Impediments to Flow: Rethinking the Lean Concept of 'Waste' in Modern Software Development

Ken Power and Kieran Conboy

[1] Cisco Systems, Galway, Ireland
ken.power@gmail.com
[2] National University of Ireland, Galway, Ireland
kieran.conboy@nuigalway.ie

Abstract. Eliminating waste is a core principle of lean thinking. Despite the emergence of literature that applies lean in the software domain, an underlying analysis of this literature reveals the fundamental interpretation of waste has remained largely unchanged since its origins in manufacturing. Lean defines waste as any activity that does not directly add value as perceived by the customer. Software development is a creative design activity, not a production activity, and agile teams and organizations are more akin to complex adaptive self-organizing systems than repetitive production lines. Waste has different meaning in such systems. This paper reframes the lean concept of waste as impediments to flow in complex human systems. Drawing from ongoing research, this paper presents an updated categorization to describe the impediments faced by teams and organizations. The categories are extra features, delays, handovers, failure demand, work in progress, context switching, unnecessary motion, extra processes, and unmet human potential. These categories provide a foundation for helping teams and organizations to see, measure and reduce impediments to flow in their systems.

Keywords: agile, lean, waste, impediment, flow, value, complexity, human systems dynamics, extra features, delays, handovers, failure demand, work in progress, context switching, unnecessary motion, extra processes, unmet human potential.

1 Introduction

The first step in creating a lean organization is learning to see and manage waste [1, 2]. Lean defines waste as any activity that does not directly add value as perceived by the customer [1]. However, the waste metaphor does not translate comfortably from its origins in automobile manufacturing to modern knowledge work [3]. End-to-end flow of work through the system is still a valuable goal for teams and organizations, yet smooth flow remains difficult or unachievable for many. Teams and organizations attempting to achieve flow face many impediments. Removing impediments to flow is critical to improving a team's or organization's process [4]. The translations of the lean concept of waste in the agile literature to date have

G. Cantone and M. Marchesi (Eds.): XP 2014, LNBIP 179, pp. 203–217, 2014.

focused on an almost literal translation of the wastes of manufacturing production. These translations are inconsistent and lack a coherent presentation in the context of modern knowledge work, including software development. This paper proposes that a more appropriate perspective on the lean concept of waste for the complexity of 21st century teams and organizations of knowledge workers is to reframe waste as impediments to flow. Its not that we simply use the terms interchangeably; they are different but related concepts. Waste still exists in software development. However, this paper argues that a focus on impediments to flow is more appropriate. There are cases where waste leads to impediments, and impediments lead to waste. There are cases where the lens of impediments is a more useful perspective than the lens of waste.

2 Background

The original waste categories were created in the 1940s to address problems and promote a focus on cost reduction in the automobile-manufacturing domain. There are several definitions of waste in the literature. Ohno originally identified seven categories of waste in business and manufacturing processes [2]. Ohno originally described seven categories of waste in manufacturing, explaining the number seven comes from an old Japanese expression "*He without bad habits has seven*", which Ohno used to reinforce the point that "*even if you think there's no waste you will find at least seven types*." [5]. Liker added an eighth waste to give what have become known as the eight wastes of the Toyota Production System [6]. Definitions of waste vary, and include defining waste as those elements of production that increase cost without adding value [2], or activities that do not contribute to operations [7]. The lean production literature defines waste as "*any human activity that consumes resources but creates no value*" [8]. Definitions of waste in software development have largely just reused the TPS and lean production definitions, emphasizing waste as anything that does not add value from the perspective of the customer [1], or that consume time and effort, therefore creating costs, without adding value [9] [10] [11] [4]. Lean Startup simply restates the TPS definition [12] [13]. Other bodies of work in manufacturing and product development use between seven and ten categories of waste [14].

Among the few authors who have written about the dissonance that comes from applying the concept of waste from the manufacturing domain to modern product development are Reinertsen [15], Shalloway [4] and Anderson [3, 16]. Anderson refers to "wasteful" activities in economic terms as costs, and describes three types of cost [3]. Transaction costs are the setup and teardown costs incurred by software projects. Coordination costs are any activities that involve communication and scheduling. The third cost, Failure load, is what Anderson defines as demand generated by customers "*that might have been avoided through higher quality delivered earlier*" [3].

Although there is much the software industry can learn from the manufacturing domain, agile teams and organizations are better understood as complex adaptive systems (CAS) that are self-organizing and have emergent properties. Dooley notes

that "*the prevailing paradigm of a given era's management theories has historically mimicked the prevailing paradigm of that era's scientific theories*" [17]. The complexity sciences have emerged as one of the prevailing paradigms for modern management thinking in general [18, 19], and agile management in particular [20]. Stacey has shown that "*all organisations are complex adaptive systems in which groups and individuals are the agents*" [21]. Waste has different meanings in such systems. Acknowledging nature of modern software development, the Scrum framework is specifically designed to deal with complex adaptive problems. Sutherland and Schwaber write that Scrum is a framework "*within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value*" [22].

3 Impediments to Flow

This research has found that discussing '*waste*' is an emotive topic in teams and organizations. It is not easy for people to see that the activities they are engaged in, which can vary from wasteful tasks to the core of their job description, are actually waste from a holistic systems perspective. Anderson writes that "*a focus on flow, rather than a focus on waste elimination, is proving a better catalyst for continuous improvement within knowledge work activities such as software development*" [16].

It has proven relatively easier to talk to people, teams and organizations about what slows them down, what impacts the flow of work through their organization. This research has also found that coming from the perspective of impediments reveals much more about what is happening within the system. By taking a purely waste-focused perspective, people tend to focus on efficiencies and costs. By taking an impediment-focused perspective, people tend to focus more on effectiveness and optimizing the flow of value.

Frameworks such as Scrum place an explicit focus on removing impediments, though without defining what impediments are, or providing guidance on learning to see, understand or manage impediments [22]. This research has found that people have difficulty understanding what an impediment is, how to see them, how to measure and quantify their impact, and how to reduce them. Using the definition and categories presented in this can paper help teams and organizations to see impediments, and give them a foundation for understanding, measuring and reducing the impediments so that work flows more smoothly through their system. While this research addresses all these areas, the scope of this paper is to provide a foundation for defining impediments and present a set of impediment categories that are used to develop the habit of spotting patterns in human systems.

Part of the challenge relates to the balance between efficiency and effectiveness. As DeMarco notes "*you're efficient when you do something with minimum waste. And you're effective when you're doing the right something*" [23]. A focus purely on waste leads to a focus on efficiency, possibly at the expense of effectiveness. A focus on impediments, on the other hand, balances the discussion with an emphasis on effectiveness. Research by Wang has shown that "*agility requires waste to be*

eliminated but only to the extent where its [the organization's] *ability to respond to change is not hindered. This does not remove the need to be economical, only lowers its priority*" [24]. Removing waste is a valuable goal in a production process, and a useful metaphor for the parts of software development that are a production activity. Removing impediments is a more useful metaphor for the creative work that is the design activity (including architecture, design, coding, testing) of software development.

Another example of where the perspective of impediments is more useful than that of waste comes into play when considering variability. TPS emphasizes removing variability from the manufacturing process through eliminating waste [5]. Variability in product development, on the other hand, is something to be embraced [15]. In agile software development, a perspective that emphasizes removing impediments to innovation is more useful than one that seeks to eliminate variability.

3.1 Definitions of Value, Flow and Impediment

The Merriam-Webster dictionary defines value as "*usefulness or importance*" [25]. According to Liker, what defines value is the answer to the question "*What does the customer want from this process?*" [6]. In other words, is what the team or organization doing delivering value for the customer? Value Stream Maps are one technique for visualizing the process that delivers customer value. According to Beck, XP team members do only what is needed to create value for the customer [26]. Scrum is designed to reveal the efficacy of the product management and development practices used to deliver value so that teams and organizations can improve [22]. Value has more than direct financial connotations. For the purposes of this research, ***value*** *is anything the customer wants, and any activity that is useful or important in the context of providing value to customers.*

Beck defines flow as one of the core principles of XP: "*Flow in software development is delivering a steady flow of valuable software by engaging in all the activities of development simultaneously*" [26]. When creating XP, Beck chose practices that are "biased towards a continuous flow of activities rather than discrete phases".

The Merriam-Webster dictionary defines an impediment as "*something that makes it difficult to do or complete something; something that interferes with movement or progress*" [27]. Synonyms include obstacle, hindrance, obstruction, interference and encumbrance. From a CAS perspective, an impediment is anything that inhibits the system from achieving its purpose or goal. From a lean perspective, one purpose of an organization is to deliver value to its customers, and balance the needs of its wider community of stakeholders. Combining these two perspectives gives this definition:

> A*n* ***impediment*** *is anything that obstructs the smooth flow of work through the system and/or interferes with the system achieving its goals.*

So, determining if something is an impediment, can be based on the answer to two questions; (1) is this thing obstructing or preventing the work from flowing smoothly through the system? (2) Is this thing preventing the system from achieving its goals? If the answer is 'yes' to either or both of these questions, it is an impediment to flow.

There is a relationship between wastes and impediments. From the earlier definition of waste, it can be seen that the definition of impediments includes waste, but broadens the perspective. In other words, *a waste is an impediment* if it obstructs the smooth flow of work through the system, or interferes with the fitness of the system. A *waste causes an impediment* if it results in something that obstructs the smooth flow of work through the system, or interferes with the fitness of the system. This multi-dimensional perspective gives us a more reasoned way to assess waste in the context of impediments.

4 Impediments in Complex Adaptive Human Systems

Wang and Conboy express a concern about whether CAS is appropriate to the study of human organizations, given its origins in the natural sciences and suggest "*a combination of CAS theory with appropriate social theories might be a promising avenue*" [28]. Recognizing that concern, this research uses a particular field of CAS study called Human Systems Dynamics, or HSD [29]. Self-organization is widely acknowledged as a key property of successful agile teams [30]. HSD provides a model for understanding self-organization in human systems.

HSD defines a CAS as a "*collection of individual agents who have the freedom to act in unpredictable ways, and whose actions are interconnected such that they produce system-wide patterns*" [31, 32]. HSD uses three core elements to describe systems: containers, differences and exchanges (CDE). Containers are boundaries within which self-organization of human systems occurs. This is accomplished through focusing and constraining the interactions among the agents in the system. Examples include teams and organizations. Differences establish the potential for change in a human system, creating the possibility for the system to self-organize to a new state. Exchanges, also known as *Transforming Exchanges*, are interactions between the agents (people, teams, etc.) in a Container, and are "*a necessary condition for self-organizing processes to occur*" [29].

In software development, and lean in general, flow is a system goal. Impediments to flow show up in the system-wide patterns that emerge as the agents interact to achieve flow. The diagram in Fig. 1 illustrates this. As the agents interact, patterns emerge in the system. Impediments influence the patterns that emerge, and create a tension in the system that in turn influences the behavior of the agents.

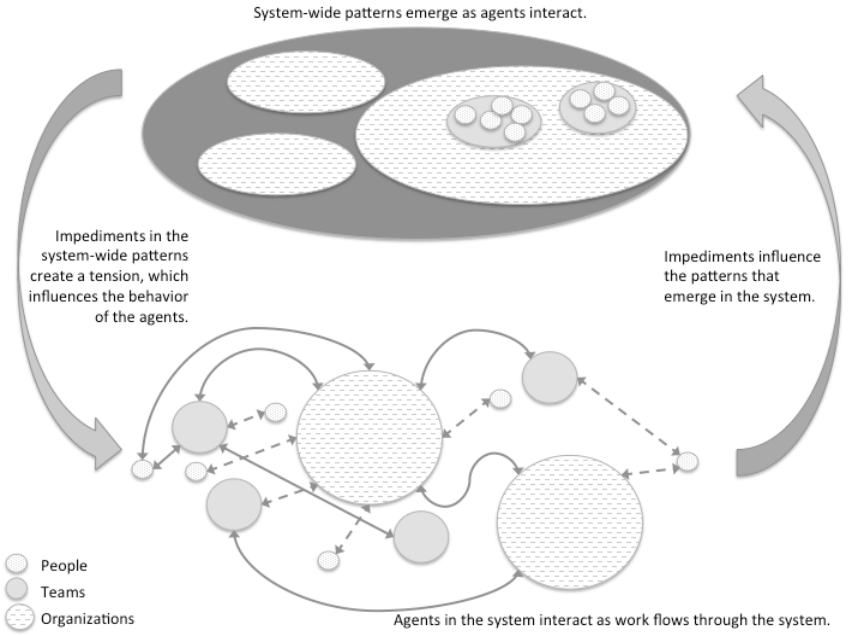

Fig. 1. Impediments influence the system-wide patterns in a CAS

5 Nine Categories of Impediments to Flow

This paper presents a framework of nine impediment categories that is built on the literature from manufacturing [2], lean production [33], lean thinking [8], lean software development [1, 11, 34, 35], product development flow [15], construction [36, 37] and healthcare [38, 39] and other sources [40].

The 9 categories of impediments to flow in software development, as identified in this research, are:

1. Extra Features
2. Delays
3. Handovers
4. Failure Demand
5. Work In Progress
6. Context Switching
7. Unnecessary Motion
8. Extra Processes
9. Unmet Human Potential

Each of these impediments is present in human systems. As with waste, having a set of categories helps reinforce the habit of seeing impediments in human systems [34]. In CAS terms, these impediments show up as patterns in human systems, as discussed in section 4 above. For the purposes of this research, and attempting to see and understand impediments, categories give a useful frame of reference and help

form the habit of seeing these patterns. The nine impediments are described in turn in the following sub-sections, and summarized in section 5.10 below. These are explored fully in a separate work [40].

5.1 Extra Features

Extra Features are those features that are added without either a proven need or valid hypothesis. Extra features impede the flow of valuable work through the system by consuming time and effort that could otherwise be spent on more value-adding work. They later prove to add no value for customers, or delay the delivery of more valuable features.

Liker refers to *overproduction* [6]. Beck notes software development "*is full of the waste of overproduction*", including "*elaborate architectures that are never used"* and *"documentation no one reads until it is irrelevant or misleading*" [26].

A Standish group report shows that approximately 45% of features in a typical system are never used, with 19% rarely used [1]. Extra features can be architecture features as well as business features.

Adding extra features significantly slows down feedback and revenue generation, as the product could be release sooner with fewer features. Many organizations do not consider the hidden economic costs of adding features that customers don't want, or for which there is not a sufficient demand. Beyond the initial costs to develop the feature, these hidden costs include:

- Time invested in maintaining the feature, possibly in multiple branches.
- Time invested in Failure Demand related to the feature, e.g., fixing defects, refactoring, or managing technical debt.
- The motivation of the team that developed the feature.

The opportunity costs associated with time lost on these other costs means the company could have been investing the time and money in something more valuable.

5.2 Delays

A delay is a situation in which something happens later than it should, and implies a holding back, usually by interference, from completion or arrival. [41]. Delays impede the flow of work through the system by adding to the overall lead time from request or idea to delivered product or service.

This impediment is also called *waiting* or *time on hand* [6]. The Poppendiecks note that "*one of the biggest wastes in software development is usually waiting for things to happen*" [1].

Delays can take many forms in teams and organizations. There is the delay that results from waiting for an activity to start or end. There is delayed learning. There is delay in information flow, resulting in the people who need the information to do their jobs do not get it in a timely fashion. This either causes them to wait, or to fill in the missing information with guesses.

Delays prevent the organization from delivering value to the customer as quickly as possible [1]. The ability of a team or organization to respond to an idea or request is directly related to the delays in the system. Reinertsen asserts that 85% of product development organizations do not understand the cost of delay associated with their projects or features [15]. He argues that understanding delay is so critical that, if organizations were to quantify just one thing, they should quantify the cost of delay, which he codifies as "*The Principle of Quantified Cost of Delay*". Brooks noted the "*severe financial, as well as psychological, repercussions*" of delays discovered late in a project [26]. Delays can have a cumulative effect. Brooks notes the secondary costs incurred by other projects waiting on the delayed project can far outweigh all other costs.

5.3 Handovers

Handovers occur whenever incomplete work must be handed over from one person or group to another. Handovers impede the flow of work through the system by adding delays, requiring more people, or losing knowledge as work is handed over from one person or group to another. Handovers are also referred to in the literature as hand-offs.

Ward argues that handovers are the most fundamental waste in companies because they separate knowledge, responsibility, action and feedback [42]. The Poppendiecks describe a case study from Ericsson that illustrates the cost of handovers [35]: "*handovers of information between functions tended to be inefficient; both knowledge and time were lost in every handover. As the number of handovers increased, the problems tended to escalate nonlinearly. Furthermore, workers in each function were assigned to multiple projects, causing severe multitasking that increased inefficiencies. The inefficiencies of handovers and multitasking showed up as decreased speed, and therefore slower time to market.*"

5.4 Failure Demand

Failure demand refers to the demand placed on systems (including teams and organizations) and is "*demand caused by a failure to do something or do something right for the customer*" [43]. It is the opposite of value demand, where the demand on systems is driven by value-adding work. [11] defines it as "*the demand on the resources of an organization caused by its own failures*". It impedes flow by consuming time and effort that could be spent on value-adding work.

Failure Demand includes what TPS calls rework [2], and is an example of *Type Two muda* [8]. Anderson calls it "*Failure Load*" [3].

Examples include defects, forced rework, technical debt, incomplete features, incorrect features, poor customer service, poor design, and poor or insufficient documentation. The Poppendiecks describe *relearning*, e.g., failing to remember what was learned at least once already [34]. Impediments occur when a support team places demands on a development team. Products that are difficult to integrate, deploy, or configure all create large amounts of failure demand. If the software "*gives operations and support organizations problems, both you and they are wasting valuable time*." [11].

Eliminating failure demand has a large economic benefit. In the financial services sector failure demand can vary from 20% to 45% of demand [43]. Seddon also shows that in police forces, telecommunications and local authorities failure demand can be as high as 50% to 80%. Removing failure demand can lead to enormous productivity improvements.

5.5 Work in Progress

Work in progress is analogous to inventory in software development. It is work that is not yet complete, and, therefore, does not yet provide any value to the business or the customer. Too much work in progress impedes the flow of work through the system by slowing down the flow of work for individual work items, and delaying the point at which value can be realized.

The Poppendieck's first book translated the TPS waste of "inventory" to "partially done work" [1]. It is also referred to as work in *process*, but that term is overloaded in software development.

Teams and organizations often get into trouble by having too much work in progress. Lots of WIP is often mistakenly taken as a measure of progress. Beck gives examples of waste resulting from excess work in progress, including "*requirements documents that rapidly grow obsolete*", and "*code that goes months without being integrated, tested, and executed in a production environment*" [44].

Example impacts include starting lots of projects or work items, but taking a long time to finish anything. Measuring progress in terms of perceived activity rather than delivered value.

5.6 Context Switching

Context switching occurs when people or teams divide their attention between more than one activity at a time [15]. Context switching impedes the flow of work through the system by adding to the overall lead time from request or idea to delivered product or service, and by causing failure demand and relearning. Context switching is sometimes necessary; the advice from this research is to use it consciously, deliberately and carefully. Unplanned context switching is generally worse than planned context switching, though poorly planned context switching is also harmful.

Context switching is often called task switching in the literature [1]. Much of the lean literature describes task switching as *working* on more than one thing at a time. However, as the wider research represented by this paper shows, context switching in knowledge work such as software development is caused by more than contending with multiple work items or tasks.

Example impediments include a developer working on more than one user story at a time. A tester working on more than one project at time. An engineer interrupted while working on a design. A team tasked with working towards delivering two or more projects at the same time. Meetings scheduled at times that guarantee interruptions.

Impediments occur when people switch their focus from one context to another. The cost includes more than lost time. Context switching is a root cause of some instances of estimation problems. Developers often do not predict unplanned context switches, deal with unplanned context switches effectively, or plan effectively for known context switches. This leads to work taking significantly longer than estimated.

In human systems the time that gets wasted is significant, but there are other costs associated with context switching. These include the opportunity cost associated with the interruption, as well as motivational costs. People have reported dissatisfaction with repeated context switching because it does not allow them to properly engage with the work, prevents them from contributing their best work, prevents them from developing mastery of their skills, and contributes to feelings of guilt because they feel they are letting down team members by not completing tasks or taking longer than they committed.

5.7 Unnecessary Motion

Unnecessary motion is any movement of people, work or knowledge that is avoidable, that impedes the smooth flow of work, or that creates additional inefficiencies. A classic example in software is the unnecessary motion caused by not having team members sitting together. Unnecessary motion impedes the flow of work through the system by adding overhead and causing delays in information or decision-making. TPS refers to the waste of unnecessary movement [6]. Authors translating from TPS to software translate *motion* to '*task switching*' ([1, 34] and [9]), but this research more accurately reflects task switching under the wider heading of *context switching* in 5.6 above.

5.8 Extra Processes

Extra processes generate extra work that consumes time and effort without adding value. Extra processes impede the flow of work through the system by adding additional steps, barriers, documentation, reviews, or other activities.

Extra processes is also referred to as *overprocessing* or *incorrect processing* [6]. In their first book the Poppendiecks translate "Extra Processing" to "Extra Processes" [1], and in a later book to "*Relearning*" – the waste often caused by long feedback loops [34]. A simple example is the relearning that developers must do to reacquaint themselves with code for a feature they worked on 6 months ago. For this research, "*relearning*" is more appropriately categorized as "Failure Demand". Another reason this research does not use the term "relearning" is that it can cause confusion. Learning is obviously a good thing in software development. Participants in focus groups have also expressed confusion about the terms. Hibbs et al translate overprocessing to "unneeded processes [9]".

Examples include paperwork and documentation that add no value. Pursuing a standard of quality that is higher than necessary. Time spent chasing an unreasonable level of certainty in estimating projects or features. Manual tasks that could be automated [9]. Forced conformance to centralized process checklists of "quality" tasks [10].

Extra processes have a demotivating impact on people who are forced to comply with non-value adding processes. Inefficiencies caused by poor tools or poor design can lead to defects (failure demand).

5.9 Unmet Human Potential

Unmet human potential is the waste of not using or fostering people's skills and abilities to their full potential. Unmet human potential impedes the flow of work through the system in many ways, though generally there is an opportunity cost through failing to reach the potential capability of the system. The flow of work, and the associated value created, is neither as effective nor efficient as it could be.

This is an expanded perspective on the waste of unused employee creativity [6]. This research categorizes this as "*unmet human potential*" because it goes beyond lack of engagement or not using employee creativity. The Poppendiecks describe the serious problem of not engaging people in the development process [34]. However, they describe this in the context of "*relearning*", which this research has framed more appropriately as *failure demand* (section 5.4). Not engaging people is more appropriately categorized under unmet human potential because it is a failure to take advantage of people's knowledge and creativity, removes ownership over their process, and removes opportunities for learning and improvement.

Deming wrote the "*greatest waste in America is failure to use the ability of people. Money and time spent for training will be ineffective unless inhibitors to good work are removed*" [45].

It impedes the potential of the individual, the team and the organization. Research into motivation has shown that engagement through sense of purpose, combined with the opportunity to develop one's skills and abilities, are vital ingredients in fostering intrinsic motivation [46].

5.10 Summary of the Nine Impediments to Flow

Table 1 summarizes the definitions of each of the impediment categories, and how they generally impede the flow of work through a system.

Table 1. Summary of Impediment Categories

Category	Definition	How it Impedes Flow
Extra Features	Extra Features are those features that are added without either a proven need or valid hypothesis.	Extra features impede the flow of valuable work through the system by consuming time and effort that could otherwise be spent on more value-adding work. They later prove to add no value for customers, or delay the delivery of more valuable features.
Delays	A delay is a situation in which something happens later than it should, and implies a holding back, usually by interference, from completion or arrival.	Delays impede the flow of work through the system by adding to the overall lead time from request or idea to delivered product or service.
Handovers	Handovers occur whenever incomplete work must be handed over from one person or group to another.	Handovers impede the flow of work through the system by adding delays, requiring more people, or losing knowledge as work is handed over from one person or group to another.
Failure Demand	Failure demand refers to the demand placed on systems (including teams and organizations) and is "demand caused by a failure to do something or do something right for the customer"	It impedes flow by consuming time and effort that could be spent on value-adding work.
Work In Progress	Work in progress is analogous to inventory in software development. It is work that is not yet complete, and, therefore, does not yet provide any value to the business or the customer.	Too much work in progress impedes the flow of work through the system by slowing down the flow of work for individual work items, and delaying the point at which value can be realized.
Context Switching	Context switching occurs when people or teams divide their attention between more than one activity at a time	Context switching impedes the flow of work through the system by adding to the overall lead time from request or idea to delivered product or service, and by causing failure demand and relearning.
Unnecessary Motion	Unnecessary motion is any movement of people, work or knowledge that is avoidable, that impedes the smooth flow of work, or that creates additional inefficiencies	Unnecessary motion impedes the flow of work through the system by adding overhead and causing delays in information or decision-making
Extra Processes	Extra processes generate extra work that consumes time and effort without adding value	Extra processes impede the flow of work through the system by adding additional or incorrect/unsuitable activities
Unmet Human Potential	Unmet human potential is the waste of not using or fostering people's skills and abilities to their full potential	Generally there is an opportunity cost through failing to reach the potential capability of the system. The flow of work, and the associated value, is neither as effective nor efficient as it could be.

6 Conclusions

This paper presented an updated framework for categorizing impediments in agile software development teams and organizations. The paper draws from ongoing research by the authors, and provided examples from both the research results and literature. The perspectives presented in this paper are compatible with, rather than competing with, other work that seeks alternative views to the traditional metaphor of waste from the manufacturing domain, in particular Anderson's cost perspective [3] and Reinertsen's economic framework [15].

The research presented in this paper is part of an ongoing program of research work by the authors. This paper provides the terminology and categories for impediments to flow. Other work by the authors provides an analysis of the causes of impediments, a detailed analysis of the impacts of the impediments, how to assess the impact, and how to reduce impediments.

Using a management paradigm grounded in the complexity sciences helps to better deal with the multi-dimensional nature of problems in software development. This research views organizations, teams, and the entire value stream as Complex Adaptive Systems, and uses Human Systems Dynamics (HSD) as a lens through which to better understand such systems. The HSD lens also helps us understand how to influence those systems to make improvements, such as removing impediments.

This paper provides the following:

- Reframe the lean concept of waste as impediments to flow.
- A set of nine categories of impediments to flow to help people see impediments.
- How each type of impediment impacts the flow of work.
- Examples of each type of impediment.
- Frame impediments to flow in the context of modern knowledge work, viewing teams and organizations as complex adaptive human systems.

Using the categories presented in this paper, researchers and practitioners can identify impediments to the flow of work in the patterns that emerge in the systems occupied by teams and organizations.

References

1. Poppendieck, M., Poppendieck, T.: Lean Software Development: An Agile Toolkit. Addison-Wesley, Bostan (2003)
2. Ohno, T.: Toyota production system: beyond large-scale production. Productivity Press, Cambridge (1988)
3. Anderson, D.J.: Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press, Sequim (2010)
4. Shalloway, A., Beaver, G., Trott, J.: Lean-agile software development: achieving enterprise agility. Addison-Wesley, Upper Saddle River (2010)
5. Ohno, T.: Taiichi Ohno's Workplace Management, Special 100th Birthday Edition. McGraw-Hill, New York (2013)

6. Liker, J.K.: The Toyota way: 14 management principles from the world's greatest manufacturer. McGraw-Hill, New York (2004)
7. Shingo, S.: A Study of the Toyota Production System. Productivity Press, New York (1989)
8. Womack, J.P., Jones, D.T.: Lean Thinking: Banish Waste and Create Wealth in Your Corporation. Simon and Schuster (2003)
9. Hibbs, C., Jewett, S., Sullivan, M.: The Art of Lean Software Development. O'Reilly Media, Inc. (2009)
10. Larman, C., Vodde, B.: Scaling lean & agile development: thinking and organizational tools for large-scale Scrum. Addison-Wesley, Boston (2009)
11. Poppendieck, M., Poppendieck, T.: Leading lean software development: results are not the point. Addison-Wesley, Upper Saddle River (2010)
12. Ries, E.: The Lean Startup: How Constant Innovation Creates Radically Successful Businesses. Penguin (2011)
13. Maurya, A.: Running Lean: Iterate from Plan A to a Plan That Works, 2nd edn. O'Reilly Media, Inc. (2012)
14. Pessôa, M.V.P., Seering, W., Rebentisch, E., Bauch, C.: Understanding the Waste Net: A Method for Waste Elimination Prioritization in Product Development. In: Global Perspective for Competitive Enterprise, Economy and Ecology, pp. 233–242 (2009)
15. Reinertsen, D.G.: The principles of product development flow: second generation lean product development. Celeritas, Redondo Beach (2009)
16. Anderson, D.J.: Lean Software Development. Lean Kanban University (LKU), Seattle (2013)
17. Dooley, K.J.: A Complex Adaptive Systems Model of Organization Change. Nonlinear Dynamics, Psychology and Life Sciences 1, 69–97 (1997)
18. Snowden, D.J., Boone, M.E.: A Leader's Framework for Decision Making. Harvard Business Review (2007)
19. Vasconcelos, F.C., Ramirez, R.: Complexity in business environments. Journal of Business Research 64, 236–241 (2011)
20. Appelo, J.: Management 3.0: leading Agile developers, developing Agile leaders. Addison-Wesley, Upper Saddle River (2011)
21. Stacey, R.: Emerging Strategies for a Chaotic Environment. Long Range Planning 29, 182–189 (1996)
22. Sutherland, J., Schwaber, K.: The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. Scrum.org (2013)
23. DeMarco, T.: Slack: getting past burnout, busywork, and the myth of total efficiency. Broadway Books, New York (2001)
24. Wang, X.: Organizing to be Adaptive: a Complex Adaptive Systems based Framework for Software Development Processes. School of Management, PhD. University of Bath (2007)
25. Merriam-Webster, http://www.merriam-webster.com/dictionary/value
26. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley, Boston (2005)
27. Merriam-Webster, http://www.merriam-webster.com/dictionary/impediment
28. Wang, X., Conboy, K.: Understanding Agility in Software Development from a Complex Adaptive Systems Perspective. In: 17th European Conference on Information Systems (ECIS), Verona, Italy (2009)
29. Eoyang, G.H.: Conditions for Self-Organizing in Human Systems. Doctor of Philosophy. The Union Institute and University (2001)

30. Cohn, M.: Succeeding with Agile: Software Development Using Scrum. Addison-Wesley, Upper Saddle River (2010)
31. Eoyang, G.H.: Human Systems Dynamics Professional Certification Training Manual. HSD Institute, Cohort 32 - Roffey Park, UK (2013)
32. Eoyang, G.H., Holladay, R.J.: Adaptive Action: Leveraging Uncertainty in Your Organization. Stanford University Press, Stanford (2013)
33. Womack, J.P., Jones, D.T., Roos, D.: The machine that changed the world: the story of lean production - Toyota's secret weapon in the global car wars that is revolutionizing world industry. Simon & Schuster, London (2007)
34. Poppendieck, M., Poppendieck, T.: Implementing lean software development: from concept to cash. Addison-Wesley, London (2007)
35. Poppendieck, M., Poppendieck, T.: The Lean Mindset: Ask the Right Questions. Addison-Wesley, Upper Saddle River (2013)
36. Mossman, A.: Creating value: a sufficient way to eliminate waste in lean design and lean production. Lean Construction Journal 2009, 13–23 (2009)
37. Sadreddini, A.: Time for the UK construction industry to become Lean. Proceedings of the Institution of Civil Engineers Civil Engineering Special Issue 165, 28–33 (2012)
38. Dickson, E.W., Anguelov, Z., Vetterick, D., Eller, A., Singh, S.: Use of Lean in the Emergency Department: A Case Series of 4 Hospitals. Annals of Emergency Medicine 54, 504–510 (2009)
39. Jimmerson, C.L.: A3 problem solving for healthcare: a practical method for eliminating waste. Healthcare Performance Press, New York (2007)
40. Power, K.: Impediments to Flow: Understanding the Lean Concept of 'Waste' in Self-Organizing Human Systems. PhD. National University of Ireland, Galway, Ireland (in Porgress)
41. Merriam-Webster, http://www.merriam-webster.com/dictionary/delay
42. Ward, A.C.: Lean Product and Process Development. The Lean Enterprise Institute Inc., Cambridge (2007)
43. Seddon, J.: Freedom from command & control: rethinking management for lean service. Productivity Press, New York (2005)
44. Beck, K., Andres, C.: Extreme programming explained: embrace change. Addison-Wesley, Boston (2005)
45. Deming, W.E.: Out of the Crisis. The MIT Press, Cambridge (1986)
46. Pink, D.H.: Drive: The Surprising Truth about what Motivates Us (2010)