Method for Identification of Waste in the Process of Software Development in Agile Teams Using Lean and Scrum

Márcio Trovão Bufon(✉) and Adriano Galindo Leal(✉)

Institute of Technological Research - IPT, São Paulo, Brazil
mubuffon@gmail.com, leal@ipt.br

Abstract. Waste in software development projects is defined as anything that consumes resources such as time, effort, room, and money without adding value to the customer. Methods and techniques to identify waste indicators, which are specific for each project, are applied to part of total interactions and the development phases; and spend analysts and developers' time and effort. Therefore, this paper aims to define a method to identify waste within the software development process in Scrum teams, from data based on JIRA tool, which supports software development planning, management and controlling activities. According to the bibliographic review are defined: (i) indicators for types of waste according to Lean software development principles; (ii) JIRA's attributes, mathematical operators, keywords, functions and reports related to such indicators. In the proposed method are defined requirements that establish a semantic relation between each indicator variables and formulas to the set of JIRA'S attributes, functions and keywords and, based on them, queries in JIRA Query Language are implemented to quantify the indicators. The method validation is performed using graphics that show queries results classified and grouped by project, indicator and type of waste, acquired from a software project base for a company in the Brazilian financial market. Through the quantitative analysis of results, it is possible to suggest a hypothesis for the occurrence of the types of observed wastes.

Keywords: Waste · Scrum · Lean · JIRA

1 Introduction

In software development processes, waste is defined as any action that interferes with delivering to the customers what they value at the time and place where this value is applicable. Doing what adds no value or what is not immediately necessary, generating delays, delivering a defective feature and consuming time, effort, room and financial resources without adding value to the customer are some examples of waste [1].

One way to eliminate waste in software development processes in dynamic teams that employ the Scrum method is to apply Lean Software Development (LSD) [2] principles and practices. Adapted from the Lean Manufacturing Thinking and the Toyota Production System, LSD defines as waste in software development projects partially finished or unfinished work, delivery of non-requested features or extra

L. Uden et al. (Eds.): KMO 2019, CCIS 1027, pp. 466–476, 2019.
https://doi.org/10.1007/978-3-030-21451-7_40

functionalities, relearning, control transfer, task switching, delays and delivery of defective software [3].

Aiming to measure such waste, indicators are defined to assess the software development flow continually [4]. These indicators allow to identify bottlenecks, to follow the ongoing development of requirements and to draw up a costs template that represents the perspectives related to investment; work performed and waste, separately. In software development projects that use Kanban it is possible to identify waste by mapping the sequence of activities [5]. For Scrum projects, waste is identified and eliminated during the planning phases by recognising unnecessary features or functionalities whose cost or deadline for development are higher or longer than the expected for the project. In [6], authors propose the identification and elimination of defects and the resulting reduction of waste in Scrum software projects using monitoring the development phase as well as the evolution of the system architecture quality. Furthermore, the identification of metrics and waste indicators are more effectual when combined with software tools to plan, manage and control the development process [2]. Also, the use of LSD analytical tools allows the application of Lean concepts and principles to software engineering processes [4]. Finally, in [7] authors recommend the development of applications aiming to evaluate the application of LSD concepts interactively.

Based on these statements and the growing application of such tools by dynamic teams [8]; in order to waste identification be an efficient process, it must be performed by, and integrated into, the same tools that provide support for the software development process planning, managing and control activities.

Therefore, the objective of this work is to propose a method to evidence indicators of waste from the data, graphs and reports available in the agile planning tool JIRA itself. The main contribution of the proposed method is to reduce considerably the effort required to identify the waste in software development projects, thus avoiding that the effort necessary to carry out such activities can be considered wasteful.

2 Lean Software Development – LSD

2.1 Principles

Lean Software Development expands the agile development foundations since it applies Lean principles to software development, and its crucial principle: **to eliminate waste** [1, 3, 9]. This principle involves total elimination of waste by identifying in a timeline the period from the customer's request to the actual delivery of value to the customer. Every activity that adds no value must be removed from the timeline.

The timeline starts when the development team, comprised of requirement and system analysts, developers, architects, software engineers, testers and so forth, receives a request to solve a set of the customer's need and finishes when the software is implemented in the production environment.

2.2 Identify and Eliminate Waste

There are two steps required to eliminate waste in the software development process. The first step involves visualising the types of waste and identifying their sources. The second step is the elimination of significant waste sources identified in the first step. These steps are repeated until all sources and wastes caused by them are removed [3]. These authors have adapted, for software development, the seven types of waste identified in the Toyota Production System. Table 1 shows these types of waste according to the Lean Software Development (LSD).

Table 1. Comparison between types of waste according Manufacturing and Lean Software Development.

The seven wastes of manufacturing	The seven wastes of software development
Inventory	Partially done work
Extra processing	Extra processes
Overproduction	Extra features
Transportation	Task switching
Waiting	Waiting
Motion	Motion
Defects	Defects

Each type of waste can be defined as [1, 9]:

- **Partially Done Work:** Refers to unfinished or partially finished software, that becomes obsolete, that hides quality problems or is subject to constant requests for changes. The stock of requirements to be solved represents the unfinished work.
- **Extra Processes:** Loss of knowledge and of processes created and executed between the development cycles, generating rework or the definition of other work processes to those mapped in the value chain.
- **Extra Features:** Inclusion of extra features to the system that was not initially mapped in the requirements, providing for a future need. These features add some technical capacity that will not be observed or that is not required for the work to be performed by the final user.
- **Task Switching:** Software development requires in-depth analysis, focusing on the complexity of what needs to be developed. When a software developer changes constantly between two different tasks, distraction can occur, adversely affecting the results of both tasks.
- **Waiting:** Waiting for documents to be signed, meetings to be held, machines to be cleared, for the availability of development environments, for tests to be performed or the approval of software versions, as well as waiting for people who are working on other areas or for the approval of changes, cause delays in the beginning of the software development process, increasing the inventory items to be developed and not delivering value to the final user.

- **Motion:** When software collaborators alternate between different projects, effort and time are required to understand the workflow and the context of the new project. At each transfer of control, there is a loss of knowledge or knowledge is limited to the person who creates it.
- **Defects:** Defects not identified in the first test cycles of software cost more to be corrected when already delivered in controlled environments for integration tests and production. The delivery of versions that repeatedly show defects is being produced by a defective software development process.

The specific software development process used in agile teams can cause waste for being excessively complex or for having characteristics of traditional development methods, such as the delivery of a large set of software functionalities after long periods of development without performing intermediary deliveries, developers with no autonomy to decide on technical questions associated to the requirements or to software quality attributes, adoption of new technologies, frameworks or processes without proper scientific analysis [1].

Additionally, development teams that not working the architecture or the refactoring the code during interactions aiming to eliminate or reduce technical debt before the end of the software project can cause wastes of defects and extra processes.

2.3 Value and Value Stream Mapping

Complying with the Lean fundamental principle of eliminating waste is only possible with the identification of what value is and with the mapping of work process flow to generate this value, referred to as value flow [3].

Value, under Lean context, is delivering to the customer what they effectively requested, within the minimum time possible, with quality and at the price they are willing to pay, while the value flow is comprised of all processes required to generate value, from the design to the delivery and utilization by the customer [10].

In [11], authors state that mapping of value flow identifies as types of waste only delays, task switching between collaborators and relearning, while [12] authors suggest that, to identify these and other types of waste, it is necessary an approach based on two different steps. The first step is the application of indicators to the existing value flow, and the second step is a detailed mapping of flows that show the worst results per indicator. In this way, it is possible to avoid the thorough mapping of flows that have no meaningful impact on value generation.

2.4 Waste Indicators

Petersen and Wohlin (2011), suggested waste indicators such as the identification of a number of bottlenecks (NB) considering the delivery time of a requirement and the number of accumulated items (NAI) of requirements for each collaborator in each development phase.

The volume of work in progress (WIP), has been used as a waste indicator regarding definitions, revisions or architecture refactoring [6]. Based on the WIP, it was proposed and applied corrections to the development flow reducing the waiting time to

start the development of stories that depended on the architectural definition and on the amount of rework required to correct defects caused by architecture faults.

3 Method

The method suggested in this study relates, in four phases, waste indicators to charts and reports in the agile planning tool of JIRA.

The first phase in this method is the specification of requirements of queries in natural language, considering the definition and breakdown of each indicator into variables and mathematical operators. It was created the requirements in natural language (RNL) board for each indicator and type of waste. In each RNL board, containing one or more requirements, it was selected the keywords used in the second phase of the method.

In the second phase, it is established the semantic relation between RNL and JIRA elements. Columns A and B are filled out respectively with the indicator variables and mathematical operators and, subsequently, with JIRA sets of attributes, keywords, JQL functions, reports and graphics, whose meanings are related to the indicator according to the model defined in Table 2.

Table 2. Example of the board of requirements in natural language.

RNL_NB_PROJECT_01	REQUIREMENT Number of bottlenecks
RNL1.1 – When using the Control Flow Diagram (CFD), the start and end dates of each Sprint must be specified as the period, excluding from the analysis issues that are already in the Done phase (status Resolved, Closed). **Keywords for semantic relations:** • period; • *Sprint*; • issues; • phase; • status;	

The third phase is the Structured Query Specification in JQL (SQSJQL), considering the items produced in phases 1 (RNL) and 2 (TSR) (Table 3).

In the 4th phase, the JQL query code defined in each SQSJQL is saved as a filter in order to be reused in control panels created in JIRA, which may contain different gadgets, such as graphics and JQL queries results, data on version builds and code coverage. For the validation of this method, it is used only for graphics and report gadgets (Fig. 1 and Table 4).

Table 3. Example of table of semantic relation.

Table of semantic relation – (TSR)	
Column A – Indicator N	Column B - *JIRA*
Variables • variable1 • variable2 • variableN	**Set of attributes or keywords** • attribute1 • attribute2 • attributeN • keyword1 • keyword2 • keywordN
Mathematical Operators • operator1 • operator2 • operatorN	**Set of functions JQL** • function1 • function2 • functionN
	***JIRA* Graphics** • graphic1 • graphic2 • graphicN

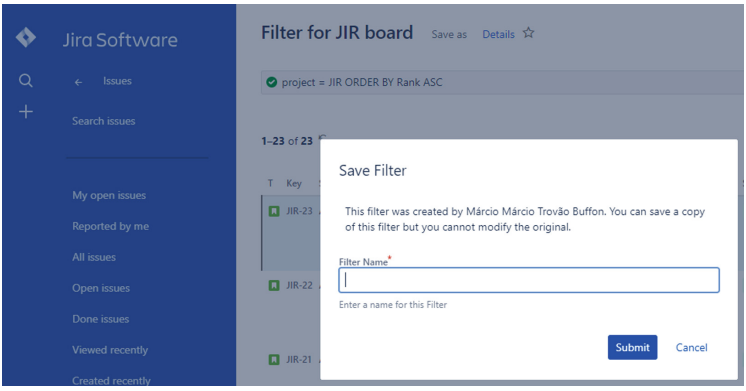

Fig. 1. Persistent filters from a JQL query

4 Method Validation and Analysis of Results

In order to validate the method, it was considered JIRA data, version v7.6.0, of a software development project, referred to in this study as PROJECT_01, for a financial market company in Brazil. The PROJECT_01 was created in JIRA as Scrum type, creating Kanban boards for control and follow-up of project activities regarding specification, implementation, tests and deployment of requirements in each Sprint.

Scrum is an agile development method designed by Jeff Sutherland in the early 1990s and later updated by Cohn (2009), Schwaber (2004) and Beedle (2001). In it, the software development process comprises the activities of requirements, analysis,

Table 4. Board **2.** Example of Structured Query Specification.

SQSJQL(VE_TI) • **RNL1.1** • **RNL1.2**	**STRUCTURED QUERY SPECIFICATION** **JQL QUERY** **Value of Efficiency for Unfinished Work**
Description	Returns by project, phase and period, the number of ISSUES.
Inputs	project identifier, identification of the phase, start date and end date or period (day, month or year), ISSUE status.
Outputs	List in Table format with the number of ISSUES, containing the Created columns, ISSUE_Key, Status, Summary, Assignee.
Constraints	The output of the query must be compatible with the expected input of the types of graphics or reports defined in step 2.
JQL code	project in ("<id_project>") and status in (Open, "On-Hold") and *Created*Date <= <data_final>
Filter Name	LSD - WASTE – VE_TI
Sort	Ascending Query Records by CreatedDate

design, evolution and delivery. These activities occur within a process pattern called Sprint, and prioritized requirements that generate customer value are added to a list called Backlog. Daily meetings of up to 15 min (stand-up meetings) allow Scrum teams to reveal problems or impediments in the development process that are reported to a team leader, called the Scrum Master. The Scrum Master is also responsible for defining with the project owner the list of requirements prioritized in the Backlog (requirements inventory) [13].

The data gathering period was restricted between SPRINT 02 (06/22/2018) starting date and the end of the third week of SPRINT 04 (09/06/2018). During the analysis period, the total of collaborators and their profiles were distributed into nine collaborators, more specifically, 01 Manager, 01 Scrum Master, 03 requirement analysts and 04 developers.

4.1 Number of Accumulated Items (NAI)

NAI allows you to identify whether there is a continuous workflow between Sprints or software versions. Over time, it is expected that its value will decrease, resulting in a reduction in the number of backlog requirements. Figure 2 represents the control panel created for this indicator in JIRA.

In PROJECT_01, dividing the number of delivered requirements by the number of weeks in each Sprint results in an average of 4.2 requirements delivered per week. Similarly, but this time dividing the number of open requirements by the number of

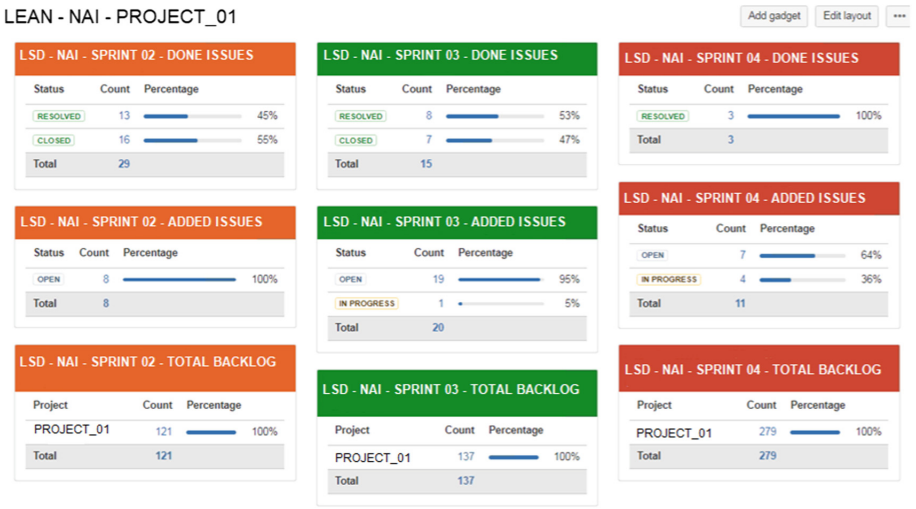

Fig. 2. Persistent filters from a JQL query.

weeks in each Sprint, results in an average of 3.5 requirements added to the stock per week. Considering that the difference between these average values is approximately 1 REQUIREMENT delivered per week and if there is no change in this difference towards the requirements delivered in the period, it will be necessary 277 weeks or 69 Sprints so that the stock of requirements is settled.

4.2 Work in Progress (WIP)

Under the context of PROJECT_01, the WIP has been adjusted to quantify all requirements that deviate from the value flow, checking the number of transitions between the phases and the status that, for each REQUIREMENT, deviate from the value flow mapping described in Fig. 3.

Considering the values of each SQSJQL, it is possible to calculate the WIP percentage that deviates from the requirements and the defects grouped by Sprint according to Table 5.

According to the result of Table 5, it is possible to observe that the work in progress deviation for requirement tasks tends to double at each new Sprint, while the percentage of defects after a drop between SPRINT 02 and SPRINT 03, remains stable. Based only on the percentages presented it is not possible to state how much the deviations to the value flow affect the developers and analysts' efforts.

However, after categorizing the WIP deviation in requirements and defects and grouping them by Sprints, it was possible to identify the wastes related to unfinished work - it happened in tasks moved from Sprints Sprint 07/05 -18/05 and Sprint 11/06 - 22/06 to the Sprints within the analysis period; extra features – occurred when there were several types of requirements tasks and defects – categorized as Bugs and Defects.

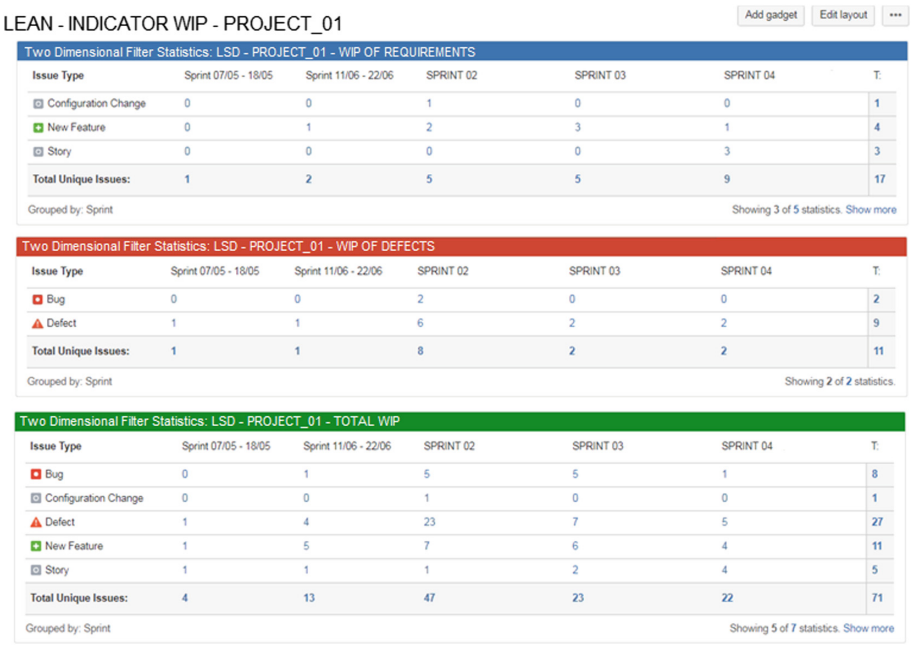

Fig. 3. LEAN Control Panel – INDICATOR (WIP) – PROJECT_01.

Table 5. Result of the percentage of WIP deviations per type of task and *Sprint*.

Sprint	Issue type	
	Requirements	Defects
SPRINT 02 – 07/20/2018	11%	17%
SPRINT 03 – 08/17/2018	22%	9%
SPRINT 04 – 09/14/2018	41%	9%

4.3 Number of Bottlenecks (NB)

It was not possible to add the NB indicator to a control panel, as performed with the other indicators, since JIRA has a set of Sprints follow-up reports and, among the reports available, there is the CFD – Control Flow Diagram.

The CFD allows to analyse the number of requirements in each column of Kanban value flow, based on the premises that the requirements to the left of the board are those under the "To Do" status, and those to the right are the ones under the "Resolved" or "Closed" status. The CFD meets the NB calculation specification by demonstrating the occurrence of work overload in a specific phase of development interaction.

In the CFD description [14] it is expected that the number of requirements in each phase is graphically represented by a homogeneous surface, demonstrating the lack of bottlenecks and proper distribution of requirements in each phase.

Figure 4 shows that among the three Sprints there is a bottleneck when starting the "to do" requirements specification, implementation, tests and deployment activities, highlighted in orange in the graphics and SPRINT 04.

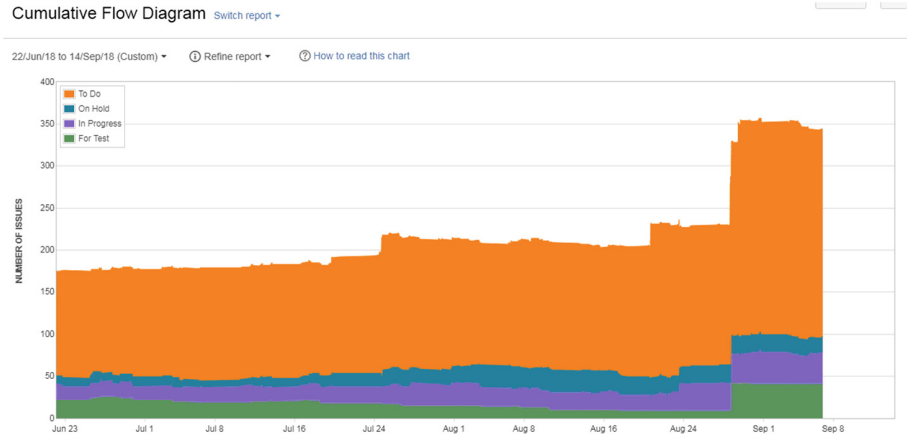

Fig. 4. Persistent filters from a JQL query.

Finally, a possible hypothesis for the bottlenecks previously mentioned is the distribution of PROJECT_01 team collaborators into 03 requirement analysts and 04 developers. It seems that this distribution is insufficient to fulfil a continuous flow of requirements over time.

5 Conclusion

The first three phases of the method have produced items that associate JIRA tool to waste indicators. These items are requirements in natural language (RNL); Tables of Semantic Relation (TSR) and; structured JQL query (SQSJQL) specifications. All these three items can be adapted to be applied to other projects.

The fourth phase of the method, which is the elaboration of reports and graphics per indicator, proved to be efficient when showing waste data in JIRA with a total effort time shorter than four week-period defined in the Sprints analysed, representing the most significant contribution of this study. The proposed method was able to identify in NAI and NB indicators the occurrence of waste in the form of delays, while the WIP indicator identified waste in the form of defects, extra features and unfinished work.

Additionally, only after performing qualitative analysis based on the indicators identified was it possible to accurately define the causes for the types of waste observed, through the identification of scenarios that directly affect the value flow transitions and the volume of requirements in stock or in each phase.

Further studies may, by adapting the semantic relation model, be extended to other agile planning tools (Monday.com, Wrike, Pipedrive, Glip etc.) or to other indicators,

making it possible to identify the impact of waste reduction when effort is estimated in story points or in hours to delivery of a requirement. These studies can implement the 4th phase of the proposed method with the use of BI tools such as Alteryx Desktop, Tableau, Microsoft Power BI.

Acknowledgement. The second author gratefully acknowledges the financial support from grants #2019/01664-6 and #2017/50343-2, São Paulo Research Foundation (FAPESP).

References

1. Poppendieck, M.P.T.: Leading Lean Software Development: Results Are not the Point, 1st edn. Pearson, Upper Saddle River (2009)
2. Ikonen, M., et al.: Exploring the sources of waste in Kanban software development projects. In: Proceedings - 36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2010, pp. 376–381 (2010)
3. Poppendieck, M.P.T.: Lean Software Development: An Agile Toolkit. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
4. Petersen, K., Wohlin, C.: Measuring the flow in lean software development BT - focus on agile software development. Softw. Pract. Exp. **41**(9), 975–996 (2011)
5. Behroozi, N., Kamandi, A.: Waste elimination of agile methodologies in web engineering. In: 2016 2nd International Conference on Web Research, ICWR 2016, pp. 102–107 (2016)
6. Nord, R.L., Ozkaya, I., Sangwan, R.S.: Making architecture visible to improve flow management in lean software development. IEEE Softw. **29**(5), 33–39 (2012)
7. Jonsson, H., Larsson, S., Punnekkat, S.: Synthesizing a comprehensive framework for lean software development. In: Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013, pp. 1–8 (2013)
8. Murphy, T.E., West, M., Mann, K.J.: Gartner https://www.gartner.com/doc/reprints?id=1-3YWBN1Q&ct=170427&st=sb%255Bgartner.com. Accessed 11 Nov 2017
9. Poppendieck e Poppendieck (2011)
10. Bell, S.C., Orzen, M.A.: Lean IT: Enabling and Sustaining Your Lean Transformation. CRC Press, Boca Raton (2016)
11. Sedano, T., Ralph, P., Peraire, C.: Software development waste. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp. 130–140 (2017)
12. Petersen, K., Wohlin, C.: Software process improvement through the lean measurement (SPI-LEAM) method. J. Syst. Softw. **83**(7), 1275–1287 (2010)
13. Pressman, Roger S.: Software Engineering - A professional Approach, pp. 77–79. McGraw-Hill Science, New York (2016)
14. Atlassian: JIRA Software. https://br.atlassian.com/software/jira. Accessed 6 May 2018