Rule ID	Scrum Rule Definition as per Scrum Guide, Scrum Alliance, Scrum.org	Scrum Rule Interpretation	Theoretical Calculation of Rules	Data Fields Used	Derived Variables	Scrum Phase Affected	Scrum Event/Artifact Affected	Scrum Role Affected	Verifiable	Not Verifiable	Pseudocode
R1	"Sprints are fixed length events of one month or less." (Schwaber and Sutherland, 2020, p. 7, para. 3)	No more than five weeks should elapse for a single sprint.	Given the data fields from the dataset, we can check this Scrum rule using the *sprint* data field. This field contains information about each sprint, including the start time and end time. However, for some of the projects, this information is not present, meaning that for those projects we can't check this rule programmatically.	1. sprint	SPRINT_START, SPRINT_END, SPRINT_DIFFERENCE, MAX_LENGTH	The Game Phase	Sprint	Developers, Product Owner, Scrum Master	☑	☐	*Drop issues that do not belong to any sprints* *Preprocess sprint values to extract sprint id, start date and end date* *Convert start date and end date as pandas datetime timestamp* *Convert sprint_id to number* Sort sprints by sprint_id MAX_LENGTH = 1 month for each **sprint** in project: *sprint_start* = // either retrieved directly from the dataset or heuristic *sprint_end* = // either retrieved directly from the dataset or heuristic sprint_difference = sprint_end - sprint_start if sprint_difference > *MAX_LENGTH* FALSE else TRUE
R2	"The Scrum Team is small enough to remain nimble and large enough to complete significant work within a Sprint, typically 10 or fewer people." (Schwaber and Sutherland, 2020, p. 5, para. 3)	The number of the Scrum Team members per project should not be **largely** less or more than 10.	**Given that the data field *assignee.name* represents the developers within the team, and the *reporter.name* and *creator.name* represent contributors to the project (developers, product owners, or people outside of the team, etc.) we can count the most active developers from these data field, thus getting clearer insights on the approximate number of developers within each project team. Unfortunately, we do not have clear data for the roles: Product Owner and Scrum Master, thus this rule can only be partially checked with the available data.**	1. sprint 3. assignee.name 4. creator.name 5. reporter.name - more data needed	N/A	The Pregame Phase	Sprint Planning, Sprint	Developers, Product Owner, Scrum Master	☐	☑	N/A
R3	"The Scrum Team consists of one Product Owner, one Scrum Master and Developers." (Schwaber and Sutherland, 2020, p. 5, para. 3)	There should be only three different roles employed within a Scrum Team.	**Given the available data fields, we can only partially check this rule. No clear and sufficient data recorded for the Product Owner and Scrum Master.**	1. assignee.name 2. creator.name 3. reporter.name - more data needed	N/A	The Pregame Phase, The Game Phase, The Postgame Phase	Sprint	Developers, Product Owner, Scrum Master	☐	☑	N/A
R4	"Working in Sprints at a sustainable pace improves the Scrum Team's focus and consistency." (Schwaber and Sutherland, 2020, p. 5, para. 4)	The duration of all sprints should follow similar pace.	Given the data fields, we can check this rule through the *sprint* data field. As mentioned previously in R1, this field contains information about the sprint start time and end time for some projects. We begin by defining a threshold duration for the sprints. Thereafter, we retrieve the start and end time of the sprint and use these measures to calculate the difference, i.e. the sprint duration. We store the duration values for all sprints in a data structure, and finally use standard deviation to check if any of the sprints' duration exceeds the given threshold.	1. sprint	THRESHOLD, SPRINT_START, SPRINT_END, SPRINT_LENGTH, LIST_LENGTHS,	The Game Phase	Sprint	Developers, Product Owner, Scrum Master	☑	☐	*Drop issues that do not belong to any sprints* *Preprocess sprint values to extract sprint id, start date and end date* *Convert start date and end date as pandas datetime timestamp* *Convert sprint_id to number* *THRESHOLD* = 1 Month sort sprints by **sprint_id** for each **sprint** in project: *sprint_start* = // either retrieved directly from the dataset or heuristic *sprint_end* = // either retrieved directly from the dataset or heuristic *sprint_length* = *sprint_end* - *sprint_start* *list_lengths*.add(*sprint_length*) if std(*list_lengths*) > *THRESHOLD* FALSE else TRUE
R5	"The Scrum Master serves the Scrum Team in several ways, including: Ensuring that all Scrum events take place and are positive, productive, and kept within the timebox." (Schwaber and Sutherland, 2020, p. 6, para. 8)	The next Sprint execution should begin only after the previous Sprint's resolution.	Given the available data fields, we check this rule using the *sprint* data field. We initally extract the start and end date for the sprints, and continue with ordering the sprints by the start date. Then, for two consecutive sprints, we check if the end date of the first sprint is later than the start date of the second sprint. As with all other rules, the output is binary, meaning that the rule can either pass (be true) or fail (false).	1. sprint	SPRINT_START, SPRINT_END,	The Game Phase	Sprint	Developers, Product Owner	☑	☐	Drop issues that do not belong to any sprints Preprocess sprint values to extract sprint id, start date and end date Convert start date and end date as pandas datetime timestamp Convert sprint_id to number order **sprints** of a project by **start_date, ascending order** sprint_start = // either retrieved directly from the dataset or heuristic sprint_end = // either retrieved directly from the dataset or heuristic for each pair of consecutive **sprints** (sprint_1, sprint_2) that belongs to a project: if sprint_1.**end_date** > sprint_2.**start_date** FALSE else TRUE
R6	"The Product Owner is also accountable for effective Product Backlog management, which includes: Developing and explicitly communicating the Product Goal; Creating and clearly communicating Product Backlog items; Ordering Product Backlog items; Ensuring that the Product Backlog is transparaent, visible and understood." (Schwaber and Sutherland, 2020, p. 5, para. 1) "Product Backlog refinement is the act of breaking down and further defining Product Backlog items into smaller more precise items. This is an ongoing activity to add details, such as a description, order, and size. Attributes often vary with the domain of work." (Schwaber and Sutherland, 2020, p. 10, para. 9)	There should be a project clarity identifier attached to each issue within the sprints.	Given the available data, we check this rule by first grouping *issues* based on the *sprints* they belong to. Afterwards, we check if the issues have a project identifier associated with them, through the *project* data field. Rule fails for the issues having no project identifier, and passes for the other cases.	1. project 2. sprint 3. key	N/A	The Game Phase	Sprint Planning, Sprint	Product Owner, Developers	☑	☐	groupby **issues** per **sprint** for each **sprint** that belongs to the project: print (grouped **issue**['**project_id**']) for each **key** if(**issue**.**project** == null and issue != null) FALSE else TRUE
R7	"A new Sprint starts immediately after the conclusion of the previous Sprint." (Schwaber and Sutherland, 2020, p. 7, para. 5)	No considerable amount of time should elapse between the finish of a sprint and the beginning of the new sprint.	Given the available data, we check this rule by first definind a maximum amount of time, in this case of 1 week. We extract the start and end time for each sprint, which we later use to order the sprints by (ordering by start time). Then, for a pair of consecutive sprints, we check if the time from the first sprint end time til the next sprint start time is more than the defined maximum duration, in which case the rule fails for the given project. In the opposite case, the rule will be true.	*1. sprint*	MAX_TOLERANCE, SPRINT_START, SPRINT_END	The Game Phase	Sprint	Developers	☑	☐	*Drop issues that do not belong to any sprints* *Preprocess sprint values to extract sprint id, start date and end date* *Convert start date and end date as pandas datetime timestamp* *Convert sprint_id to number* *MAX_TOLERANCE* = 1 week sprint_start = // either retrieved directly from the dataset or heuristic sprint_end = // either retrieved directly from the dataset or heuristic order **sprints** of a project by start_date for each pair of consecutive **sprints** (sp1, sp2) of the **project**: if sp2.start_date - sp1.end_date > *MAX_TOLERANCE* FALSE else TRUE

R8	"The Developers are always accountable for: Adapting their plan each day toward the Sprint Goal." (Schwaber and Sutherland, 2020, p. 5, para. 7)	There should not be a considerable amount of time for a developer to volunteer and start a new issue after she/he has completed the previous one per each sprint.	Given the available data, we check this rule using the fields of *asignee.name*, *created* and *resolutiondate*. The first field represents the developer who is assigned to a specifc issue. We first define a maximum duration of 3 hours. We then group the issues by the *sprints* and/or *developers*. For each pair of consecutive issues assigned to the same developer, the rule fails if the difference between the start time of the second issue and completion time of the first issue is greater than the defined maximum. Otherwise, thr rules passes for the given project.	*1. sprint* *2. key* *3. assignee.name* *4. created* *5. resolutiondate*	MAX_TOLERANCE, DIFF_ISSUE_TIME,	The Game Phase	Sprint	Developers	☑	☐	*Retain only active developers (as per R27)* *Drop incomplete issues* *Drop issues belonging to no sprints* *MAX_TOLERANCE* = 2 days order **sprints** of a project by **sprint**, **start_date, issue** groupby **issues** per **sprint** AND **developer** print(**key**['**developer**', '**start_date**', '**end_date**']) for each pair of consecutive issues (issue_1, issue_2) assigned to 1 developer if(issue_2.start_date - issue_1.end_date) > *MAX_TOLERANCE* FALSE else TRUE
R9	"The Developers are always accountable for: Creating a plan for the Sprint, the Sprint Backlog." (Schwaber and Sutherland, 2020, p. 5, para. 7)	Active members of the development teams should be included in additional activities, other than development.	**Given the data fields from the dataset, we can check for this rule by inspecting the active developers in the *assignee.name* field, and checking whether any of those developer names is to be found on the *creator.name* data field, which indicates the roles responsible for planning the Sprints artifacts and activities. Unfortunately, we can only partially check this rule, since we are not able to identify what those additional activities are because we have a limit amount of data to explore.**	*1. sprint* *2. key* *3. asssignee.name* *4. creator.name* *- need more data*	N/A	The Pregame Phase	Sprint Planning	Developers	☐	☑	N/A
R10	"The Daily Scrum is a 15-minute event for the Developers of the Scrum Team." (Schwaber and Sutherland, 2020, p. 9, para. 4)	Daily Standups/Scrums should take no more than around 15 minutes.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Daily Scrum	Developers, Product Owner, Scrum Master	☐	☑	N/A
R11	"Daily Scrum is held at the same time and place every working day of the Sprint." (Schwaber and Sutherland, 2020, p. 9, para. 4)	Per each Sprint, there should be a constant time and place when/where the daily standup-s take place.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Daily Scrum	Developers, Product Owner, Scrum Master	☐	☑	N/A
R12	"The Sprint Review is timeboxed to a maximum of four hours for a one-month Sprint. For shorter Sprints, the event is usually shorter." (Schwaber and Sutherland, 2020, p. 9, para. 10)	The Sprint Review event should take no more than around 4 hours for longer Sprints (one month), and even less for shorter Sprints.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Sprint Review	Developers, Product Owner, Scrum Master, Stakeholders	☐	☑	N/A
R13	"The Sprint Review is the second to last event of the Sprint." (Schwaber and Sutherland, 2020, p. 9, para. 10)	All Sprint iterations/increments should be over by the time the Sprint Review begins.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Sprint Review	Developers, Product Owner, Scrum Master, Stakeholders	☐	☑	N/A
R14	"The Sprint Retrospective concludes the Sprint." (Schwaber and Sutherland, 2020, p. 10, para. 4)	All Scrum Events should be over by the time Sprint Retrospective begins.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Sprint Retrospective	Developers, Product Owner, Scrum Master	☐	☑	N/A
R15	"The Sprint Retrospective is timeboxed to a maximum of three hours for a one-month Sprint. For shorter Sprints, the event is usually shorter." (Schwaber and Sutherland, 2020, p. 10, para. 4)	The Sprint Retrospective event should take no more than around 3 hours for longer Sprints (one month), and even less for shorter Sprints.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Sprint Retrospective	Developers, Product Owner, Scrum Master	☐	☑	N/A
R16	"For each selected Product Backlog item, the Developers plan the work to create an Increment that meets the Definition of Done." (Schwaber and Sutherland, 2020, p. 8, para. 9)	There should be an agreed Definition of Done for each increment.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Sprint Planning	Developers	☐	☑	N/A
R17	"The Sprint Goal, the Product Backlog items selected for the Sprint, plus the plan for delivering them are together referred to as the Sprint Backlog." (Schwaber and Sutherland, 2020, p. 9, para. 1)	Spring Backlog contains the Sprint Goal, PBIs and PBI implementation plan.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Sprint Planning	Developers	☐	☑	N/A

R18	"Multiple Incrementes may be created within a Sprint." (Schwaber and Sutherland, 2020, p. 12, para. 1)	There should be at least one Increment deriving from a sprint.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Product Backlog	Developers	☐	☑	N/A
R19	"The moment a Product Backlog item meets the Definition of Done, an Increment is born." (Schwaber and Sutherland, 2020, p. 12, para. 4)	The Product Backlog Items should fulfill the *Definition of Done* in order to be considered as an Increment.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Increment	Developers	☐	☑	N/A
R20	"If the Definition of Done for an increment is parat of the standards of the organization, all Scrum Teams must follow it as a minimum." (Schwaber and Sutherland, 2020, p. 12, para. 6)	Scrum Team members and mostly developers should adhere to the organization-wide agreed DoD for the PBIs implementation.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	**Event**: Sprint **Artifact**: Increment	Developers, Product Owner, Scrum Master	☐	☑	N/A
R21	"If there are multiple Scrum Teams working together on a product, they must mutually define and comply with the same Definition of Done." (Schwaber and Sutherland, 2020, p. 12, para. 7)	In case of more than one collaborative Scrum Teams, cross-team members should adhere to the mutually agreed DoD.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	**Event**: Sprint **Artifact**: Increment	Developers, Product Owner, Scrum Master	☐	☑	N/A
R22	"The Sprint Review should never be considered a gate to releasing value." (Schwaber and Sutherland, 2020, p. 12, para. 1)	An Increment may be delivered to stakeholders prior to the end of the Sprint.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	**Event**: Sprint Review **Artifact**: Increment	Developers, Product Owner	☐	☑	N/A
R23	"The Product Owner is one person, not a committee." (Schwaber and Sutherland, 2020, p. 6, para. 4)	No more than one Product Owner should be employed in the Scrum Team.	**Not enough data available to check for this rule.**	N/A	N/A	The Pregame Phase (planning the development team)	Sprint	Product Owner	☐	☑	N/A
R24	"The Product Owner is also accountable for effective Product Backlog management, which includes: Developing and explicitly communicating the Product Goal; Creating and clearly communicating Product Backlog items; Ordering Product Backlog items; Ensuring that the Product Backlog is transparaent, visible and understood." (Schwaber and Sutherland, 2020, p. 6, para. 1)	The backlog does not contain meaningless or empty issues. **I consider meaningless issues the issues that belong to no project and have no issue body, description, start and end time and other details that are relevant to developers and other roles.** Product Backlog Items should be understandable, therefore should contain a clear description, name, priority, and be identified correctly.	Given the available data, we can check this rule using the columns of *description, issuetype.name, priority.name, summary, storypoints, key, project*. Since all of these rules display information about the issues reported in Jira, we can check for issues that are against the principles of transparency, visibility and understandability. Therefore, we will report the issues that are not clearly reported and understandable, or issues having no clear description, type, summary, priority, belonging to no sprints or project. We check this rule by grouping all issues by the sprint they belong to. Afterwards, for each issue in the given sprints, we check (manually) in case the aforementioned attributes are not present, thus rendering the issue as unclear or irrelevant for the developers.	1. project 2. sprint 3. key 4. resolutiondate 5. description 6. issuetype.name 7. priority.name 8. summary	N/A	The Game Phase	Sprint	Product Owner, Developers	☑	☐	*NB: not all projects used storypoints to assign to the tasks. This attribute is not considered for projects who had the major number of issues without storypoints.* for each **sprint** that belongs to the **project**: groupby **issue** per **sprint** print (grouped **issue**['**project_id**', '**description**', '**issue_type**', '**summary**', '**storypoints**', '**priority**']) for each issue: if((**issue.project_id** OR **issue.start_date** OR **issue.end_date** OR **issue.description** OR **issue.summary** OR **issue.issue_type** OR **issue.project_priority)** == null): FALSE else TRUE
R25	"If the Product Owner or Scrum Master are actively working on items in the Sprint Backlog, they paraticipate as Developers." (Schwaber and Sutherland, 2020, p. 9, para. 4)	Product Owner and/or Scrum Master should contribute towards the Sprint Goal in order to be part of Daily Scrum meetings.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Daily Scrum	Product Owner, Scrum Master	☐	☑	N/A
R26	"Only the Product Owner has the authority to cancel the Sprint." (Schwaber and Sutherland, 2020, p. 8, para. 3)	In case of cancelled Sprints, Product Owner should be the only role within the Scrum Team that can do that.	**Not enough data available to check for this rule.**	N/A	N/A	The Game Phase	Sprint	Product Owner	☐	☑	N/A

R27	Proposed Rule: Development teams consist of approximately 8 developers. Based on R2 and R3.	No more than 8 active developers should be involved in development tasks.	Given that the data field *assignee.name* indicates the developers in the team, we can count the unique active assignees to whom the tasks are assigned. We begin by defining a threshold value as a maximum number of developers needed per sprint. Then, for each sprint in the given project, we check if the number of active developers exceeds the maximum, in which case the rule fails. In other cases, the rule passes for the given project.	*1. sprint* *2. assignee.name*	MAX_DEVS, NO_DEVS	The Pregame Phase	Sprint	Developers	☑	☐	*rename column of **assignee.name** to **developers*** *no_devs = retain only the active developers in each team* *MAX_DEVS* = 8 for each **sprint** in project: *no_devs* = // retrieve the number of unique assignees in the sprint group **developers** by each **sprint** if ***no_devs*** > ***MAX_DEVS***: FALSE else TRUE
R28	*Proposed Rule:* The development lifecycle must not include unusual workflow or changes in the issue statuses. For instance, an issue marked as 'complete' should not re-appear with a status of 'unresolved' or 'open' after it has been completed and closed.	The status of issues should always follow the agreed workflow, depending on the project and development team.	Given all the available data, we check this rule using the fields: *status. name, status.id* and *status.statusCategory.name*. Since these two columns convey necessary information about the status of the tasks, we first group the *issues* by their *sprints*, and order them by the issue name. Afterwards, for each issue, we print the name, id and category of the status. For this rule, we make use of the changelog data, in which we check for unusual transitions in the issue statuses, for instance: an issue obtaining the 'in-progress' status after it has been 'closed' or 'completed'.	*1. sprint* *2. key* *3. status.name* *4. status_id* *5. statusCategory_name*	PROJECT_WORKFLOW	The Game Phase	Daily Scrum	Developers	☑	☐	*drop incomplete issues* *drop issues that belong to no sprints* **PROJECT_WORKFLOW** = extract issue statuses per each project groupby **issues** by the **sprints** order **issues** by **issue_name, start_date** for each **issue**: only retain **issues** whose status is found in **PROJECT_WORKFLOW** print(**issue**['**status_name**', '**status_id**', '**statusCategory_name**']) if(**issue**.**status_name** OR **issue**.**status_id** OR **issue**.**statusCategory_name** == null): FALSE else TRUE
R29	*Proposed Rule:* The Scrum Team does not include any other people (e.g. a manager who doesn't do tasks).	There should not be any unexpected or random person involved at a rather strange point/part of the Scrum activities, phases or events.	**Given all the data fields from the dataset, we can check for this rule by inspecting the columns of assignee.name, creator.name, reporter.name, key, sprint, and project. We will first cluster the different projects and the sprints associated with each project. The same applies for grouping the issues based on the sprint they belong to. Finally, we can list the unique names of the people who are employed in the Scrum activities and check for their contributions and obligations (who was the issue reporter, issue creator and issue assignee) in order to check whether there are persons that come into the picture in an unexpectedly uncommon and unusual point/part of the development process.** **Unfortunately, we do not have sufficient data to check for all Scrum Roles or activities, therefore, this rule can only be checked partially.**	1. sprint 2. key 3. assignee.name 4. creator.name 5. reporter.name - need more data	N/A	The Game Phase	Sprint	Developers, Product Owner, Scrum Master	☐	☑	N/A
R30	*Proposed Rule:* Issues resolved in previous increments do not appear again in future increments (rework).	No previously resolved issue should reappear in a future sprint.	Given all the available data, we check this rule through the columns of *key, sprint*. Firstly, we will group the issues that belong to their specific sprints by using the aforementioned columns. Afterwards, we will check whether completed issues belonging to the very first sprint for instance, have reappeared in an upcoming sprint during the development phase. By identifying such behavior, this rule fails for the given project.	1. sprint 2. key 3. created 4. resolutiondate	N/A	The Game Phase	Sprint	Developers	☑	☐	group **issues** by **sprint** order **issues** by **issue_name, start_date** for each single **issue_x** print(**issue_x**['**name**', '**status**', '**start_date**', '**end_date**', '**sprint**']) if(issue_x in sprint_1 is complete AND issue_x is in sprint_2): FALSE else TRUE
R31	*Proposed Rule:* Each Scrum Sprints can be considered a short project, thus they are uniquely identified.	There should be a unique identifier/name associated with each Sprint.	Given all the available data, we check this rule using the values present in the *sprint* data field. First, we create a list which will store all the names of unique sprints in the given project. Then, for each existing sprint, we check whether its name can be found in the already created list containing all sprint names in the project.	*1. sprint*	SPRINT_NAMES	The Game Phase	Sprint Planning	Product Owner	☑	☐	*SPRINT_NAMES* = non_empty_string for each **sprint** in project: print(**sprint**['name']) if(**sprint**['name'] is not in *SPRINT_NAMES*): FALSE else TRUE
R32	*Proposed Rule:* Issue have a corresponding type (bug, task, etc.).	There should be a type, such as bug, improvement or task, associated to each issue.	Given the available data, we check this rule using the columns: *issuetype.name, key* and *sprint*. First, we create a list which stores all the different types an issue can get, such as: bug, story, task, etc. depending on the project. Then, after grouping the issues by the sprint field and ordering the issues by their names, we can verify whether for each issue, is its type not empty and contained in the already defined list. In cases this is not valid, the rule fails for the given sprint/project. The occurence of such missing values indicates that the development team does not know what type of issue they are about to deal with, and that the PO did not specify well the issue details, which as per the agile principles, is not satisfactory. We also check and report the most frequent issue types as well as the least occuring ones.	*1. sprint* *2. key* *3. issuetype.names*	ISSUE_TYPES_PER_PROJECT	The Game Phase	Sprint Planning	Product Owner	☑	☐	*ISSUE_TYPE_PER_PROJECT* = ['bugfix', 'feature', 'refactor', ...] group **issues** by **sprint** order **issues** by **issue_name** for each **issue** in **sprint**: if(**issue**[**issuetype.name**] is not in *ISSUE_TYPE*ISSUE_TYPE_PER_PROJECT: FALSE else TRUE
R33	*Proposed Rule:* Each issue belongs to a specific sprint.	There should be a sprint identifier attached to each issue.	Given the available data, we check this rule taking into account only the issues that already have a type name. We begin by creating a list containing all the sprint identifiers available in the given project. After checking the column *key*, and simultaneously verifying that the *sprint* identifier is not present in list, we can uncover the issues that do not belong to any of the sprints in the given project.	*1. sprint* *2. key*	SPRINT_NAMES	The Game Phase	Sprint Planning	Product Owner	☑	☐	*SPRINT_NAMES* = [extract sprint names from **sprint** field] group **issues** by **sprint** order **issues** by issue_name for each **issue** in **sprint**: if(**issue**['sprint_name'] is not in *SPRINT_NAMES*): FALSE else TRUE
R34	*Proposed Rule:* All issues must be uniquely identifiable.	There should be a unique identifier associated with each issue.	Given the available data, we check this rule utilizing the column: *key*, which indicates the unique identifier of a specific issue. Since all agile-managed projects with Jira automatially create IDs for each issue (task, bug, etc), we will first group issues by their sprints and order by their name. Later, we filter out the issues that eventually do not contain any *key* or *ID*. This rule can be later beneficial while tracking the changing status of the issues, which is needed to check other rules as well.	*1. sprint* *2. key*	N/A	The Game Phase	Sprint Planning	Product Owner	☑	☐	group **issues** by **sprint** order **issues** by issue_name for each **issue** in **sprint**: if(**issue**['issue_name'] == null): FALSE else TRUE

R35	*Proposed Rule:* Scrum Sprints have a starting time and date.	There should be timestamp indicating the sprints kick-off.	Given the available data, we check this rule using the field *sprint*, as this column contains information indicating the starting date and time of the sprints. However, before doing that, we order the sprints by their names. Then, for each sprint in the project, we check whether sprints that have issues developed during them, have a valid start date, in what case this rule passes for the given project and fails if otherwise.	*1. sprint*	SPRINT_START_DATE	The Game Phase	Sprint Planning	Product Owner	☑	☐	*Drop issues that do not belong to any sprints* *Preprocess sprint values to extract sprint id, start date and end date* *Convert start date and end date as pandas datetime timestamp* order **sprints** by sprint_name **sprint_start_date** = // either retrieved directly from the dataset or heuristic for each **sprint** in project: if(**sprint_start_date** == null): FALSE else TRUE
R36	*Proposed Rule:* Scrum Sprints have a completion time and date.	There should be timestamp indicating the sprints completion.	Given the available data, we check this rule using the field sprint, as this column contains information indicating the end date and time of the sprints. However, before doing that, we order the sprints by their names. Then, for each sprint in the project, we check whether sprints that have issues developed during them, have a valid end date, in what case this rule passes for the given project and fails if otherwise. In case there is no end date recorded in the *sprint* field, we consider the *start* time of the next sprint to be the *end* time for the current sprint.	*1. sprint*	SPRINT_END	The Game Phase	Sprint Retrospective	Product Owner	☑	☐	*Drop issues that do not belong to any sprints* *Preprocess sprint values to extract sprint id, start date and end date* *Convert start date and end date as pandas datetime timestamp* order **sprints** by sprint_name **sprint_end_date** = // either retrieved directly from the dataset or heuristic for each **sprint** in project: if(**sprint_end_date** == null): FALSE else TRUE
R37	*Proposed Rule:* The number of PBIs selected from the Backlog for a Sprint depends on the developers, but it should not be zero PBIs per sprint.	There should be a minimum of one issue, representing a Sprint Backlog Item, per each Sprint.	Given the available data, we check this rule using the fields of *sprint* and *key*. We begin by defining a variable which serves as the *minimum threshold*. Then, we group all issues by their sprint and order them by their names. After that, for each sprint, we *count* the number of issues present in the given sprint and store this *count* in another variable *issue_count*. Lastly, we check if the *issue_count* variable is less than the minimum threshold value, in which the sprint will fail for the given project, and pass if otherwise.	*1. sprint* *2. key*	MIN_ISSUES, ISSUE_COUNT,	The Game Phase	Sprint Planning, Sprint	Product Owner	☑	☐	*MIN_ISSUES* = 1 group **issues** by **sprint** order **issues** by issue_name for each **sprint**: *issue_count* = COUNT (number_of_issues_per_sprint) if(*issue_count* < *MIN_ISSUES*): FALSE else TRUE
R38	*Proposed Rule:* "The Product Owner is also accountable for effective Product Backlog management, which includes: Developing and explicitly communicating the Product Goal; Creating and clearly communicating Product Backlog items; Ensuring that the Product Backlog is transparaent, visible and understood." (Schwaber and Sutherland, 2020, p. 5, para. 4) "Product Backlog refinement is the act of breaking down and further defining Product Backlog items into smaller more precise items. This is an ongoing activity to add details, such as a description, order, and size. Attributes often vary with the domain of work." (Schwaber and Sutherland, 2020, p. 10, para. 9)	There should be timestamp indicating the issue development kick-off.	Given the available data, we check this rule using the column *created*, as this column indicates the starting time of the issues. We group issues by their sprints and order by the start time. Then, for each issue within the corresponding sprints, we check for issues that do not have a starting time, i.e. a valid timestamp value in the *created* data field.	*1. sprint* *2. key* *3. created*	N/A	The Game Phase	Sprint Planning	Product Owner, Developers	☑	☐	group **issues** by **sprint** order **issues** by **start_date** for each **issue** in **sprint**: if(**issue['start_date']** == null): FALSE else TRUE
R39	*Proposed Rule:* "The Product Owner is also accountable for effective Product Backlog management, which includes: Developing and explicitly communicating the Product Goal; Creating and clearly communicating Product Backlog items; Ensuring that the Product Backlog is transparaent, visible and understood." (Schwaber and Sutherland, 2020, p. 5, para. 4) "Product Backlog refinement is the act of breaking down and further defining Product Backlog items into smaller more precise items. This is an ongoing activity to add details, such as a description, order, and size. Attributes often vary with the domain of work." (Schwaber and Sutherland, 2020, p. 10, para. 9)	There should be timestamp indicating the issue development completion. Even though Jira itself makes sure that complete issues have a completion date and time, there are high amounts of incomplete issues present in the data of each project. According to Scrum Guide "All incomplete Product Backlog Items arere-estimated and put back on the Product Backlog. The work done on them depreciatesquickly and must be frequently re-estimated", having incomplete issues in the backlog is not a good practice.	Given the available data, we check this rule using the column *resolutiondate*, as this column indicates the completion time for the issues. We group issues by their sprints and order by the start time. Then, for each issue within the corresponding sprints, we check for issues that do not have a starting time, i.e. a valid timestamp value in the *resolutiondate* data field.	*1. sprint* *2. key* *3. created* *4. resolutiondate*	N/A	The Game Phase	Sprint	Product Owner, Developers	☑	☐	group **issues** by **sprint** order **issues** by **start_date** for each **issue** in **sprint**: if(**issue['end_date']** == null): FALSE else TRUE