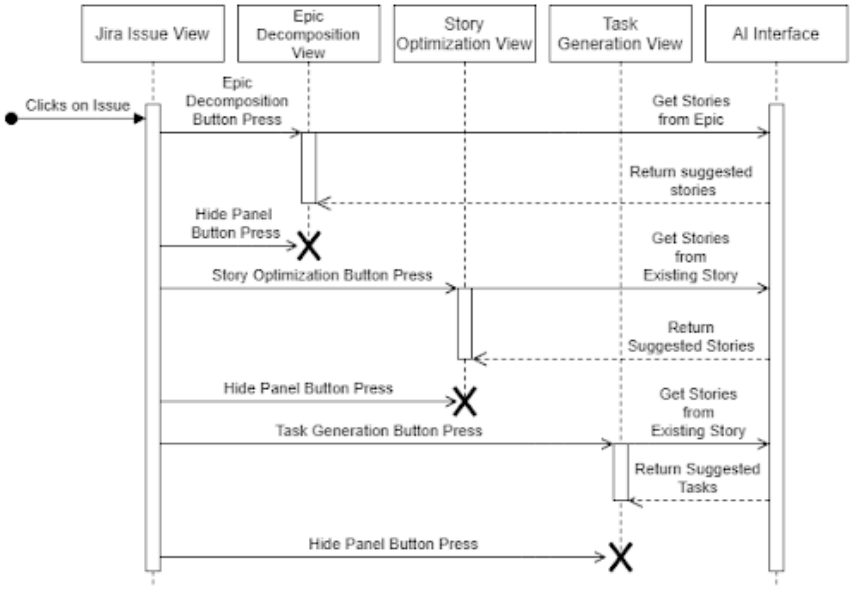


Fig. 9: Suggestion web panel sequence diagram of messages between the frontend Jira web page and AI4Agile backend

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.

D. Hosting

The AI4Agile Jira Connect App front-end and back-end is hosted as three Azure Web Apps. The front-end is a single Node.js Web App and the back-end is two Python Web Apps. As previously discussed, the front-end uses AJAX and the back-end uses Flask Web Framework for communication. Jira communicates with the front-end Azure Web App to get assets that are to be

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	scikit-learn: machine learning in Python [9]
	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]

rendered within Jira. The back-end is separated into two sets of features: dependency visualization and text processing. The dependency visualization back-end accesses the data from Jira, which is used to construct graphs. The text processing back-end contains all three of the primary processes (epic decomposition, story optimization, and task generation). The Jira web page does not directly communicate with back-end. Rather, the Jira web page communicates with the front-end and then the front-end communicates with the back-end.

VI. Demonstration and Testing

A. Demonstration by User Story 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).

Now, Frank looks at the suggestions the AI came up with (Fig. 11a). If he decides he does not want to use any of these suggestions, he can click the righthand side ellipsis dropdown to hide the Epic Decomposition panel. 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

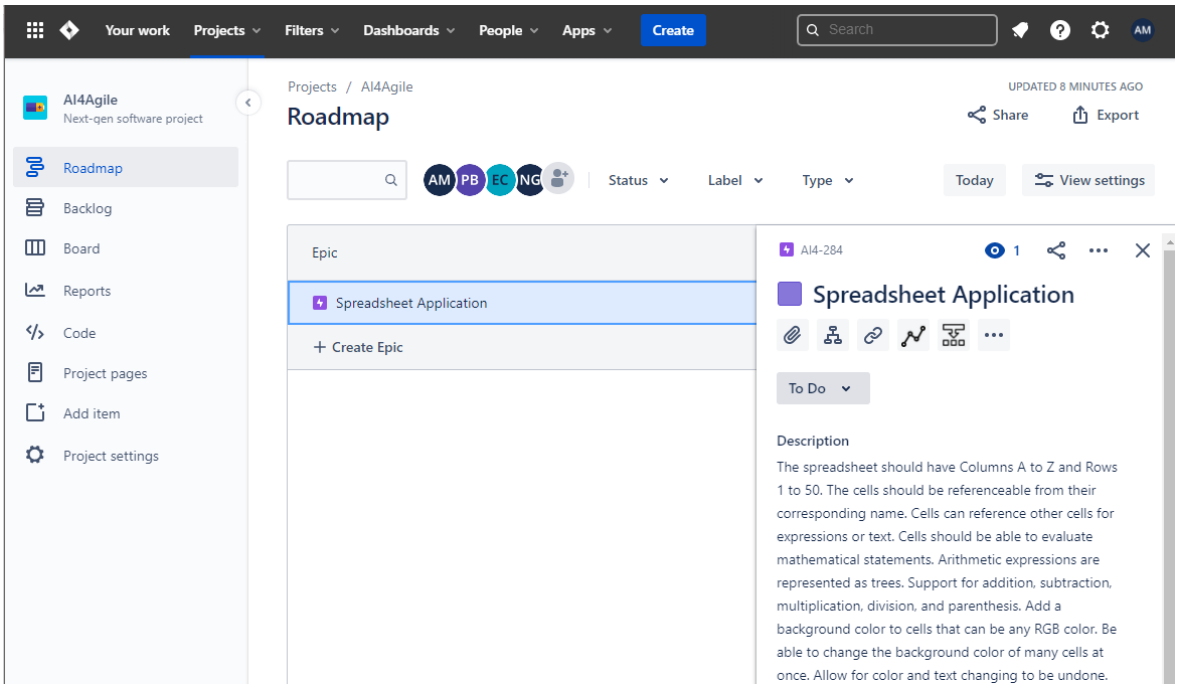


Fig. 10: Overall view of Jira Roadmap with Decompose Epic button in focus

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

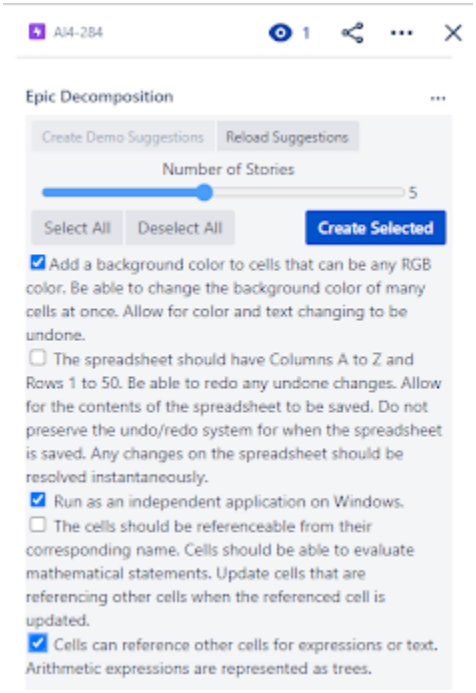
Now there are five user stories since the fourth Story was optimized into two separate stories. To continue the decomposition process, Frank opens a story and clicks Generate Tasks (Fig. 12a).

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.

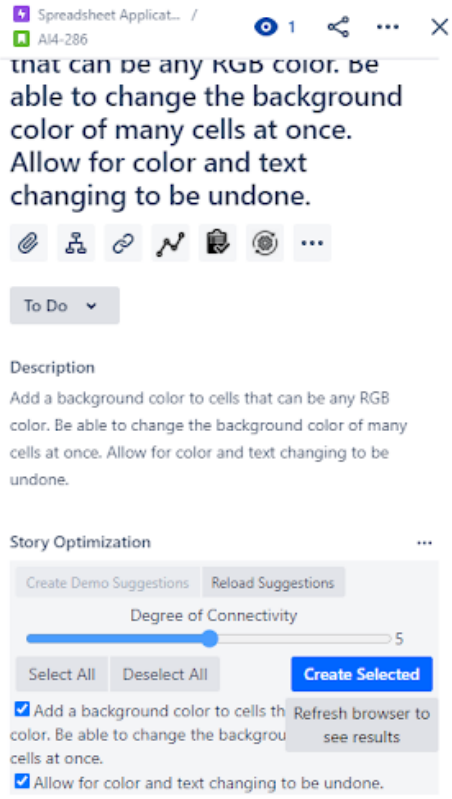
The tasks have now been created and linked to their parent story to indicate a blocking relationship. All Frank had to do was make some decisions and maybe edits, and now he's got one story fully decomposed (Fig. 13).

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 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). To see the previous graph type, she can Click to view Issue Tree button is in the top left.



(a) Generated User Story Suggestions



(b) User Story Optimization Suggestions

Fig. 11: User Story Suggestions Views

B. Demonstration by Video and Source Code

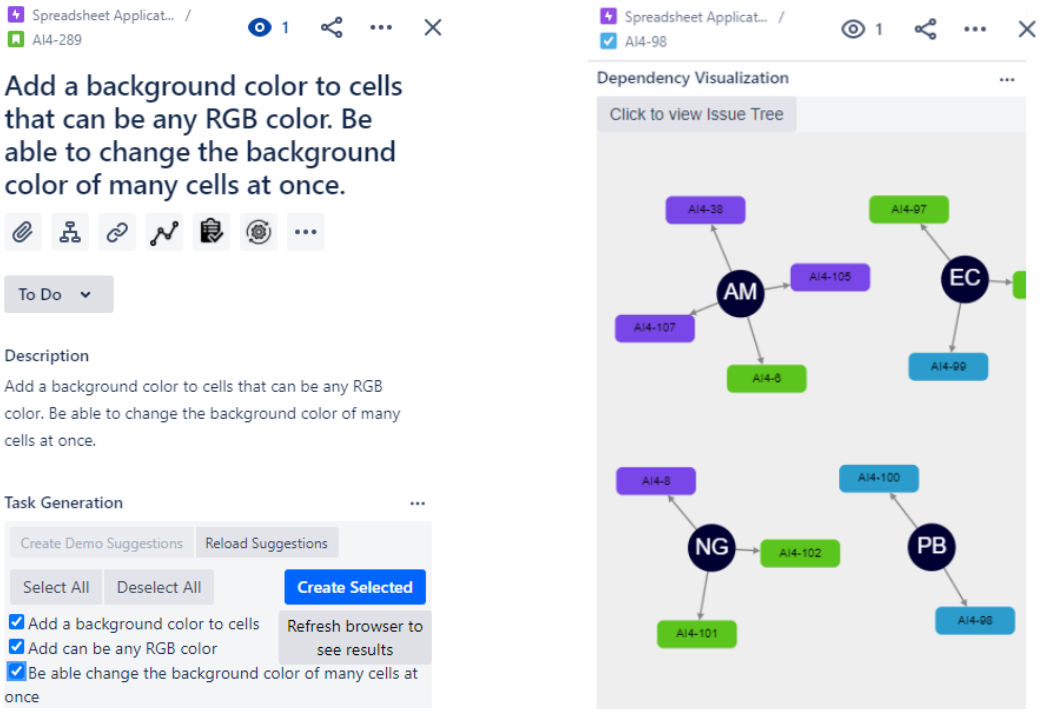
A video demonstration of the functioning AI4Agile application can be found here: http://www.youtube.com. Additionally, the source code and documentation for this project has been made available at this address: https://github.com/AricMonary/AI4Agile/tree/submission.

C. 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



(a) Task Suggestions page

(b) Developer cluster graph on a selected task

Fig. 12: Task suggestions and Cluster Graph

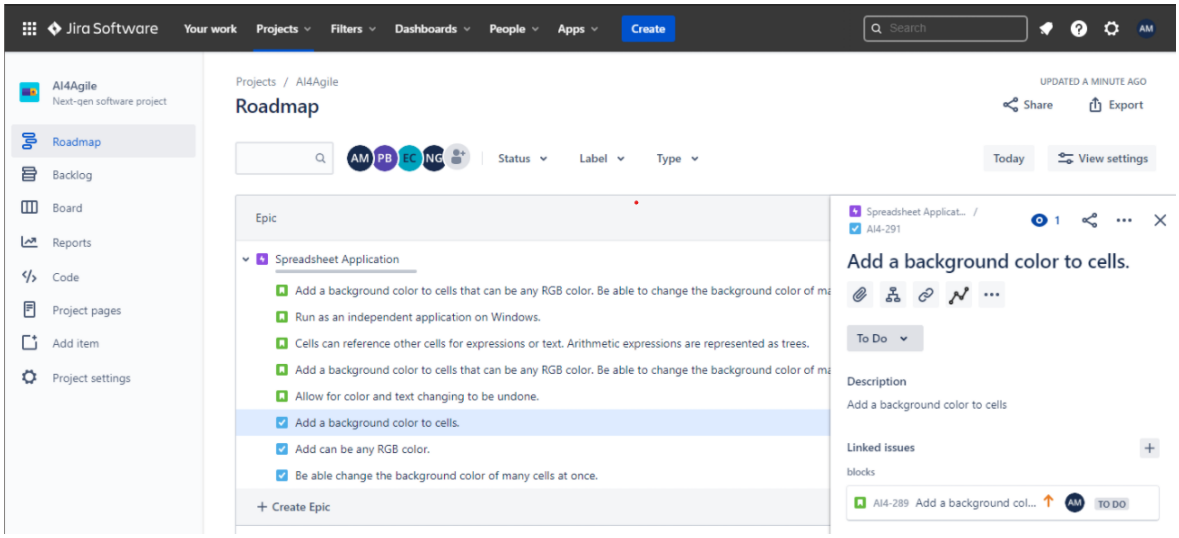


Fig. 13: View of newly created tasks for one fully decomposed user story

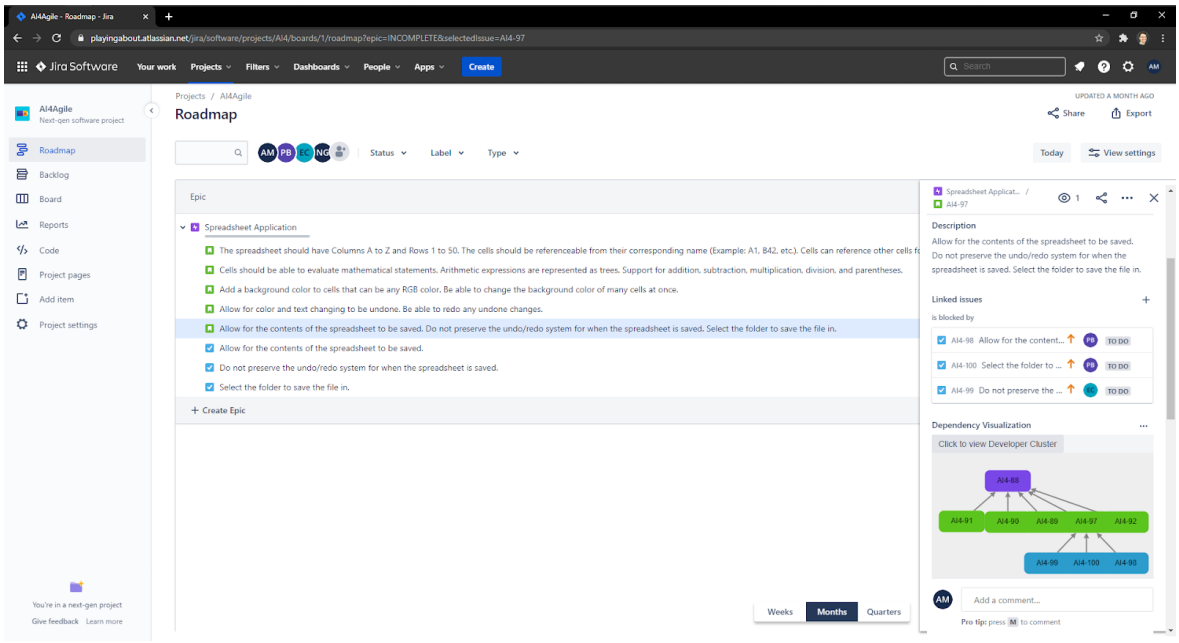


Fig. 14: A zoomed out view of navigating to the issue relationship graph

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 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. In 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 involved. 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 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.