The Identification and Classification of Impediments in Software Flow

Completed Research

Noel Carroll
Lero, National University of Ireland Galway, Galway, Ireland
Noel.Carroll@nuigalway.ie

Mairead O'Connor
Lero, National University of Ireland Galway, Galway, Ireland
M.OConnor67@nuigalway.ie

Henry Edison
Lero, National University of Ireland Galway, Galway, Ireland
Henry.Edison@nuigalway.ie

Abstract

Software impediments are anything that obstructs the smooth flow of work through a system or interferes with the system achieving its goals. However, it is unclear how impediments in software practices are identified, measured and managed across multiple practices and projects in teams that are often globally distributed, yet continues to consume vast amounts of organisational resources. Managers face growing pressure to introduce techniques to improve software productivity and reduce impediments with little guidance from best practice. As a first step to address this challenge, this article carries out a structured review on how impediments to software flow are identified and classified across the literature. The article describes the problem associated with managing software productivity and impediments. The article also presents the development of software flow impediments taxonomy and its contribution to the Information Systems literature. We adopt Design Science to guide our research efforts on the development of a taxonomy for impediments in software flow. From both a theoretical and practitioner perspective, this research provides a necessary foundation to support understanding regarding the nature and components of software impediments that are necessary for effective decision-making and management of modern software practices.

Keywords

Impediments, Software Flow, Software Productivity, Taxonomy, Design Science

Introduction

Within the Information Systems (IS) field, the importance of software development practices has long remained a central focus (Hassan and Mathiassen 2018). Across various software development practices, pressures on software productivity often come in the form of volatile markets (Ebert and Shankar 2017), frequent changing customer requirements (Maruping et al. 2009), relentless pressure from shorter time-to-market (Farid et al. 2017), and rapidly advancing technological and developmental change (Kautz et al. 2007). In response to such volatility, organisations grapple to alter their software development methods to ensure that they can better structure, plan, and control software development projects.

Studies have found that one size does not fit all in terms of using one software development method over another. Instead, each software team prefers the flexibility and authority to choose their method. This leads to the emergence of process independent performance metrics which has been shown to increase overall performance of software teams (Nidumolu and Subramani 2003). However, as the evolution of software development methods changes in response to organisational needs, there is an apparent mismatch in how changing software development methods and productivity are effectively managed within software organisations (Maruping et al. 2009).

Vast amounts of literature suggest that software development is a complex socio-technical activity that involves coordinating different disciplines and skill sets. This leads to ample opportunities for software impediments to emerge (Sedano et al. 2017). Power and Conboy (2014, p.4) define impediment as "*anything that obstructs the smooth flow of work through the system and/or interferes with the system achieving its goals*". Based on the definition, anything can be an impediment if it obstructs or prevents the work from flowing smoothly through a software project or if it prevents a software project from achieving its goal.

In software development and lean in general, flow is a system goal. Therefore, the impediments emerge in the system-wide patterns as the teams interact to achieve the flow across multiple teams and indeed locations to achieve software project objectives. This is evident within technology companies, whereby there are diverse, yet highly effective methods adopted across software teams (Vijayasarathy and Butler 2016). Individuals and teams usually 'customise' these methods further by tailoring for specific projects or by choosing pieces of methods in an 'a la carte' manner. However, despite the hype and purported benefits diverse software methods, organisations have been slow to adopt productivity assessment frameworks. Indeed, due to the lack of software productivity frameworks, it is unclear how impediments for software practices are identified, measured and managed across global organisations. This is a growing problem since the demand for cost savings in recent years has forced the software industry to look towards lean methods with the ambition of applying them to software production (Poppendieck and Poppendieck 2003; Middleton and Sutton 2005).

Research Focus

A fundamental and growing challenge in software flow is the classification of impediments and placing them into a taxonomy in order to make sense of the range of impediments which impact on software development practices. Addressing this problem is important since the demand for cost savings in recent years has forced the software industry to look towards lean methods with the ambition of applying them to "software production" (Poppendieck and Poppendieck 2003; Middleton and Sutton 2005). As a result, there have been growing efforts to adopt lean principles to define novel measures in software development practice (Ebert et al. 2012). Research indicates that we cannot expect to improve software productivity without measuring it (Anselmo and Ledgard 2003), which depends upon the understandability and independence of modules produced. Much emphasis has been placed on identifying methods to increase throughput and reduce lead-time to achieve high responsiveness to customers' needs, while also providing a tracking system that shows the progress/status of software product development (Power and Conboy 2014). This is important since it is "*an old wisdom from software development that most features in any product do not add value, but rather create unnecessary cost and complexity*" (Ebert et al. 2012, p.22). To address this, we set out to establish a Software Flow Impediments Taxonomy. Developing such a taxonomy, however, is a complex process that has not been adequately addressed in the IS literature (Nickerson et al. 2013). To address this challenge within a software flow context, this research sets out to develop a Software Flow Impediments Taxonomy. To guide the development of the Software Flow Impediments Taxonomy, we draw on a comprehensive literature review regarding software flow, impediments to flow, and taxonomy development. Therefore, the objectives of this research are (i) to identify software flow impediments in software organisation, (ii) to classify software flow impediments in software organisation, and (iii) to summarise the cause and impact of software flow impediments on software productivity.

Contemporary and Emergent Software Methodology

Software development teams are considered a core and often the differentiating asset, which exploits technology to connect, streamline and sometimes revolutionize all elements of organisational work (Colbert et al. 2016). However, industry is increasingly adapting software methods in parallel with the complex task of managing software teams and their productivity (Nidumolu and Subramani, 2003). For example, over the last two decades, agile has largely been regarded as the most popular software development method to counteract unpredictable conditions (Fitzgerald et al. 2006; Wang et al. 2012). The agile family is broad but over the last decade lean software development in particular has seen increasing uptake (Ebert et al. 2012).

Many of the emergent and contemporary software development methods build on each other's flaws. For example, agile solves many of the issues plan driven methods expose, and lean software development solves many of the issues agile expose (Ramasubbu et al. 2015). The increasingly popular lean software development focuses on reduction in waste and increased efficiencies which positively impacts quicker time to market (Poppendieck and Poppendieck 2003). An improvement in the flow of software development is the central and differentiating concept in lean software development (Womack and Jones 2003). Within literature, there have been recent empirical and conceptual contributions to software development flow thinking (O'Connor et al. 2017). An increase in software development flow has been shown to enhance visualisation and visibility among various software team performances. In fact, the Kanban, value stream mapping, and cumulative flow diagrams are increasingly popular flow focused tools within the lean software development family. Kanban has led to increases in (i) lead time; (ii) quality; (iii) communication and coordination; (iv) consistency of delivery; and (v) decreased customer reported defects (Ahmad et al. 2013). Value stream mapping has led to identification of value adding and non-value adding activities which streamlines the flow of software development (Ali et al. 2016). Cumulative flow diagrams have also led to increased visibility in the progress of complex project tasks (Petersen and Wohlin 2009). While there have been contributions to the flow of work in the form of new methods and tools, the problem of impediments to flow still exists (Power and Conboy 2014; O'Connor et al. 2017).

The literature suggests that problems emerge when organisations adopt multiple software development methods. This problem is exacerbated when it occurs at the project level (Fitzgerald et al. 2006; Wang 2011; Ramasubbu et al. 2015). Research regarding this phenomenon and its complexity needs to be addressed (Wang 2011). The use of multiple contemporary and emergent software development methods can take several different forms. One of the popular combination of methods is the alignment of lean software development with agile (Wang 2011). Multiple combinations are used to increase performance within projects. For example, the use of more than one software development method increases flexibility in adapting to user requirements and flexibility of dealing with context specific problems (Ramasubbu et al. 2015). Management increasingly utilize this process diverse view. However, this phenomenon becomes increasingly complex as managers seek to acquire performance metrics which are independent of the methods in which the teams use (Nidumolu and Subramani 2003).

Software Productivity: From Waste to Flow

The Lean concept of waste was originated from the manufacturing domain back in 1940. It set out to address opportunities to reduce the cost across the entire production of a vehicle. Lean thinking advocates any activity that does not directly add value to the finish product is considered as waste (Womack and Jones 2003). The agile world adopted this literal translation of waste and emphasised the value from the perspective of users (Poppendieck and Poppendieck 2003; Ries 2011). Another interpretation is provided by Anderson (2010) and refers to waste in economic terms, i.e. transaction costs, coordination costs and failure load.

Lean software development interprets the waste as partially done work, extra features, relearning, handoffs, delays, task switching and defects (Poppendieck and Poppendieck 2003). Different forms of waste can straightforwardly be detected in the manufacturing domain by observing the physical or tangible production flow. However, this is not the case in the software domain. Waste in software development is intangible, thus restraining the overall progress (Ikonen et al. 2010). For example, user stories that are created far in advance of coding will quickly become outdated. To solve this issue, extra processes are generated to repeat this process, which creates waste.

Reducing waste is critical to improve the overall team or organisational processes (Shalloway et al. 2010; Power and Conboy 2014). However, identifying waste remains a core challenge for managers – especially where intangible assets are core, such as within a software context (Sedano et al. 2017). Cognitive phenomena hinder people from noticing the waste in the current activities in which they are engaged (Sedano et al. 2017; Power and Conboy, 2014). Therefore, Anderson (2010) suggests focusing on flow rather than waste elimination to provide better catalyst for continuous improvement.

Flow is focused on managing a continuous value stream without any stoppage, scraps or backflows (Womack and Jones 2003; Anderson 2010; Reinertsen 2009). It is considered as a key concept for continuous software engineering (Fitzgerald and Stol 2017). Continuous software engineering advocates

to perform development activities as a continuous movement rather than a sequence of discrete activities. Unlike traditional project management, flow emphasises the need to manage queues rather than project phases and milestone and overcoming impediments to flow (O'Connor et al. 2017).

Software Impediments

There are some research efforts that examine the application of lean principles in software engineering, however there is little consensus as to how software impediments are identified and how software impediments are classified. The fundamental interpretation of waste in software productivity has remained largely unchanged. This is a problem as Power and Conboy (2014) contend that by focusing on waste, people focus on efficiency and cost. By focusing on impediments, people tend to focus more on effectiveness and optimising the flow of value. Therefore, they suggest shifting from waste of inefficiency to impediments to flow. Existing agile frameworks such as Scrum have suggested removing impediments. However, such frameworks neglect to define what constitutes as an impediment. Of consequence, there is little guidance in the agile/lean literature to identify or manage impediments (Power 2014).

Based on literature on manufacturing, lean and product development flow, Power and Conboy (2014) identify nine impediments of flow, which is summarised in Table 1. They argue that impediments obstruct the smooth flow of work through the system or factors that interferes with the fitness of the system. Yet, little is documented regarding the cause and impact of such impediments on software productivity – which makes it challenging to understand the wider context of software impediments. As a first step, we identify a need to develop a taxonomy to identify and classify impediments to software flow.

Impediment	Description
Extra features	Occur when features are developed without a proven need or valid hypothesis.
Delays	Imply a holding back from completion or arrival; something happens later than it should, e.g. waiting for an activity to start or end, etc.
Handovers	A situation when incomplete work must be handed over from one person or group to another.
Failure demand	Refers to demands caused by a failure to do something required by customers, e.g. defects, technical debts, etc.
Work in progress	Similar to inventory in software development; refers to works that are not yet complete and does not provide any value to customers, e.g. having too much work in progress.
Context switching	Happens when software development teams divide their attentions between a number of activities at any given time, e.g. meetings, working on more than one project.
Unnecessary motion	Refers to any movement of people, work, or knowledge that is avoidable, e.g. the motion caused by not having a co-located team.
Extra processes	Refers to over processing or unneeded processes, e.g. manual tasks that can be automated.
Unmet human potential	The waste of not using or fostering people's skills and abilities to their full potential.

Table 1 Impediments of Flow (Power and Conboy 2014)

Research Methodology

We carried out a structured literature to achieve the research objectives and guide the development of the data-gathering framework. This informed us on what should be considered for inclusion in the development of the Software Flow Impediments Taxonomy. We examined peer-reviewed journal and conference articles that examines impediments to software development flow. A search for relevant literature was conducted on the metadata, in particular on the title, abstract and keyword. We decided to use electronic databases that have good coverage, familiarity, reputation, advanced feature and exportability (Falagas et al. 2008). The electronic databases used to search for relevant literature review were Scopus and ISI Web of Science. ScienceDirect, Wiley Science, ACM, IEEExplore databases. These also return similar search results as Compendex or ISI Web of Science (Dybå et al. 2007). Hence, in this study, we used ISI Web of Science. We also selected Scopus since it is considered as the largest abstract database for peer-reviewed articles (Jabangwe et al. 2015).

The generic search string was: "(agile or lean) AND (waste* or impediment*) AND (categor* or classif* or taxonom* or typolog*) AND (software or system*)". We removed all articles that were not written in English. To ensure both software engineering and IS fields were considered, articles categorised under the following research areas were reviewed: computer science, business, management and accounting and information science. This search resulted in 44 articles from Scopus and 16 articles from ISI Web of

Science. To identify relevant studies, we read the title, abstract, and text to examine relevance to address the research objectives. We included an article only if it identifies and classifies the waste or impediments in a software or IT context. Thus, this resulted in the acceptance of six relevant articles as primary studies. The six primary studies are listed in Table 2 and form hereon referred using their IDs throughout the article. In addition to searching the literature, we also identified an opportunity to bring structure to the presentation of the findings through a taxonomy guided by Design Science. A taxonomy refers to a systematic approach to arrange and derive meaning through the science of classification according to a predetermined system, with the resulting catalogue used to provide a conceptual framework for discussion, analysis or information retrieval.

ID	Author(s)	Title	Venue	Year
P1	G. K. Kundu and B. M. Manohar	IT Support Service: Identification and Categorisation of Wastes	International Journal of Value Chain Management	2011
P2	K. Power and K. Conboy	Impediments to flow: rethinking the lean concept of 'waste' in modern software development	Proceedings of the 15th International Conference on Agile Software Development	2014
P3	O. Al-Baik and J. Miller	Waste identification and elimination in information technology organisations	Empirical Software Engineering Journal	2014
P4	F. Wijnhoven, D. Beckers and C. Amrit	Reducing Waste in Administrative Services with Lean Principles	Proceedings of the 37th International Conference on Information Systems	2016
P5	T. Sedano, P. Ralph, and C. Péraire	Software development waste	Proceedings of the 39th International Conference on Software Engineering	2017
P6	Y. Tiamaz and N. Souissi	Extended Classification of Waste in the Hospital System	Proceedings of the International Conference on Industrial Engineering and Operations Management	2017

Table 2 Studies Included

Developing a taxonomy played a key role in the research methods and presentation of research findings. Developing a taxonomy is often considered "*a complex process that has not been adequately addressed in the information systems (IS) literature*" (Nickerson et al 2013, p.1). Therefore, it is important to gain a clear insight on the purpose of a taxonomy before we can effectively design one. For example, within an IS context, Glass and Vessey (1995) describe how taxonomies provide a structure and an organisation to the knowledge of a field. Within such a taxonomy, the research community can examine the relationships among key concepts to hypothesize about these relationships. As researchers within IS, this encourages us to revisit some presumptions within the field and explore the research territory and core relationships (Iivari 2007) which bind our understanding of a specific domain and contribute to theory building (Doty and Glick 1994). We adopted a similar approach to Nickerson et al. (2013) and employed Design Science to create a Software Flow Impediments Taxonomy. Design Science research focuses on the development and performance of (designed) artefacts with the explicit intention of improving the functional performance of the artefact to derive new knowledge. The methods adopted to identify and categorize impediments of flow is also intended to support Design Science scholars developing taxonomies for similar or different IS domains. As part of this first phase of the research, we evaluate the taxonomy by assessing its efficacy in classifying impediments of software flow. By following Design Science principles, we developed an artefact as a method of enquiry, the purpose of which is to build or develop another artefact (i.e. a taxonomy). In the process of inductively developing the taxonomy, we were mindful that in the dynamic and volatile software domain, such taxonomies may be moving targets that evolve over time. In Design Science, it considers such a design approach as a search process, whereby "*the search for the best, or optimal, design is often intractable for realistic information systems problems*" (Hevner et al. 2004, p. 88).

Summary of Findings: Software Flow Impediments Taxonomy

The literature review findings support the identification of categories for impediments to software flow (i.e. addressing the first research objective). As part of the next step, the authors examined the classification of the impediments to software flow (i.e. addressing the second research objective). The term classification is used to refer to both the system and process of organising objects of interest and the organisation of the objects according to a system (Nickerson et al. 2013). In this case, we identified common categories related to impediments to software flow.

Category	Classification			Source(s)
	Impediment	**Cause**	**Impact**	
Software Development	Building wrong or extra features	• Poor requirement elicitation • Tension between user needs and business wants • Lack of customers involvement	• Affect team moral, team code ownership and customer satisfaction • Making inappropriate assumptions	P2, P3, P5
	Lack of equipment	• The equipment available cannot support the working capacity or no dedicated equipment for a desired task	• Delay in development flow	P6
	Information overflow	• Poor requirement elicitation	• Extra processes to define customers' needs	P3, P6
	Deferred verification and validation (V&V)	• Excessive standards to follow that encourage avoiding testing • Not enough training or awareness of, the importance of testing • Limited time to release	• Defective software • Rework	P3
	Defects	• Inaccurate or obsolete specifications	• Defective software • Rework	P1, P3
	Outdated information or working version	• Do not integrate frequently • Missing, inaccurate or incomplete documentation for complex system	• Rework	P3
	Extra processing	• Backlog inversion • Working too many features simultaneously	• Adding delays of key features • Lower team productivity • Duplicate work	P1, P3, P5
	Handovers	• Lack of time to complete a task	• Slowing down the flow • Knowledge loss	P1, P2
	Failure demand	• A support team places demands on a development team • Technical debts	• Waste time and resources	P1, P2, P5
	Context switching	• The absence of project prioritisation • People start multitasking	• Waiting	P2, P3, P5
	Work in progress	• Working on too many features simultaneously	• Slowing down the flow	P1, P2
Time	Delays	• Context switching • Waiting for approval or customer feedback	• Waste time and delay the project	P1, P2, P3, P4, P5
Software Team	Unnecessary motion	• Team churn • Knowledge silos	• Knowledge loss	P1, P2, P4
	Work-related stress	• Rush mode • Interpersonal or team conflict • Unnecessary context switching	• Absenteeism • Burnout • Lower productivity • A variety of health problems	P5
	Lack of staff	• Team member leaves a team or company	• Unable to complete the tasks assigned in the required time	P6
	Unmet human potential	• Poor training and communication.	• Lost people's motivation and creativity	P2, P4
	Ineffective communication	• Large team size • Asynchronous communication • Inefficient meeting	• Ineffective retrospective and sharing knowledge/perspective	P1, P3, P5
Software Project Management	*Poor planning*	• Poor management practice • Lack of training or knowledge transition	• Unmet human potential • Lost opportunity for carrying out more efficient process and outcome	P3
	Centralised decision-making	• Inappropriate organisational structure, policy or business model	• Adding delays to flow • Rework	P1, P6

Table 3 Software Flow Impediments Taxonomy

To address the third research objective, we identified the cause and impact of each impediment that also influenced the overarching labelling of impediment categories. In addition, we adopted the term classification to provide abstract groupings or categories to derive meaning on impediments of software flow. Table 3 provides a summary of these findings, followed by an in-depth discussion on their cause and

impact in terms of software productivity. While previous study identified 9 impediments (see Table 1), our literature review findings add 10 new types of impediments (listed on Table 3 in italics).

Category 1: Software Development

The findings reveal eleven impediments that are related to software development activities: building wrong or extra features, lack of equipment, information overflow, deferred V&V, defects, outdated information or working version, extra processing, handovers, failure demand, context switching, and work in progress. While an agile/lean approach advocates for close collaboration with customers, in reality they are not always reachable. The lack of customer involvement and poor requirement elicitation leads to the creation of inappropriate assumptions (P3). The use of personas may misrepresent the intended users. The requirements may also reflect business needs rather than actual users' needs. These make the developed features or product risky and wasteful activities. In such situations, teams may develop wrong or extra features or unnecessarily complex solutions. This affects team moral, team code ownership, and customer satisfaction (P2, P3, P5). In addition, the lack of equipment adds to the delays experienced in software development flow (P6). Defects are also considered impediments to software development flow (P1, P3). Inaccurate or obsolete user requirements require rework for some parts of the code or design. Ultimately, any rework wastes valuable time and resources across software teams. While agile/lean practices reduce the cost of fixing defects, there is emphasis on revealing and fixing the defect during its development phase (P3). Therefore, deferring V&V to later stages of development is also considered wasteful. Some of the key reasons to defer the V&V process are linked to teams' unwillingness to follow any standards written by others, lack of training on, or awareness of the importance of testing and limited time to release (P3). Out-dated information is another software development impediment which essentially becomes obsolete (P3). Outdated information and obsolete working versions are caused by infrequent integration, missing or inaccurate information, or incomplete documentation for complex system. This leads to significant levels of rework (P3). Extra processes are considered any additional processes that cause unnecessary, timely additional work that deters from adding value (P2). Extra processing, over specification, double handling, duplicate process and mismanaging the backlog are typically caused by backlog inversion and working too many features simultaneously. This adds delays to key features, lowers team productivity and increases duplication of work (P1, P3, P5). In addition, impediments relating to handovers refer to incomplete tasks by a team member and must be handed over to another team member (P2). Handovers are caused by a lack of time or knowledge to complete a task that typically leads to slowing down the flow and knowledge loss (P1, P2). Failure demand is caused by the demand and pressure placed on team members that are caused by incomplete work (P2). Failure demand, rework, and recurring incidents are typically caused by a support team by placing increased demands on a development team and technical debts. This leads to an increase in time impediments and misuse of resources (P1, P2, P5). Multitasking is the undertaking of multiple tasks at once (P5). Multitasking and context switching are caused by the absence of project prioritisation and individual multitasking. This also leads to an increase in waiting times (P2, P3, P5). Too much work in progress simultaneously leads to slowing down the flow of work (P1, P2).

Category 2: Time

The literature findings reveal that delay is a significant time impediment. Delays are a major and regular source of problems for software development flow. Delays in this instance describe the obstacles for completion of a task (P2). For example, delays and waiting time are caused by context switching (P1, P2) and waiting for approval or customer feedback (P3, P4, P5). This increases time wasted and subsequently delays projects.

Category 3: Software Teams

The findings also identify impediments in relation to software teams. Indeed, there are five key impediments in this category namely, unnecessary motion, work-related stress, lack of staff, unmet human potential, and ineffective communication. Studies on productivity place more emphasis on technical factors such as size and complexity while human-related factors have been largely disregarded (Wagner & Ruhe 2008). In fact, the scale and complexity of development activities is typically managed by software teams. The team characteristics (e.g. personalities, competences, skills, roles, teamwork etc.)

at one side may compromise their effectiveness and on the other side may bring benefits to maximize the productivity and improve quality (Licorish et al. 2009). Therefore, understanding and leveraging these characteristics can enable improved productivity within organizations, e.g. profiling software teams and projects.

Unnecessary motion often occurs when a knowledgeable team member leaves a team or company. In a project where tacit knowledge is a core resource, team churn suggests that management must actively invest in knowledge management strategies to avoid such impediments. Software teams need to examine code repository and backlog to understand the flow of a system (P4). In addition, work-related stress is caused by extraneous cognitive load and stress (P5). The pressures associated with development processes, e.g. releasing a feature at certain times, solving overcomplicated stories, waiting too long to resolve personal issues, etc., lead to inefficient work practices. Psychological issues can lead to other impediments as a result of absenteeism, burnout, lower productivity, and a variety of health problems (P5). Under-utilising people's skills, abilities, and competences to their full potential is also considered waste (P2). It affects people's motivation and creativity, which impedes the development flow to achieve the intended value. Communication also impacts on software flow. For example, ineffective communication includes the incorrect, inefficient or absence of communication that reduces team productivity (P5). This can occur when the software requirements or design is not documented properly, thus making it difficult to understand the document scope (P3). In a large distributed setting, communication issues continually arise in the retrospective meeting process. Imbalanced communication during the meeting hinders the knowledge and perspective sharing inside the team to reflect and respond to the issues.

Category 4: Software Project Management

The findings also indicate that poor planning and centralised decision making are the main impediments related to software project management. Poor planning refers to the inefficient use of resources, including human resources and tools/technology used during the development (P1, P6). This may surface as the result of poor project management practices, lack of training, or knowledge transition on specific tools/technology. This could lead to the unmet human potential to create value for customers and result in lost opportunity for carrying out more efficient activities. In a centralised decision-making model, any decisions that are made by lower-level roles are subject to the approval of more senior level role (P3). Such decision-making models could be the results of inappropriate organisation structure, policies and business model, thus making the decision-making process inefficient (P6). As a consequence, it takes much longer to implement decisions since they require more information at various organisational levels and create waste from waiting for specific outcomes. By the time the decision may be made, it may be obsolete and generate extra processes to rework the related tasks.

Discussion and Conclusion

This article describes the need to examine software impediments as they consume significant organisational resources. As a first step, this research carries out a structured review on how impediments to software flow are identified and classified. As a means to structure this process, we adopted Design Science to develop the Software Flow Impediments Taxonomy. Through the taxonomy, our research objectives are achieved by identifying and classifying software flow impediments in software organisation as well as summarising the cause and impact of flow impediments on software productivity. The taxonomy will shed further light on patterns in impediments of software productivity. We envisage that the model will also provide a rich exploratory platform to develop explanatory theories from industry case studies. This research contribution further supports managers to examine where process improvements may be achieved to determine project management success. It will also support managers' evaluations to better prepare IT project management practitioners across software project portfolios. This contribution differs to a typical study of project risk management because it focuses on software productivity (i.e. flow) rather than risk. However, combining software flow impediments and risk evaluation may form part of future avenues of research. Additional research efforts should focus on the potential mismatch in the methods used to assess business value and understanding process maturity (Carroll and Helfert 2015) within a software flow context. In addition, there are research opportunities in evaluating the taxonomy artefact to identify additional impediments to software flow in practice. By adopting, adapting, or

developing new insights on how one can monitor software impediments, there are growing opportunities to support organisations to continually embrace continuous improvement. As a first step, this article presents a review on the identification and classification of impediments to software flow. There are also future research opportunities to understand how digital transformations (e.g., changes in organisations' business processes, software practices, or management structures) are influenced, impacted, and modified by data analytics and process flow metrics across global organisations. We will also build on this article to examine how various categories on impediments impact on software productivity and organisational transformations and develop new recommendation strategies to overcome impediments to flow.

Acknowledgments

This research was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094.

References

Ahmad, M. O., Markkula, J., and Oivo, M. 2013. "Kanban in Software Development: A Systematic Literature Review," in *Proceedings of 39th Euromicro Conference on SEAA*, pp. 9-16.

Ali, N.B., Petersen, K., and Schneider, K., 2016. FLOW-assisted Value Stream Mapping in the Early Phases of Large-scale Software Development. *Journal of Systems and Software*, *111*, pp.213-227.

Anderson, D.J. 2010. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, Sequim, Washington.

Anselmo, D., and Ledgard, H. 2003. "Measuring Productivity in the Software Industry," *Communications of the ACM* (46:11), pp. 121-125.

Carroll, N., and Helfert, M. 2015. Service capabilities within open innovation: Revisiting the applicability of capability maturity models. Journal of Enterprise Information Management, 28(2), 275-303.

Colbert, A., Yee, N., and George, G. 2016. "The Digital Workforce and the Workplace of the Future," *Academy of Management Journal* (59:3), pp. 731-739.

Doty, D.H., and Glick, W.H. 1994. "Typologies as a Unique Form of Theory Building: Toward Improved Understanding and Modeling," *Academy of Management Review* (19:2), pp. 230-251.

Dybå, T., Dingsøyr, T., and Hanssen, G. K. 2007. "Applying Systematic Reviews to Diverse Study Types: An Experience Report," in *Proceedings of 1st International Symposium on Empirical Software Engineering and Measurement,* pp. 225–234.

Ebert, C., Abrahamsson, P., and Oza, N. 2012. "Lean Software Development," *IEEE Software* (29:5), pp. 22-25.

Ebert, C., and Shankar, K., 2017. Industry Trends 2017. *IEEE Software*, *34*(2), pp.112-116.

Falagas, M. E., Pitsouni, E. I., Maletzis, G.A., and Pappas, G. 2008. "Comparison of PubMed, Scopus, Web of Science, and Google Scholar: Strengths and Weaknesses," *FASEB Journal* (22:2), pp. 338–342.

Farid, A.B., Helmy, Y.M., and Bahloul, M.M., 2017. "Enhancing Lean Software Development by using DevOps Practices," *IJACSA* (8:7), pp. 267-277.

Fitzgerald, B., and Stol, K. J. 2017. "Continuous Software Engineering: A Roadmap and Agenda," *Journal of Systems and Software* (123), pp. 176-189.

Fitzgerald, B., Hartnett, G., and Conboy, K. 2006. "Customising Agile Methods to Software Practices at Intel Shannon," *European Journal of Information System* (15:2), pp. 200-213.

Glass, R.L., and Vessey, I. 1995. "Contemporary Application-domain Taxonomies," *IEEE Software* (12:4), pp. 63–76.

Hassan, N.R., and Mathiassen, L. 2018. "Distilling a Body of Knowledge for Information Systems Development," *Information Systems Journal* (28), pp. 175-226.

Hevner, A.R., March, S.T, Park, J., and Ram, S. 2004. "Design Science in Information Systems Research," *MIS Quarterly* (28:1), pp. 75–105.

Iivari, J. 2007. "A Paradigmatic Analysis of Information Systems as a Design Science," *Scandinavian Journal of Information Systems* (19:2), pp. 39–64.

Ikonen, M., Kettunen, P., Oza, N., and Abrahamssson, P. 2010. "Exploring the Sources of Waste in Kanban Software Development Projects," in *Proceedings of 36th Euromicro Conference on SEAA*.

Jabangwe, R., Börstler, J., Smite, D., and Wohlin, C. 2015. "Empirical Evidence on the Link between Object-Oriented Measures and External Quality Attributes: A Systematic Literature Review," *Empirical Software Engineering Journal* (20:3), pp. 640–693.

Kautz, K., Madsen, S., & Nørbjerg, J. 2007. "Persistent Problems and Practices in Information Systems Development," *Information Systems Journal* (17:3), pp. 217-239.

Licorish, S., Philpott, A., and Macdonell, S. G. 2009. "Supporting Agile Team Composition: A Prototype Tool for Identifying Personality (In)Compatibilities," in *Proceedings of the ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, pp. 66–73.

Maruping, L. M., Venkatesh, V., & Agarwal, R. 2009. A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research*, (20:3), 377-399.

Middleton, P., and Sutton, J. 2005. *Lean Software Strategies: Proven Techniques for Managers and Developers*. CRC Press.

Nickerson, R. C., Varshney, U., and Muntermann, J. 2013. "A Method for Taxonomy Development and its Application in Information Systems," *European Journal of Information Systems* (22:3), pp. 336-359.

Nidomolu, S. R., and Subramani, M. R. 2003. "The Matrix of Control: Combining Process and Structure Approaches to Managing Software Development," *Journal of Management Information Systems* (20: 3), pp. 159-196.

Nord, R. L., Ozkaya, I., and Sangwan, R. S. 2012. "Making Architecture Visible to Improve Flow Management in Lean Software Development," *IEEE Software* (29:5), pp. 33-39.

O'Connor, M., Dennehy, D., and Conboy, K. 2017. "Examining the Concept of Temporality in Information System Development Flow," In *International Conference on Information Systems (ICIS), Seoul, South Korea, 10-13 December*.

Petersen, K. and Wohlin, C., 2009. "A Comparison of Issues and Advantages in Agile and Incremental Development between State of the art and an Industrial Case," *Journal of Systems* and *Software* (*82:*9), pp. 1479-1490.

Poppendieck, M., and Poppendieck, T. 2003. Lean Software Development: An Agile Toolkit: An Agile Toolkit. Addison-Wesley.

Power, K. 2014. "Impediment Impact Diagram: Understanding the Impact of Impediments in Agile Teams and Organizations," in *Proceedings of Agile Conference*.

Power, K., and Conboy, K. 2014. "Impediments to Flow: Rethinking the Lean Concept of 'Waste' in Modern Software Development," in *Proceedings of Agile Conference,* pp. 203-217.

Ramasubbu, N., Bharadwaj, A., and Tayi, G. K. 2015. "Software Process Diversity: Conceptualization, Measurement, and Analysis of Impact on Project Performance," *MIS Quarterly* (39:4), pp. 787-807

Reinertsen, D. G. 2009. The Principles of Product Development Flow: Second Generation Lean Product Development, Celeritas Redondo Beach, Canada.

Ries, E. (2011). The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business USA.

Sarker, S., and Sarker, S. 2009. "Exploring Agility in Distributed Software Development Teams: An Interpretive Study in Offshoring Context," *Information Systems Research* (20:3), pp. 440-461.

Sedano, T., Ralph, P., and Péraire, C. 2017. "Software Development Waste," in *Proceedings of the 39th International Conference on Software Engineering (ICSE)*. Buenos Aires, Argentina, May 20-28.

Shalloway, A., Beaver, G., and Trott, J. 2010. *Lean-Agile Software Development: Achieving Enterprise Agility,* Addison-Wesley, Upper Saddle River, NJ.

Vijayasarathy, L. R., and Butler, C. W. 2016. "Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?" *IEEE Software* (33:5), pp. 86-94.

Wagner, S., and Ruhe, M. 2008. "A Systematic Review of Productivity Factors in Software Development," in *Proceedings of the 2nd International Workshop on Software Productivity Analysis and Cost Estimation*.

Wang, X., 2011. "The Combination of Agile and Lean in Software Development: An Experience Report Analysis," in *Proceedings of Agile Conference,* pp. 1-9.

Wang, X., Conboy, K., and Cawley, O. 2012. "Leagile" Software Development: An Experience Report Analysis of the Application of Lean Approaches in Agile Software Development," *Journal of Systems and Software* (85:6), pp. 1287-1299.

Womack, J.P., and Jones, D.T. 2003. Lean Thinking: Banish Waste and Create Wealth in Your Corporation, Simon and Schuster.