AI4Agile Solution Approach

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

Date Submitted: 11/20/20

1. Product Solution

As an overview to the AI4Agile plugin, it is be broken down into five blocks:

1. Interface - The front end of both the Jira platform and our plugin, which is integrated within the platform.
2. Text Preprocessing - The first step to each of the AI processes to prepare the text data, either stemming from user input or generation from the AI processes.
3. Epic Decomposition - A clustering process that groups together requirements and specifications, which are in the form of individual sentences, into blocks based on related wording.
4. Story Optimization - A process of breaking down user stories by grouping sentences that has high semantic similarity.
5. Task Generation - Process of generating tasks, which are deliverables, based on simple sentence generation.

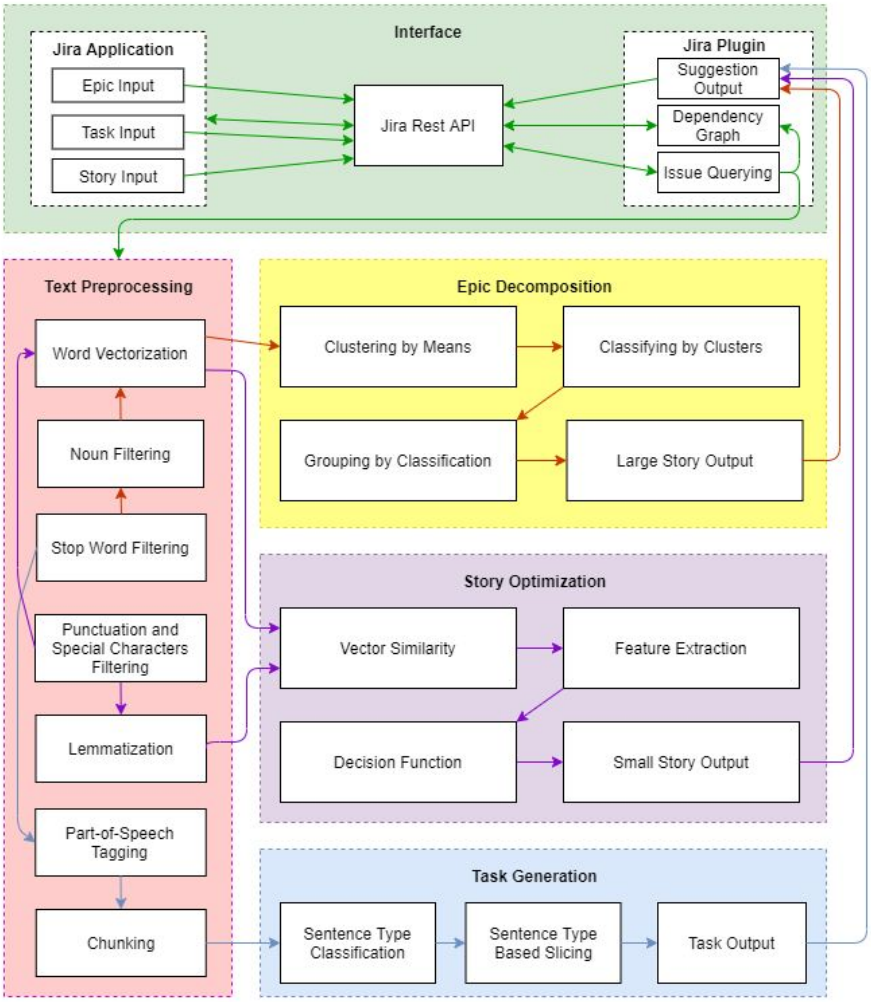

Figure 1.1: Block diagram, wherein arrow colors correspond to the component they belong to. Inputs to the text preprocessing portion are sent to their respective starting point subcomponent depending on where the original request originated from, i.e. which process is being called.

The processes of decomposing, optimizing, and generation could be executed out of order, independently, or sequentially. A typical user starts by inputting an epic, decomposing it, optimizing all of the generated stories, and finally generating tasks from the now optimized stories. A user may also choose to enter new stories or change existing stories between each process. As an example of independent execution, a user may only choose to use task generation from stories that have been manually entered.

Figure 1.2 shows how a process, such as story optimization, can be iterative while all other processes occur sequentially or independently. It also shows how additional stories can be manually inputted within the processes (note the User Inputted Stories box feeding into the center of the diagram).

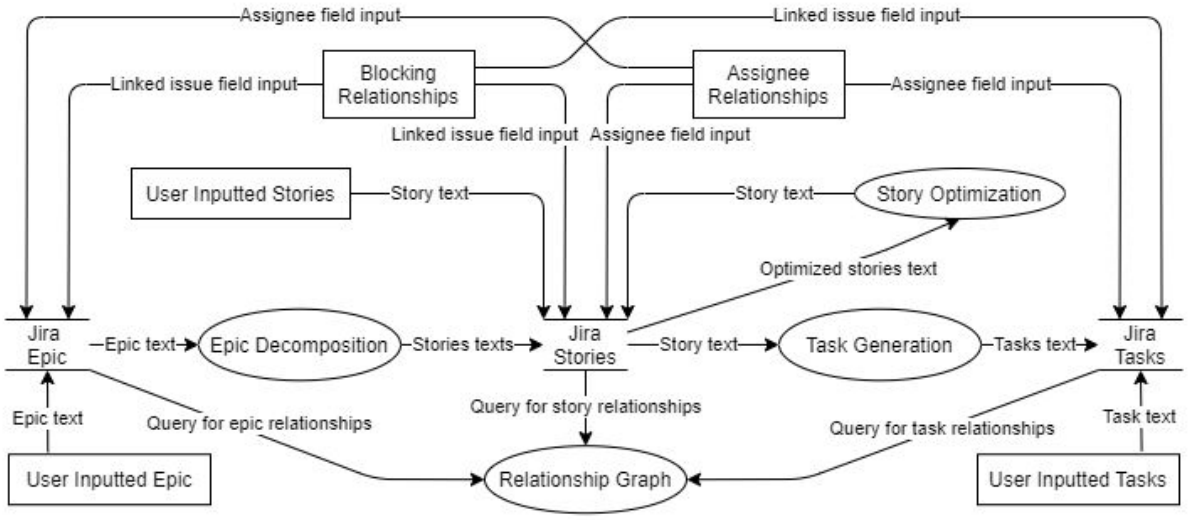

Figure 1.2: Data flow diagram. Users can input epics, stories, and tasks, and for the purposes of this application, they may use any of the processes (shown in ovals) as starting points for interaction.

1.1 UI Concept

The UI component of the plugin is responsible for facilitating interactions between the Jira application and the major feature processes: epic decomposition, story optimization, and task generation. Interactions are initiated from a sidebar pane in Jira that comes up when an issue is selected on the Backlog or Issues pages for basic Jira Scrum-template projects. There, buttons specific to the plugin are available, depending on what type of issue is selected (Epic, Story, or Task). UI activities include querying the Jira database for the information needed for the graph or AI component inputs, sending that to said components and signalling them to start, and displaying their results in the form of suggestions to the user. Each of the suggestions can then be edited, accepted, or ignored by the user.

To demonstrate the UI concept, a storyboard was created as a prototype of the UI's look and feel as it operates within the pre-existing Jira platform. This document goes through two user stories for different user classes, showing how they might interact with the application.

Link to Storyboard

1.1.1 Relationship Graph Algorithm

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. Below is given a preliminary representation of the algorithm to generate such a graph.

```
When issue pane is opened and Relationship Graph field is present:
query Jira for dependencies involving selected issue and assignees on any of those issues; store in issueSet
foreach issue in issueSet:
        create node
        add issue link to node as href
        set node color based on issue type

        if issue is involved in dependency
                if other issue node has been created
                        add edge from blocking node to blocked
        if issue has assignee
                add assignee representation to node
display html component
```

1.2 AI Components

Each of the three main processes of the AI4Agile app—epic decomposition, story optimization, and task generation—use independent algorithms, as their outputs and inputs differ. The implementations of the algorithms used for these processes are found in individual documents, which further elaborate on the flow of data, the implementation plan details, and the validation steps to be carried out.

1.2.1 Epic Decomposition Algorithm

The algorithm for the epic decomposition process takes an epic entered in Jira which can be in the form of paragraphs or ungrouped sentences. The formatting is parsed out, and blocks of sentences that are grouped together by similarity form the outputs. The algorithm utilizes the clustering models and text preprocessing functions of the scikit-learn python library.

Link to Epic Decomposition Algorithm Details

1.2.2 Story Optimization Algorithm

The algorithm for the story optimization process takes a story from Jira, which can be derived from an epic decomposition output or manually entered by a user. The results are either optimized blocks of sentences (stories) that are grouped together by semantic similarity, or in the case of already optimized story input, the same story. The algorithm utilizes a pre-trained word2vec model, along with SIF and cosine similarity functions from the scikit-learn python library.

Link to Story Optimization Algorithm Details

1.2.3 Task Generation Algorithm

The task generation algorithm determines how the optimized user story is broken down into individual tasks. Ideally, a task is a deliverable which one developer should be able to complete in no longer than 8 hours. The input is a user story, and the output is multiple tasks, or possibly one single task. The main python libraries being used are NLTK and StanfordNLP.

Link to Task Generation Algorithm Details

2. Design Choices

Detailed in the design choice section are the decision processes for each of the application's major components. This includes mainly the use of concept-scoring matrices and prototyping to compare potential solutions for the various subcomponent implementation pieces.

2.1 UI/UX Design Choices

For the user interface and experience, a combination of prototyping and stakeholder meetings went into the decisions displayed in table 2.1. 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.

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 current release, it was determined that a database would only add an unnecessary level of complexity. For later features that 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 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.

Table 2.1: UI and UX design choices for the AI4Agile plugin.

2.2 AI Design Choices

In order to narrow the number of concepts quickly, a concept-scoring matrix was used for the epic decomposition and story optimization components. For the specific approach targeted by task generation, only one method was determined to be at all feasible for generating tasks, removing the need for evaluation of choices. In the concept-scoring matrix, a weighted sum of ratings to determine a concept's ranking. The first concept in the table serves as the overall reference concept. From the result of the matrix, the concept with the highest score is chosen for the final concept.

Tables 2.2 and 2.3 show the five different options that were considered for the epic decomposition and story optimization components. For each component, there are four main criteria: Speed, Accuracy, Inclusion, and Ease of Use, and two categories to measure these criteria, which are weight and value. The higher the weight is, the more important that criterion is, e.g. in table 2.3, Accuracy has a weight of 40%, which is the highest out of the criteria. The higher the value, the better that technique is in that area. Lastly, these weights and values are multiplied for each criterion and the technique with the highest weighted total is chosen. As such, for epic decomposition, Mean-Shift was chosen. For story optimization, SIF, Word2Vec, and Cosine Similarity were all chosen.

Epic Decomposition Choices		K-Means		Mean-shift		DBSCAN		OPTICS	
Selection Criteria	% Weight	Value	Weighted Value	Value	Weighted Value	Value	Weighted Value	Value	Weighted Value
	(W)	(V)	(W x V)	(V)	(W x V)	(V)	(W x V)	(V)	(W x V)
Speed	20%	3	0.60	5	1.00	3	0.60	4	0.80
Accuracy	40%	4	1.60	4	1.60	4	1.60	4	1.60
Inclusion	30%	2	0.60	4	1.20	4	1.20	1	0.30
Ease of Use	10%	1	0.10	3	0.30	2	0.20	4	0.40
Weighted Value Totals:	100%		2.90		4.10		3.60		3.10
	Total Score	2.90		4.10		3.60		3.10	
	Rank	4		1		2		3	
	Chosen	No		Yes		No		No	

Table 2.2: Concept-scoring matrix for the epic decomposition algorithm.

Story Optimization Choices		TF-IDF		SIF		Word2vec		Cosine Similarity		Word Mover's Distance	
Selection Criteria	% Weight	Value	Weighted Value	Value	Weighted Value	Value	Weighted Value	Value	Weighted Value	Value	Weighted Value
	(W)	(V)	(W x V)	(V)	(W x V)	(V)	(W x V)	(V)	(W x V)	(V)	(W x V)
Speed	20%	5	1.00	3	0.60	3	0.60	5	1.00	2	0.40
Accuracy	40%	1	0.40	5	2.00	5	2.00	4	1.60	5	2.00
Inclusion	30%	3	0.90	5	1.50	5	1.50	5	1.50	3	0.90
Ease of Use	10%	5	0.50	5	0.50	5	0.50	5	0.50	3	0.30
Weighted Value Totals:	100%		2.80		4.60		4.60		4.60		3.60
Total Score		2.80		4.60		4.60		4.60		3.60	
Rank		3		1		1		1		2	
Chosen		No		Yes		Yes		Yes		No	

Table 2.3: Concept-scoring matrix for the story optimization algorithm.

Tabel 2.4 shows the three AI components of the project and expands upon the reasoning behind the decisions of the above scoring matrices in tables 2.2 and 2.3.

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

Table 2.4: AI design choices for the AI4Agile plugin.

3. Implementation Framework

With the specific techniques weighed and decided, the implementation framework can be set out and further detailed. Table 3.1 describes an overview of the components and subcomponents, along with the resources used to implement these pieces. Most of these subcomponent techniques are able to make use of pre-existing libraries or tools, as detailed in the Source column. The relationships between these components are centralized in the UI: epic decomposition, story optimization, and task generation are all independent pieces that get fit into the UI in order to interact with Jira and thus the user. As figure 1.2 shows, despite the independence of the back-end components themselves, it is possible with this implementation for the user to go through the app's system in a variety of flows. Regardless, this doesn't necessitate communication between the back-end components, as it all goes through the UI.

Component	Subcomponent	Source
Epic Decomposition	K-means clustering	scikit-learn: machine learning in Python [1]
Story Optimization	Word embedding	Google Word2Vec Pre-trained Model
	Feature extraction	scikit-learn: machine learning in Python [1]
	Cosine similarity	scikit-learn: machine learning in Python [1]
	Function decision	None
Task Generation	Sentence classification	Stanford CoreNLP [2]
	Part of speech tagging	Natural Language ToolKit [3]
UI	Tree relationship graph	Cytoscape [4]
	Clustered relationship graph	Cytoscape [4]
	Jira general front-end	Atlassian Connect Express Modules [5]

Table 3.1: Components and their subcomponents, along with the sources used for implementation of those subcomponents.

4. Preliminary Software Testing Plan

Testing of AI4Agile is to be completed in two stages: per unit and by integration. Due to each AI feature having many different inputs, testing for those is based on the main inputs expected from the user. For the Main GUI, the focus of testing is to see if the added features have the same feel and flow as other Jira features, and that they are intuitive to the user. Using mocked data for the suggestions generated for each AI component allows for independent testing of the UI features without having fully functional or fully integrated AI components. For the Graph, various sizes of issue sets are used to exercise the scaling and zoom features. Further testing of the graph covers the different options a user has of relationship types they want to display. The final piece for graph testing is done in order to ensure that the querying of the Jira database is done properly so that what is displayed matches expectations.

Feature		**Testing Plan**	**Target Date**
Epic Decomposition		Inputs: ● Ideal epic ● Disjoint epic (many unrelated requirements) ● Semi-ideal epic	11/27/20
Story Optimization		Inputs: ● Already optimized story ● Minimally related story (story optimization as an entry point) ● Ideal unoptimized story ● Semi-ideal unoptimized story	11/27/20
Task Generation		Inputs: ● Unoptimized story (task generation as an entry point) ● Optimized story	11/27/20
UI	Main GUI	User Trials with mocked processes for intuitiveness of button locations and ease of use testing	11/24/20
	Suggestions	Using mocked data for Epic, Story, and Task types to manually test results from the Create Selected button after editing, selecting, and deselecting various suggestions.	11/17/20
	Graph	Inputs: ● Large and small issue sets for testing zoom and scaling ● Assignee-only/dependency-only/none/both relationships turned on ● Ensuring proper querying per issue chosen	11/27/20

Table 4.1: Unit testing plan per component

For each AI component, once they are integrated they can be used disjointly by the user, therefore the testing is based on different choices the user can make, as well as different types of starting inputs. For the UI/UX, testing is focused on ensuring that each component continues to function as expected when integrated with the interface.

Integration	Testing Plan	Target Date
Epic Decomposition	Flows: ● Epic Decomposition directly to Story Optimization ● Epic Decomposition directly to Task Generation	12/03/20
Story Optimization	Flow: ● Directly to Story Optimization ● Story Optimization directly to Task Generation	12/03/20
Task Generation	● Directly to Task Generation ● Epic to Task Generation ● Optimized story to Task Generation	12/03/20
UI/UX	● Integration testing per sub-component plugged in ● User observation	12/01/20

Table 4.2: Integration testing plan

5. Project Status

Generally, we have not been able to follow our original project timeline as prepared at the beginning of the project due to further refinement of scope and limitations. The project timeline has been revised to include specific deliverables.

Link to Updated Project Timeline

Progress on the project can be broken into two pieces: UI and AI Components. The UI was further broken down into three parts—the main UI, suggestions system, and dependency visualization (graph). The AI still maintained its parallel development as the processes are not connected beyond their outputs feeding in as inputs for other processes, directly through a lack of user change or indirectly through additional edits to the output stories.

The progress of development for the AI systems was slowed, especially in the case of task generation as little research has gone into the topic of breaking stories down by its sentence structure. Development of the epic decomposition and story optimization algorithms were able to be completed within the specified times however the tweaking of these algorithms to provide meaningful clustering is still continuing past the original timeline.

6. References to Utilized Resources

1. scikit-learn: machine learning in Python
2. Stanford CoreNLP
3. Natural Language ToolKit
4. Cytoscape
5. Atlassian Connect Express Modules