

Agile operational research

Melina Vidoni, Laura Cunico & Aldo Vecchietti

To cite this article: Melina Vidoni, Laura Cunico & Aldo Vecchietti (2020): Agile operational research, Journal of the Operational Research Society, DOI: [10.1080/01605682.2020.1718557](https://doi.org/10.1080/01605682.2020.1718557)

To link to this article: <https://doi.org/10.1080/01605682.2020.1718557>



Published online: 13 Feb 2020.



Submit your article to this journal [↗](#)



Article views: 25





View related articles [↗](#)



View Crossmark data [↗](#)

Agile operational research

Melina Vidoni^a , Laura Cunico^b and Aldo Vecchietti^b 

^aRMIT University – School of Science, Computer Science and Software Engineering, Melbourne, Australia; ^bInstitute of Design and Development (INGAR CONICET-UTN), National Scientific and Technical Research Council, Santa Fe, Argentina

ABSTRACT

As project management has become a critical subject in modern-world organisations, Operational Research (OR) needs to incorporate mechanisms to deal with rapid, unplanned changes as well as confusing information and stakeholders with conflicting values. Agile methods are widely used and tested in Software Engineering (SE) to deal with problems of the characteristics above. Because of this, after establishing that both OR interventions, as well as SE developments, have common stages and information evolution, this proposal aims to pose the challenge of applying agility to manage OR projects. Guidelines to adapt Agile Methodologies to OR are proposed, and a case vignette is studied as an initial test. Finally, future lines of work are considered to define how the larger project in which this proposal is embedded will continue.

ARTICLE HISTORY

Received 25 October 2018
Accepted 12 January 2020

KEYWORDS

Systems thinking; soft-OR;
project management;
practice of OR

1. Introduction

For several decades until now, Operational Research (OR) models are known as vital instruments in many organisations to make decisions in complex problems. As many authors pointed out, the development and implementation of such mathematical models face several difficulties: stakeholders with conflicting interest, poor definitions, confusing information, changing environments and problems with constant unclear ramifications (Ackoff, 1979; Churchman, 1967).

Researchers and practitioners proposed different methodologies to manage OR interventions, known as Soft-OR and Problem Structuring Methods (PSM), characterised by the inclusion of external stakeholders, as well as the use of techniques to improve requirement elicitation (Mingers & White, 2010). Some of them are Soft Systems Methodology (Checkland & Poulter, 2010), Strategic Options Development and Analysis (SODA) (Eden & Ackermann, 2001), and Strategic Choice Approach (Friend, 2006). This research area targeted an unresolved issue for OR interventions; then, due to the wide range of options of these approaches, different authors provided frameworks to compare them. For example, while Smith and Shaw (2019) tried to identify similarities in different methods with a framework structured around some traditional assumptions (ontological, epistemological, axiological, and methodological), Midgley et al. (2013) provided an approach for long-term comparisons, while remaining locally meaningful.

Nevertheless, these methodologies only focus on the initial stage of the process, without providing a global approach. Even more, their use in practice is still a topic under discussion: while it is widely accepted in some academic circles (Mingers, 2011), it is shunned in others as is not based on rigorous mathematical techniques (Ackermann et al., 2009).

Few articles focus on describing the use of Soft-OR or a PSM during an OR intervention. Da Silva Filho (2015) recommended PSM to an organisation to carry out their interventions after determining that their comprehension of wicked problems was skewed. Cabrera, Cabrera, Powers, Solin, and Kushner (2018) used a Soft-OR approach to showcase the impact of a given design in Community OR while discussing its consequences and lessons learned. Finally, Schramm and Schramm (2018) used a group-decision approach like SODA to support different decision-making processes in Brazilian watershed committees.

Nonetheless, it is more frequent to find articles describing the mathematical background and details of an intervention to the detriment of the application itself and the whole system project (Ackermann et al., 2009; Ormerod, 2014). OR still centres in mathematical models and algorithms, instead of its ability to formulate management problems, solve and implement them (Ackoff, 1979; Ormerod, 2014). As a result, it has continually focused on the mathematical representations to the situation addressed at the expense of systems thinking (Mingers & White, 2010). In many cases, this led to

Table 1. Agility values and principles.

	Code	Definition
Values	V1	Individuals and interactions over processes and tools.
	V2	Working software over comprehensive documentation.
	V3	Customer collaboration over contract negotiation.
	V4	Responding to change over following a plan.
Principles	P1	Satisfy the customer through the early and continuous delivery of valuable software.
	P2	Changing requirements, even in late phases, to enhance customer's competitive advantage.
	P3	Frequently delivery of working software, preferring shorter timescales.
	P4	Business people and developers must work together throughout the project.
	P5	Build projects around motivated individuals.
	P6	Conveying information on development teams through face-to-face conversation.
	P7	To use working software as the primary measure of progress.
	P8	Sustainable development: everyone should be able to maintain a constant pace indefinitely.
	P9	To have continuous attention to technical excellence and sound design.
	P10	Simplicity is essential.
	P11	Self-organising teams produce the best designs and architectures.
	P12	The team must regularly reflect on how to become more effective.

solving a situation that is widely different to reality (Ackermann, 2012).

In general, in an OR intervention, the team needed to implement a decision-making model consists of – at least – a mathematical modeller and a software engineer. They work in the whole process with ideas, goals, and definitions of other stakeholders such as end-users, supervisors, and managers. The generated mathematical model must be linked to the organisation's information system to get data and to provide results. From this perspective, an OR intervention is similar to the development and implementation of a software-intensive system. Its lifecycle starts with ideas, followed by design, execution, tuning, and maintenance. In this regard, they need to be managed as systems.

Software Engineering (SE) has several accepted, tested proposals for dealing with information systems. SE moved from focusing on how to develop software (Kneuper, 2017) to establishing process management methodologies under the name of lifecycles (Birrell & Ould, 1985; Boehm, 1986; Royce, 1970). However, the emergence of the Internet, the requirements for shorter time-to-market and the increase in changing requirements demonstrated the need for more lightweight methods. These were later grouped under the concept of agile methods (AM), founded in the Agile Manifesto (Beck et al., 2001). AMs are widely accepted in the SE community (Dingsøyr, Nerur, Balijepally, & Moe, 2012; Melo, Cruzes, Kon, & Conradi, 2011).

This article analyses the use of SE agility for OR interventions. For this purpose, the characteristics of the main stages of an OR intervention are identified together with the information managed at each stage. This evidences a match between OR interventions and agile lifecycles stages, showing that agility can be applied to project management in OR. This is further elaborated in three-step guidelines, which include selecting a methodology, organising a project, and representing the information in artefacts.

Finally, a case study vignette supports the viability of this approach.

This article is organised as follows. Section 2 discusses why agility is considered an option, while Section 3 proposes a solution to manage the information and divide the project into stages, creating the empirical context for using agility in OR. Section 4 provides practical guidelines to do so, discussed using a case vignette in Section 5. Finally, Sections 6 and 7 present discussions and conclusions.

2. Agile: Why?

AM is a type of project management process. They anticipate changes and allow a high degree of flexibility – in both project and source code – to rapidly adapt them. Because of the process and tools that AMs provide, stakeholders can make small objective changes without considerable amendments to the budget or schedule (Dingsøyr et al., 2012). This concept is also applied to manufacturing and supply chain management (Gligor, Esmark, & Holcomb, 2015).

In SE, the rise of AMs was powered through the generation of the Agile Manifesto (Beck et al., 2001): a document establishing the goals and philosophy for agility. It established four values and twelve principles that any method must have to be agile; these can be seen in Table 1. Even more, the Agile Manifesto's influence contributed to AMs' increasing acceptance (Kneuper, 2017; Tarhan & Yilmaz, 2014).

Many of these values and principles have a direct correlation to OR interventions. For example:

- **(V4) Responding to change over following a plan.** The emergence of the Internet made developments to face a shorter time-to-market and a need to better adapt to unclear and changing requirements (Boehm, 2006). Thus, agility states that a plan must: (a) be flexible enough to allow

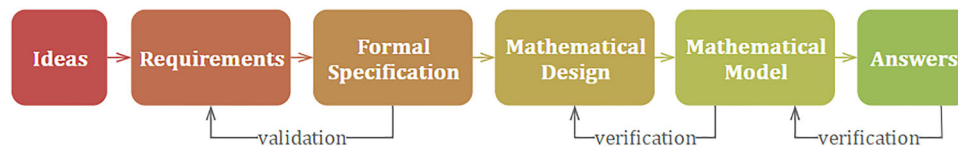


Figure 1. The process of information evolution during an OR project.

adapting the system to the required changes, and (b) provide artefacts and processes to do so. Usually, an OR model is not able to adapt to the stakeholders' needs, because its development is centred in technical, mathematical properties and does not consider future changes; then it loses its usability as it drifts away from the real, targeted situation to solve a more ideal, outdated case (Checkland & Poulter, 2010).

- **(P2) Changing requirements, even in late phases to enhance the customer's competitive advantage.** Related to value V4, requirements can also change in late stages of developments – i.e., when the model is reaching completion or is in-use in the organisation. Having a lifecycle that provides tools and management process to review and modify the source code when a change is detected, can reduce the rate of new errors. This may encourage the reuse of the mathematical code.
- **(P10) Simplicity is essential.** Complex, difficult-to-read source code or documentation is more prone to create misunderstandings; even more, it can increase the cost of modifying an existent model to the point where it is less expensive to start over than salvaging what already exists (Ahmed, Ahmad, Ehsan, Mirza, & Sarwar, 2010; Dingsøyr et al., 2012). Agility aims to simplify the code, even if it means rewriting it until it maximises its performances and readability. If the source code is easy to read, then it is possible to reuse it in new problems (Haefliger, von Krogh, & Spaeth, 2008), reducing developing times and costs, while at the same time refining solutions (Frakes & Kang, 2005). This property is crucial for OR models in changing environments.
- **(P12) Regularly reflect on how to become more effective.** Agility proposes that teams should be able to recognise what they did wrong in a project, to learn from their mistakes and apply that new knowledge in future interventions (Dingsøyr et al., 2012). This allows refining practices and process, to improve steadily.

Current PSM and Soft-OR methodologies imply capturing and visually representing the stakeholders' points of view (Mingers & White, 2010). Thus, PSMs focus on the most "social" aspects of agility, such as V1, V3, P2, P4, and P6 of Table 1. The remaining properties are left behind even when they

are a prominent part of an OR project's process (Ackermann, 2012).

Creating a project and source code that it is easily modified without negatively affecting its quality is not an easy task. This becomes even more challenging as the size and complexity of projects increase. AMs apply a technique known as *divide-and-conquer*, which implies fragmenting a set of requirements, selecting a few, and incrementally building the system by iteratively adding more requirements to it (Beck et al., 2001). This enables developers to frequently deliver working code, reducing the return-of-investment time, and testing the system-model "on-site" and interoperating with the others (Boehm, 1986). This is an intrinsically agile characteristic reflected in the properties P1, P3, P7, P8, and P10 (see Table 1).

3. Agile and or: Aspects in common

Managing OR interventions implies distinguishing the elements affecting the process, the emerging systems and people's rationales (Mingers & White, 2010); as a result, it requires to also focus on others activities beyond writing mathematical code. This section establishes the empirical context that will be used in Section 4, identifying the states that represent the evolution of information within an OR project and that defines its lifecycle.

3.1. Information evolution

In an OR intervention, information evolves and grows during a project, becoming more refined (Ormerod, 2008). Figure 1 summarises the evolution of the information states of an OR project lifecycle. This proposal showcases the similarities between OR and SE projects, aiming to facilitate the adaptation of AMs.

Although the process for OR is sequentially pictured, it is considered a *progressive elaboration*: continuously improving and adding details, as more specific, accurate information becomes available while the project progresses (Project Management Institute, 2017). Whereas it is possible to go back to several states earlier in order to add more detail, it is not tolerable to "skip" more refined states when moving forward. For example, it is feasible to go back from "Mathematical Model" to "Formal Specification," but after completing the changes, the

progression must improve the “Mathematical Design” before addressing the “Mathematical Model” once again.

Regarding each state, in particular, any project starts with “Ideas,” which defines that a given need should be satisfied, while “Requirements” formalise the conditions and capabilities limiting the solution to that need. Current PSM and Soft-OR methodologies aim to elicit these two states of information and agreements about them.

“Formal Specification” aims to structure “Requirements” to discover specific information, and prioritise functionalities of the model. This aims to shape the project process in incremental, iterative cycles, by following the three steps of Figure 2:

“Mathematical Design” is the primary step before coding the model. It consists of diagrams and documentation that communicate its structure to different stakeholders by using the Software Architecture concept of *points-of-view*: elements that document the same model from unique perspectives but with complementary specifications (ISO/IEC/IEEE, 2011).

Finally, the “Mathematical Model” is the source files with the model coded in the selected mathematical programming language, while the “Answers” are reports of results such as charts, files, spreadsheets, etc, derived from raw results, and also related analysis, and other internal process.

3.2. Lifecycle stages

The process of managing and evolving this information defines different periods of a project’s lifecycle. Each of these phases has a defined goal regarding the information to be used, and the refinement it

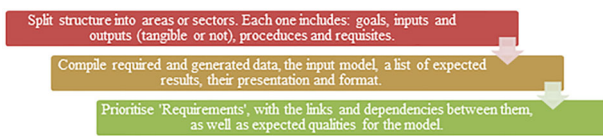


Figure 2. Steps to refine “Requirements” into a “Formal Specification.”

produces. The output generates the artefacts (see Section 4.3) used as input on the next stage, as the information evolves with the project.

Phases are related to short-term goals of the project and not to the situation it aims to solve. SE established that it is more productive to move in smaller steps: understanding the problem, designing and evaluating solutions, and then coding. This is done instead of attempting to perform all the steps at the same time – i.e., coding while requesting additional, improved data.

Figure 3 presents the proposed phases and corresponding information states, adding brief definitions on the phases. The use of stage names established in SE and nomenclature that is known to both practitioners and academics, simplify the adaptation of existing methodologies.

“Analysis” focuses on defining who is part of the project (clients and developers/modellers), what the project is about, and what the clients genuinely need. It is essential to understand the value of the project as a whole, integrate the participants’ knowledge and favour the synergy of collective work. PSM and Soft-OR methodologies can be applied during this phase (Checkland & Poulter, 2010; Eden & Ackermann, 2001; Friend, 2006).

“Design” aims to structure the requisites for formal modelling that does not require mathematical code. This is used as documentation, composed of the “Formal Specification” and the “Mathematical Design,” and is a base upon which accountable people can be defined, requisites prioritised, and how they will be translated to the model.

Any misunderstandings, lack of detail or missing agreements dragged on from the “Analysis” phase will negatively affect the “Design,” and cause defective refinements of information. In the “Design” phase, it is documented how decisions were made, how the code is organized, which parts of the code contain each functionality, and how changes should be handled and addressed. If possible modifications are considered at the “Design,” later changes in the

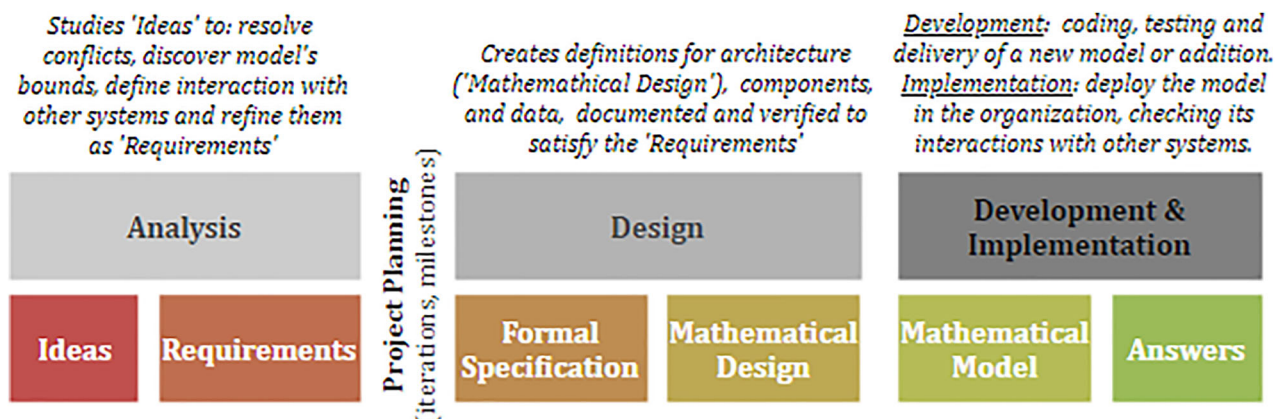


Figure 3. Proposed lifecycle stages for OR projects.

“Mathematical Model” become less “traumatic”; the cost of reworking the code and the chance of introducing new faults are lowered.

The following step is “Development,” and its goal is to code the model in a given mathematical language, following the specification created during the “Design” phase: it is the most traditional and core activity of any OR project. It also includes the generation of answers, testing regarding the inputs, and its validation, compared to the specifications of “Requirements,” and to the goals of the project.

The final stage is “Implementation.” Figure 3 depicts it together with the “Development” as it uses the same states of information. However, its goal is to deploy the model in the client’s organisation, to ensure its adequate interaction with existing systems, and to train users in its use. Often this stage does not receive the importance it should, especially in projects that generate models used to assist in a single decision. Putting a model into use is essential to its success.

4. Agile: How?

Making an OR intervention agile implies three steps, visible in Figure 4. The following subsections propose guidelines for each of them.

4.1. Selecting a method

Choosing the right method to arrange interventions allows the management sequence to be defined. Since AMs have been used in SE for almost two decades, many authors have compiled experiences to define specific recommendations using different

parameters; the authors Qumer and Henderson-Sellers (2008) provided an extensive report about it, according to different parameters such as team sizes, project length, geographical distribution, among others.

Though there are several papers along these lines in SE, they are not directly applicable to OR, as some parameters – such as project or team size – are not the same in both disciplines. Furthermore, offering such detailed selection guide requires having several real-world application reports specific to OR, as was the case in SE (Qumer & Henderson-Sellers, 2008). Therefore, until such data have been collected, this section only states an initial selection guideline, to kickstart its application in different interventions.

Overall, AMs can be grouped into two categories:

- Code-focused methods* are especially suited for smaller teams and projects. Though these AMs provide a specific reduced project management frame, most of their practices, activities, and processes are concentrated in coding. Examples are XP and Crystal.
- Project-focused methods* offer practices, artefacts, and activities linked with the global vision of the project. They are usually targeted to mid/large project or team sizes, or situations in which an overall organization is mandatory. As a result, they can often be merged with other AMs to obtain a more thorough methodology. Examples are Scrum, Kanban, and Lean.

This categorisation narrows the search, by evaluating *internal* and *external characteristics*. The first set has perks related to the team itself, its training, background, and organisation, while the second group involves restrictions coming from the contracting organisation. Figure 5 summarises this process.



Figure 4. Steps to apply agility to OR interventions.

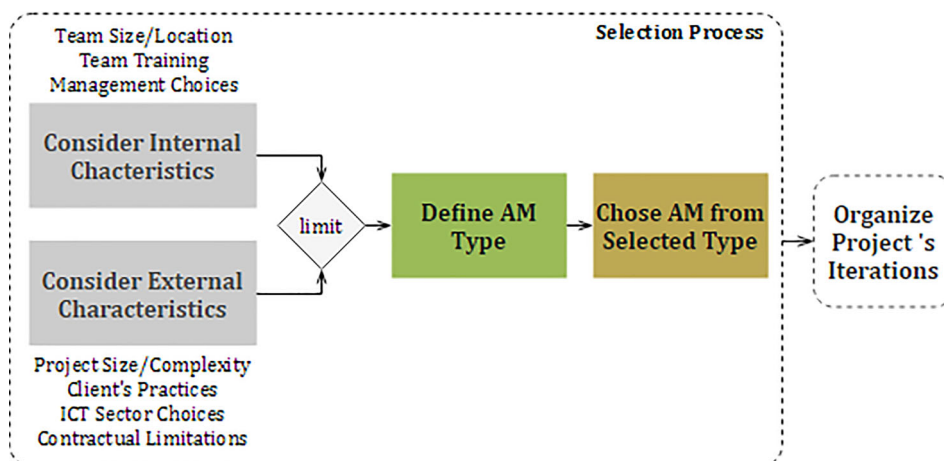


Figure 5. Overview of the AM selection process.

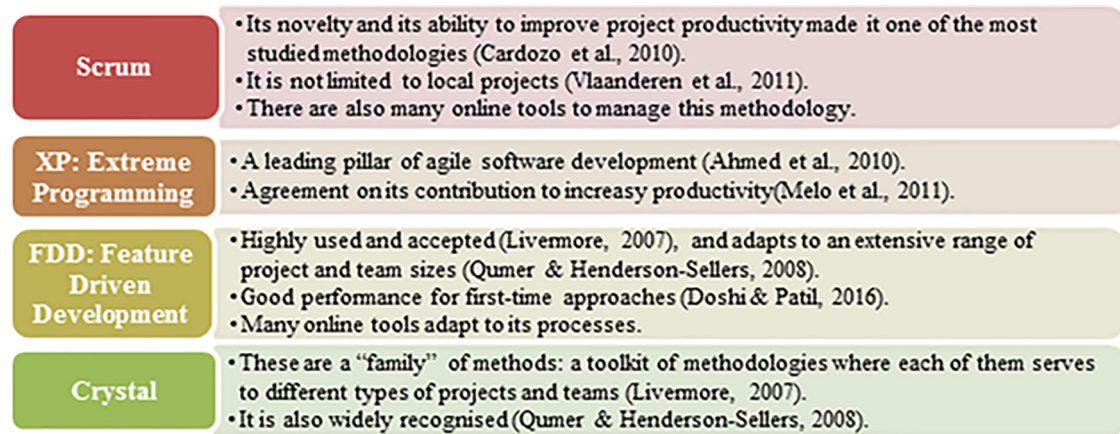


Figure 6. Selected Agile Methodologies, and reasons for selection.

Table 2. Comparison of the proposed OR project stages, and AMs processes.

	Scrum	FDD	XP	Crystal
Analysis	Initiate: Project vision, participants, create a backlog, initial release planning. Plan & Estimate: Create and approve user stories, tasks, and sprint backlog.	Develop an Overall Model: Initial problem model. Build a Feature List: Features grouped in sets & subject areas. Plan by Feature: Development plan, class owners, feature set owners.	Listening: Get feedback and client involvement. Designing: Simple design, class owners, coding style, and so on. Defines the iteration's user stories to be developed.	Project: Build a core team, explore requirements, build an initial plan, and shape the methodology. Delivery: Check the release plan, organise the iterations, and work on their features.
Design	Implement: Create deliverables (code and design), daily stand-up, groom the prioritised backlog.	Design by Feature: Incrementally detailed modelling of the system.		Iteration: Improve requirements and overall framework. Define the features.
Development	Review & Retrospect: Reviews deliverables, and performance. Define how to improve.	Build by Feature: Incremental and iterative coding and testing. It is a completed client-valued function.	Coding: Prioritize working code. Testing: Integrated with coding. Reduces bugs and confirms the client's approval.	Development: code a feature, and queue it for integration. Integration: unify code, and run automated unit testing.
Implementation	Release: Handle the accepted deliverables to the customer. Define the lessons learned.			Delivery: Handle the accepted code to users. Reflect on the lessons learned.

Regarding *internal characteristics*, smaller teams with lower-to-none experience in applying agility to OR should use Type A methods; thus the change in their practices is less significant, while at the same time includes elements of project management. Otherwise, larger and distributed teams should move towards Type B, as these AMs imply a more radical change, being harder to implement if the modellers are not trained or present a higher resistance to adjusting their practices. Concerning *external characteristics*, longer projects are more dependent of managerial activities; other elements to consider the provision of documentation as part of the agreement and the selection of an AM already used at the client's ICT department, among others.

As can be seen in Figure 5, internal and external characteristics are not excluding, and need to be considered at the same time. Which one weighs more towards a final decision ultimately depends on the specific characteristics of the situation. To remain in scope, this article works with four of the

most currently accepted methodologies; Figure 6 summarises why they are selected.

The parallelism showed in Section 3 between the information managed in OR and SE projects, and their lifecycle stages allow the definition of which processes of AM are parts of each stage. Table 2 shows this correspondence. It is noteworthy that, even though they have different organisations, they always fit their activities into the lifecycle stages presented previously.

4.2. Project organization

Organising a project through an AM requires defining intermediate goals, to establish *iterations*: transitional steps that incrementally build the leading towards the final product. Each one yields an improved, more refined version of the previous *release*, and the last one should be the final version.

Therefore, it is vital to determine how many iterations there would be, when will they start and end,

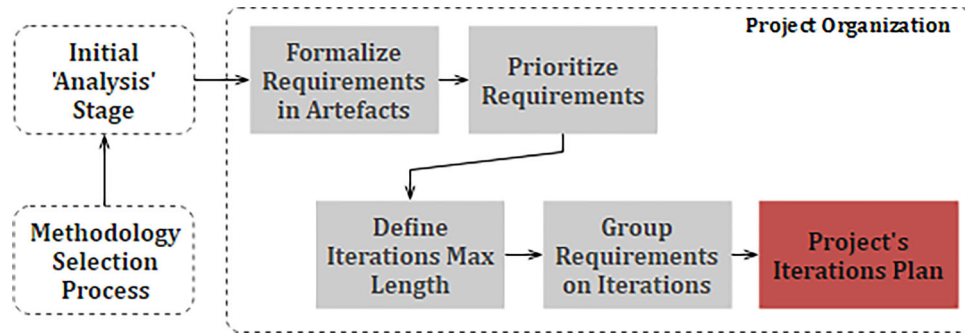


Figure 7. General steps to organise an OR intervention in iterations, following an agile philosophy.

Table 3. Match between the information states and AMs artefacts for their representation.

Information states	Agile methodologies			
	Scrum (Schwaber, 1995)	XP (Beck, 1999)	FDD (Hunt, 2006)	Crystal (Cockburn, 2004)
Requirements	User Stories	User Stories	Features Feature Sets Feature List Selected Feature List	Requirements File Use Cases Project Map & Release Plan Iteration Plan
Formal Specification	Product Backlog Sprint Backlog	Release Plan Iteration Plan Acceptance Test Unit Tests		
Mathematical Design				Test Cases Architecture Components & Build Help Training & Manual
Mathematical Model	Extensions Product Increment	Working Source Code End User Material	Overall Model Release	
Answers	Extensions		Statistics	

and which functionalities they will develop. To do this, it is required to perform an initial “Analysis,” outside any iteration. Though AMs allocate processes for doing this – i.e., Scrum’s *Initiate*, FDD’s *Develop an Overall Model*, XP’s *Listening* or Crystal’s *Project* – PSMs can be used for the same goal. Mixing them with AMs is entirely possible as agility is not rigid: it only suggests techniques. Thus, the general steps can be seen in Figure 7.

Formalising the requirements implies documenting them. How this is done depends on the specific AMs that have been selected; i.e., using FDD leads towards a *feature list*, while if using XP the team builds *user stories*. Section 4.3 will discuss the information representation.

Prioritising requirements involve deciding which ones should be developed first and why; the reasons are usually dependencies between them, the effort needed to code, and its impact on the system’s final functionality. Requirements should also have an estimated development time; this is calculated based on experience and, if the team tracks it through metrics – as done in SE (Achimugu, Selamat, Ibrahim, & Mahrin, 2014) – this value can be impartially obtained. This is done to divide a project into smaller iterations, where each one has an overall goal – i.e., making a specific part of the whole process – and a list of the included requirements. Those with a higher priority must be included at earlier iterations. This is done considering a maximum time limit so that all iterations have a similar length and work-load.

4.3 Information representation

Artefacts materialise the information evolution presented in Section 3.1. They are tangible by-products that describe a given aspect of the system, which can be represented using different notations (IEEE Computer Society, 2014). Each AM often offers different artefacts tuned to its specific proposals and activities. All of them can be framed in the states seen in Figure 1.

Table 3 condenses these relationships, but it is not an exhaustive list, only disclosing the most relevant or commonly used artefacts; descriptions are not included to keep it concise. However, no artefacts are included for “Ideas” as they are often provided as an artefact by the client; thus, even though AMs acknowledge it, they do not provide specific representations for it.

Many of these artefacts are generic elements directed towards information specific to the project management; examples are “User Stories,” “Product/Sprint Backlog,” “Features, Feature Sets,” “Use Cases,” “Iteration Plan.” Adapting them to an OR project becomes straightforward. Something similar happens to the code: AMs do not prescribe how the code is written or structured, though some of them suggest best practices – i.e., XP’s coding in pairs-; as a result, they can also be translated to OR.

Other artefacts may prove to be more challenging, such as writing test cases or generating a “Mathematical Design” as architecture, in the terms known to SE.

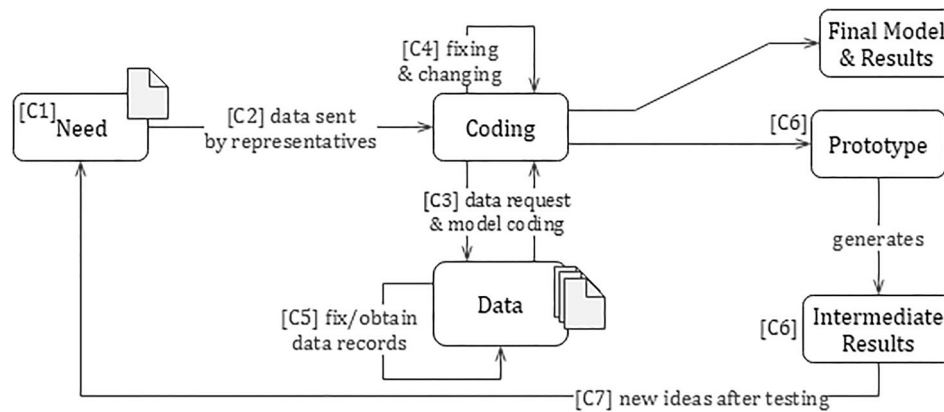


Figure 8. Original process for the RSC intervention.

OR modellers are not used to creating this type of information refinement before coding (Ackermann, 2012). However, an advantage of AMs is that they allow building them iteratively and incrementally: it starts as a general overview (i.e., FDD's Overall Model), and it is cyclically refined and expanded at each iteration through activities like *Implement, Design by Feature*. More importantly, it is not necessary to create new artefacts since there is currently an extensive gallery to choose from. This reduces the resistance to incorporate the new practice.

5. Case vignette: Retail stores chain

This section presents a case study through a reverse engineering approach of an academic collaboration performed with a local retail stores chain – herein named RSC. This was originally carried out using a traditional unstructured approach, without following any project-driven lifecycle as a guide; as a result, several problems appeared throughout the venture. This process is analysed, issues are recognised, and different possible agile solutions are presented. The contrast between the approach originally used and the Agile Operational Research highlights how agility can prevent or solve the occurrence of this type of problems. It is worth noting that this does not imply that problems need to be known before starting a project because, in most cases, this is not feasible.

RSC has retail shops in almost half the country and supplies them through two different warehouses. All the stores used SAP – a proprietary ERP – as their central enterprise system. The company asked for an evaluation of changes in the structure of its supply chain in a ten-year period, concerning specific warehouses, and to minimise its operating costs. The development of a mathematical model was proposed to estimate costs and benefits of the three available options: removing the warehouse, keeping it as-is, or transforming it to cross-docking.

The intervention started in December 2016 and completed by August 2017. Even when the result

was positive for RSC, it presented some project management difficulties. In the following paragraphs, some of the showcased texts are extracted from official communications and reports, meeting recordings, emails and modellers' notes, among others. Figure 8 shows the overall process of the RSC intervention, using the codes referenced between brackets that appear in the figure – i.e., [C1] – to relate that stage to the textual description.

At the first meeting, the representatives commented on their intentions to change the warehouse, and the need to evaluate the feasibility and convenience of carrying it forward or not. After that meeting, modellers rushed to the model development using it to represent the current situation [C1]. Because the data required were sent in emails with a significant delay, and without the corresponding documentation [C2], an additional effort was needed to understand the information received; this resulted in assumptions made to move forward early.

However, while the coding advanced, modellers required more data [C3]. The issue was that, regardless of using an SAP, RSC had inconsistent information and many sectors were isolated from the primary system, using local databases and generating information gaps. The following excerpts of different emails of the managers over a three-month period exemplify that situation:

- “[...] I apologise, but we were not able to obtain this data. I could give you only the following items [...]. We will need to start collecting the rest of it, as most of them cannot be obtained through the enterprise system, and we need to inquire the Distribution centre about them [...]”.
- “[...] The inconsistency in the cubic meters seen in December is caused by a wrong input data from some products. We are trying to locate which one is, so we can study and decide how to fix it [...]”.

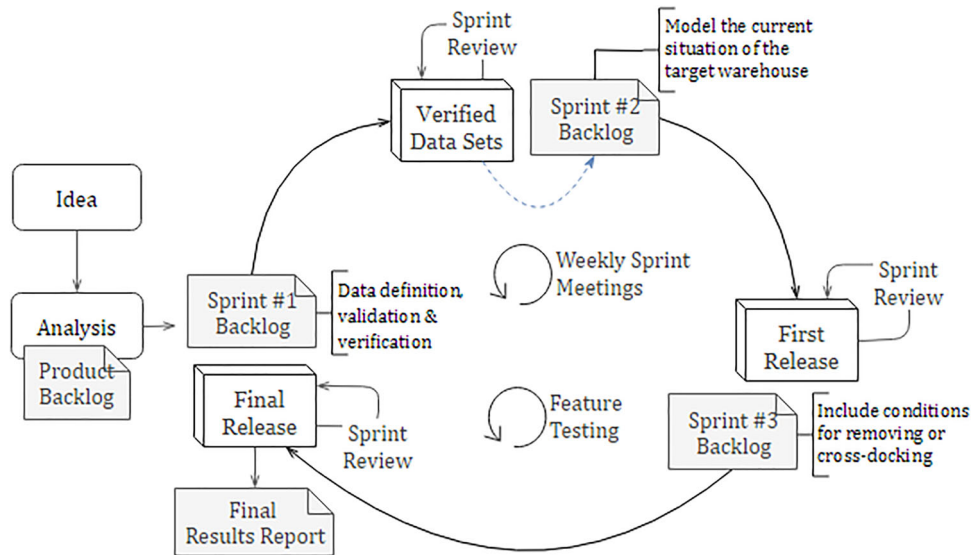


Figure 9. Proposed management of RSC's intervention using Scrum.

Then, RSC's representatives included mid-line managers in the project, without any formal introduction, and tasked them with preparing the required information [C5]. This caused delays in the generation of data and misunderstandings in communication. The following are two different requests:

- A modeller requested: "We need the number of truck travels (both for your trucks and those outsourced), on each period (week or month) for each sale point [...]". The logistic representative answered: "[...] This question is too ambiguous. Could you be more specific?".
- In another case, a modeller wrote to a mid-line manager: "[...] To improve the model, we need the supply routes you are actually using. Is it possible to obtain that information? [...]".

As RSC attempted to fix their databases, managers became interested in new requirements [C5]. These were communicated through email to the developers. The code was continuously adapted to include each request [C4] and, as there were no formally specified requirements, changes were not recorded, and the model was not versioned. Hence, what should have been final releases became prototypes with intermediate results [C6], affecting the provided results, and requesting new constraints and data to be coded [C7].

Near the end of the intervention, at a meeting scheduled to show preliminary results, the modellers discovered that they had dealt with middle managers, without any decision-making power. New requirements appeared when presenting the final report to the company's heads, some of which compromised the very purpose of the project. This deficiency in the elicitation and selection of

stakeholders delayed the project's completion far beyond schedule.

After this summary of the project life flow, it is visible that there was no consistent stream of activities, and that any form of project management was disregarded in the intervention, by both modellers and clients.

5.1. Methodology selection and process organization

Many AMs can face the same problem differently, but still, provide a global vision and a satisfactory outcome. To show there is no single, perfect choice for each project, two proposals are made regarding how the RSC situation could have been managed. The main problem, in this case, is the lack of data and the database's inconsistencies [C2, C3, C5] that delayed the project and forced developers to fix the model as new, verified data were provided continuously [C4].

The first proposal uses Scrum as AM, and its process can be seen in Figure 9.

The generation of the product backlog through an elicitation method in "Analysis" phase, makes visible the lack of data [C1–C5]. Thus, the first iteration is dedicated to determining which data are required for the model, while RSC reorganises their databases. Then, the outcome is the complete set of verified information to be used as an input in the model. The second and third iteration, develop the mathematical models through incremental sprints [C6], first for the current conditions, and then for the other decision options. New ideas [C7] can be included on each sprint after detection on the weekly meetings/reports. They are translated to *user stories*-fragments of functionalities to be

Table 4. Examples of “User Stories” for RSC’s case vignette.

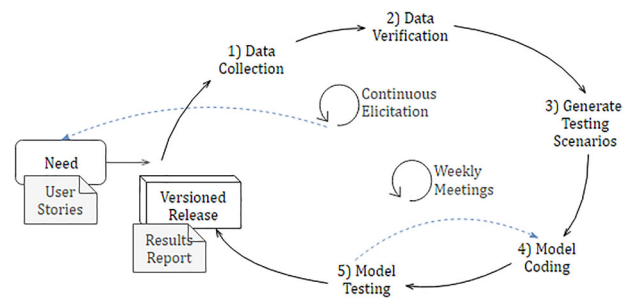
Code	Story	Relevance	Estimate	Dependencies
E01	As a manager, I want to compare the operational costs of maintaining the warehouse as is, changing it to cross-docking, or closing the location.	5		E02, E03, E04
E02	As a manager, I want to know the operational costs (salaries, products transport costs, earning from sales, warehousing fees, and taxes) for maintaining the warehouse as-is, for the next five years.	4		US12, US30, US34, US40
E03	As a manager, I want to know the operational costs (redundancy payments, products distribution costs, earnings for sales, trucks outsourcing costs, and taxes) for closing the warehouse.	4		US13, US24, US40
E04	As a manager, I want to know the operational costs (salaries, products transport, and distribution costs, earnings from sales, and taxes) for changing the warehouse to a cross-docking location and keeping it for the next five years.	4		US14, US23, US30, US40
US9	As a modeller, I want to have data (sales average, sales points locations, and distances between them, latest salaries, additional outsourcing costs, latest taxes) from the past five years in CSV format.	4	5 hours	US30
US12 US13 US14	As a modeller, I want to create input data by inferring the costs related to travelling from one sales point to another, in the scenario of [keeping the warehouse, changing to cross-docking, closing the warehouse].	2	1 day	
US23	As a modeller, I want to have constraints to define which sales points are supplied by the cross-docking location.	2	2 days	
US24	As a modeller, I want to have constraints to determine the number of products to be distributed from the central warehouse to each sales point location.	4	3 days	US14, US30
US30	As a database manager, I want to retrieve the sales volume, organised by month and by sales point, and export it to a CSV.	2	3 days	
US34	As a modeller, I want to minimise the operational costs to infer earnings for keeping the warehouse during the next five years.	4	3 hours	
US40	As a manager I want the model to export the results of each scenario to an Excel file, with charts and simple tables.	3	2 days	

implemented (see examples in Table 4) and added to the backlogs to be included on versioned models.

The second proposal uses Extreme Programming (XP), and the overall process can be seen in Figure 10. XP provides a code-centric and more straightforward approach, but still adding project management to the intervention. As it focuses on creating the tests for the functionalities before developing them (Beck, 1999), instead of moving directly to coding, the modellers first prepare the tests using the provided data. With this, RSC’s data deficiencies would have been evident sooner, giving RSC representatives time to organise a process of database unification.

Regarding the project’s organisation, a couple of initial meetings allow delineating *user stories* preventing gaps in the requirements specifications and establishing a baseline upon which to work [C1, C2, C7]; examples are shown in Section 5.2. This is revisited through weekly meetings, acknowledging the current progress, and re-discovering new requirements.

XP process could group its iterations to release versions for modelling each situation. The intervention starts building the model that reflects keeping the warehouse as-is so that RSC representatives can conduct the database unification and reorganisation. During that process, the information can be handled in *stacks* to the modellers. Each stack represents a small-iteration of the lifecycle and allows working on specific features: coding only those functionalities that have verified data and tests. This produces a *versioned release* (of the specific situation),

**Figure 10.** Proposed management of CRS’s intervention using XP.

accompanied by documentation registering each functionality and change committed to the code and a formal *results report*.

As a result, the “Development” would have progressed in parallel with RSC’s enterprise system reorganisation, producing an iteration for each stack of data, until finishing the group for the as-is situation. Then, further iterations would only require a minimal amount of additional data and would be able to skip the first two steps from Figure 10.

A critical difference between the AMs used is that XP provides an overall structure but performs smaller iterations in which execute the five activities (data collection and verification, testing scenarios and models, and coding), while Scrum has a prefixed amount of more significant iterations (even the first one is dedicated to defining the required data). Thus, while this XP organisation poses an even workload for both client and modellers, the Scrum proposal makes the first iteration more dependent on the client’s database reorganisation.

5.2. Artefacts examples (XP)

Two different possibilities were shown for RSC: XP and Scrum. To keep this article in focus, only fragments of the artefacts created for XP are presented, to act as illustrations.

The first artefacts are “User Stories,” used to formalise the “Ideas” into “Requirements.” They should be prioritised, to later define the iterations. Table 4 proposes examples, along with three prioritisation criteria: estimated development time, if it depends on other stories and relevance for the client (with five being the most relevant). The codes of Table 4 are deliberately designated to skip numbers: this is to show that there should be many more stories.

It is worth noting that user stories are always written from a specific stakeholder role’s point of view. They are often detailed while developers and stakeholders brainstorm together and structured to be easy to read to the latter. They must be simple enough to portray the requirements straightforwardly but reducing possible misunderstandings. Some interesting points can be discussed in these examples:

- There can be *epics*: stories that are too generic and need to be specified into smaller stories. [E01] and its subdivisions in [E02], [E03], and [E04] are examples these. The subdivisions can also be considered epics, but are used for grouping the iterations to build each specific model, as mentioned in Section 5.1.
- There should be a story for each input data that needs to be inferred from the existing data. The stories [US12], [US13], and [US14] show that data inference must also be documented for each scenario that needs to be developed.
- Likewise, there should also be stories for each group of constraints on each scenario (such as

[US23] and [US24]) and stories for each objective to be used (i.e., [US34]).

- The expected type of reports to be generated also need to be addressed as stories, as in many cases this needs to be coded as part of the model; [US40] is an example of this.
- Obtaining and organising the input data was vital. It is suitable to represent each data stack requirement as a story itself. This is done from the database manager’s perspective (as in [US30]) and the modellers’ standpoint (i.e., [US9]).

With this, it is possible to document the iterations through a “Release Plan,” to indicate when each iteration should be completed, and the “Iteration Plans” for each one of them. This can be done using many of the available online tools existing for software development projects.

In particular, Figure 11 shows a “Release Plan” created with FeatureMap (Salience, 2018), which can be used with most AMs. Furthermore, most of these tools allow collaboratively updating of the diagrams, reflecting changes to the iterations. As the agile values state, it is better to respond to change than to follow a plan (value V4) strictly.

Regarding the “Mathematical Design,” Section 3.1 defined that it should be generated following the concept of *points-of-view*; this implies showcasing the model from the perspective of different stakeholders. Therefore, it also aligns with how the *user stories* are written. An important element of the “Mathematical Design” is that it is also incrementally developed: only a general structure is presented at the beginning of the project, and then it is iteratively refined at each iteration; this results on a final, detailed structure, created alongside the project and model.

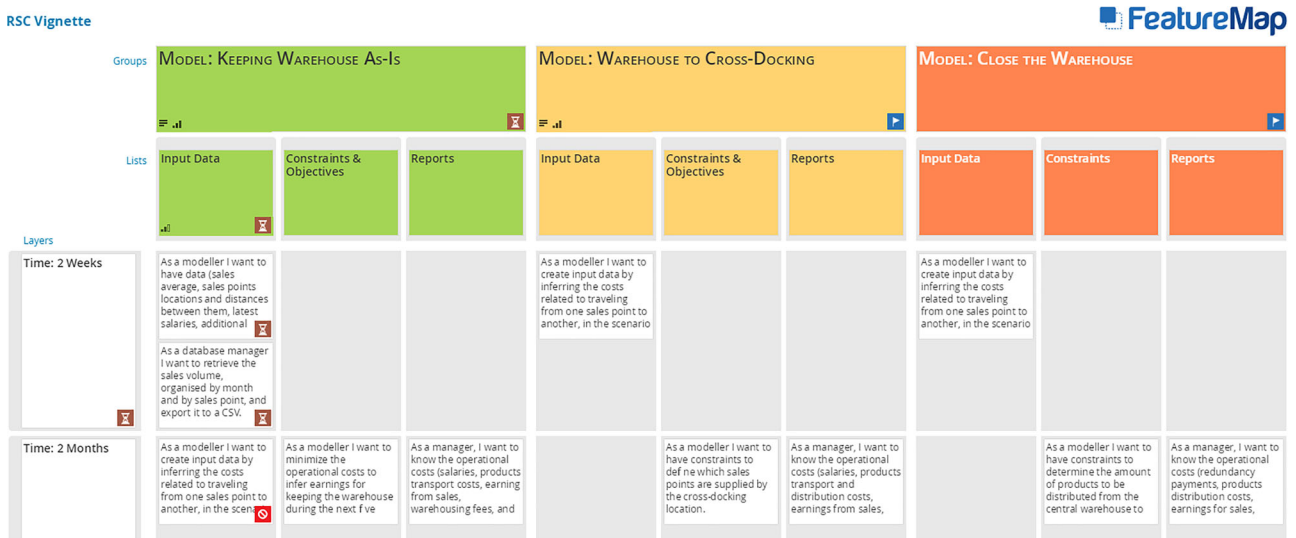


Figure 11. Example of “Release Plan” created using FeatureMap.

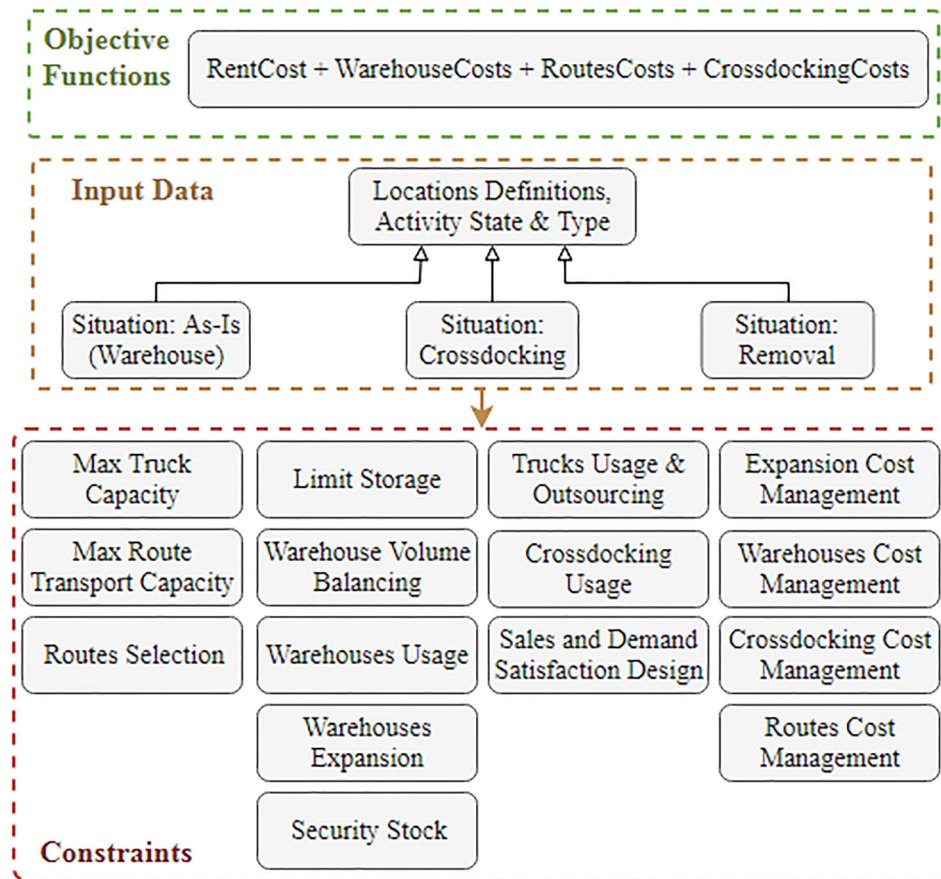


Figure 12. “Mathematical Design” for the case vignette, from a developers point-of-view. This is an initial model, previous to any project iteration.

To keep the article in scope, only two possible views of the case vignette are presented; both of them belong to a general, pre-coding state.

In particular, Figure 12 shows a general structure diagram, similar to SE packages; it is targeted at modellers and project leaders and shows which principal components are present in the model. Here, as there is a considerable number of input data, only those relevant to understand the general structure are part of the diagram. Each situation has an input data that defines what each location is (a warehouse, a sales point, a cross-docking, and others) and if it is active or not; constraints use that specific input data in order to simulate each situation. This is possible because these types of input data are written with the same structure (as represented by the arrows). Furthermore, variables are not present because, at this stage, there is no need to declare them concretely.

However, a decision-maker stakeholder is not usually interested in understanding coding details. Therefore, their point-of-view of the “Mathematical Design” is different, as seen in Figure 13. Here, they see how the models are used to obtain the answers they need: each situation is simulated to obtain individual results, and then they are compared and presented as a report. In particular, this is using BPMN

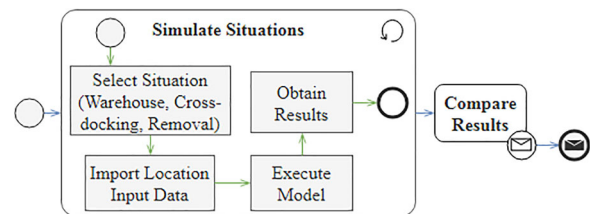


Figure 13. “Mathematical Design” for the case vignette, from a decision-maker client point-of-view. This is an initial model, previous to any project iteration.

(Business Process Model Notation), which is standard nomenclature for showcasing business processes.

6. Discussions and limitations of the proposal

Accepting and validating AMs in SE was a long process, which took around 10-years, requiring practitioners to apply these methods in practice and researchers to study them (Cardozo, Araújo Neto, Barza, França, & da Silva, 2010; Dingsøyr et al., 2012; Kneuper, 2017). As a result, ensuring the success of this article’s proposal would be hurried, since it would require applying the same process, by using agility in many OR interventions (Ormerod, 2008). Although AM advantages in SE are verifiable, its applicability to OR interventions may encounter the

resistance to move to new practices. Furthermore, it should be noted that poor project management does not necessarily imply its failure or that of its solution. Because of this, this article introduces AMs to OR by stating the feasibility of the adaptation, aiming to have more reports on their use and results.

It is important to mention that AMs are compatible with existent PSM or Soft-OR techniques, as they do not prescribe specific techniques. Moreover, AMs-based proposed techniques are not mandatory either, as these lifecycles support process flexibility above all.

An advantage of the proposal is converting the information into tangible artefacts and using this in OR potentially increase its adoption in different organisations. SE artefacts and AMs are extensively used in Information and Communications Technologies (ICT) departments as well (Cardozo et al., 2010), and can reinforce the integration, by providing a seamless link to the use of the newest techniques (Ranyard, Fildes, & Hu, 2015).

On the other hand, many of the concepts introduced here may be considered as basic for SE. Nonetheless, their use and modification to fit OR concepts and practices is novel. This is done by defining concepts in common, and clear steps to adopt any AM to OR based on the underlying principles. As changes usually require a transition period where more effort is made to accept them, then, though AMs are based on the same values and principles, it is possible that code-centric methods -such as the XP or Crystal- appeal more to practitioners, as they are closer to current practices.

Because each OR intervention has its intrinsic characteristics that depend on the situation being analysed, it is not possible to provide a universal, specific sequence on how to adapt every AM, as this will ignore the changeable nature that characterises most situations. Therefore, training practitioners in the selected AMs techniques is critical to being able to use them on OR interventions. Although this is not part of the scope of the article, the authors have evaluated the possibility of working in coordination with another academics specialized in OR to apply this proposal in future interventions to local industries, to not only train them on AM but also apply these methodologies in practice.

7. Conclusions

This article acknowledges that project management has become critical to modern organisations and that, as a result, OR models should be created taking into account the continuously changing environments, their role in the decision-making process and the constant unclear ramifications the target

situations may have. This perspective differentiates from traditional model-focused processes, also studying its integration with existing systems. In addition, it contributes to addressing models as systems *per se*.

SE agility is particularly strong for taking care of increasingly demanding projects, favouring the ability to adapt to changes and establishing short delivery deadlines that imply faster investment returns. However, current PSM and Soft-OR focus only on understanding the problem and ignoring the other stages without acknowledging several aspects of a full lifecycle.

As a result, this article proposes the challenge of adapting and using SE agility to manage OR interventions. As it was in SE, an extensive, widespread use of AMs in OR interventions is mandatory to obtain data generalizable enough to discuss advantages and drawbacks in more detail. Therefore, this article introduces AMs to OR by stating the feasibility of the adaptation, aiming to have more reports on their use. This is founded on the fact that many concepts can be adapted from SE to OR, as both disciplines have a similar project lifecycle, as well as an equivalent evolution of information. General guidelines are provided to instruct on how to port Agile Methods to OR projects. This is exemplified in a case study vignette, using both Extreme Programming and Scrum.

However, this is only the front end of a larger project, enabling possibly future works. The most relevant is using AM such as Scrum or XP in real-world interventions with local industries from different backgrounds. In addition, other related concepts can be explored, such as formalising requirements and quality attributes for OR, as well as studying the impact that code and design smells have in this type of projects. Moreover, other specific agile techniques can be explored in the context of OR interventions, such as estimations, prioritisations and sprint retros, among others. Finally, as agility derives from SE, it would be possible in the future -after several agile OR interventions are performed- to study how both disciplines can be integrated through this shared project management process.

Acknowledgements

The authors gratefully acknowledge the anonymous reviewers whose comments and suggestions helped improve and clarify this manuscript. Furthermore, authors are especially thankful to Prof Richard Ormerod for his contributions and insight.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Melina Vidoni  <http://orcid.org/0000-0002-4099-1430>
 Aldo Vecchietti  <http://orcid.org/0000-0002-0791-9496>

References

- Achimugu, P., Selamat, A., Ibrahim, R., & Mahrin, M. (2014). A systematic literature review of software requirements prioritization research. *Information and Software Technology*, 56(6), 568–585. doi:10.1016/j.infsof.2014.02.001
- Ackermann, F. (2012). Problem structuring methods ‘in the Dock’: Arguing the case for Soft OR. *European Journal of Operational Research*, 219(3), 652–658. doi:10.1016/j.ejor.2011.11.014
- Ackermann, F., Bawden, R., Bosch, O., Brocklesby, J., Bryant, J., Buede, D., ... White, L. (2009). *The case for Soft O.R.* INFORMS Pubs Online.
- Ackoff, R. (1979). The future of operational research is past. *Journal of the Operational Research Society*, 30(2), 93–104. doi:10.1057/jors.1979.22
- Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E., & Sarwar, S. (2010). Agile software development: Impact on productivity and quality. *IEEE International Conference on Management of Innovation and Technology (ICMIT)* (pp. 287–291). Singapore: IEEE.
- Beck, K., Beedle, M., Bennekum, A. C., Cunningham, W., Fowler, M., Grenning, J., ... Thomas, D. (2001). *Manifesto for Agile Software Development*. Retrieved 2018, from <http://www.agilemanifesto.org/>.
- Beck, K. (1999). *Extreme programming explained—Embrace change* (1st ed.). Boston, USA: Addison-Wesley Professional.
- Birrell, N., & Ould, M. (1985). *A practical handbook for software development* (1st ed.). New York, USA: Cambridge University Press.
- Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4), 22–24. doi:10.1145/12944.12948
- Boehm, B. (2006). A view of 20th and 21st century software engineering. 28th International Conference on Software Engineering (ICSE) (pp. 12–29). Shanghai, China: ACM New York. doi:10.1145/1134285.1134288
- Cabrera, D., Cabrera, L., Powers, E., Solin, J., & Kushner, J. (2018). Applying systems thinking models of organizational design and change in community operational research. *European Journal of Operational Research*, 3(1), 932–945. doi:10.1016/j.ejor.2017.11.006
- Cardozo, E., Araújo Neto, J., Barza, A., França, A., & da Silva, F. (2010). Scrum and Productivity in Software Projects: A Systematic Literature Review. 14th International Conference on Evaluation and Assessment in Software Engineering (EASE) (pp. 131–134). Swindon, UK: BCS Learning & Development Ltd. doi:10.14236/ewic/EASE2010.16
- Checkland, P., & Poulter, J. (2010). Soft systems methodology. In M. Reynolds & S. Holwell (Eds.), *Systems approaches to managing change: A practical guide* (1st ed., pp. 191–242). London, UK: Springer London.
- Churchman, C. (1967). Wicked problems. *Management Science*, 14(4), B-141–B-146. doi:10.1287/mnsc.14.4.B141
- Cockburn, A. (2004). In A. Cockburn & J. Highsmith (Eds.), *Crystal clear: A human-powered methodology for small teams* (1st ed., Chapters 2 and 4). USA: Addison-Wesley Professional.
- da Silva Filho, M. (2015). Problem structuring methods recommendation for a public organization of the Rio de Janeiro State. *Procedia Computer Science*, 55, 196–202. doi:10.1016/j.procs.2015.07.033
- Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6), 1213–1221. doi:10.1016/j.jss.2012.02.033
- Doshi, V., & Patil, V. (2016). Competitor driven development: Hybrid of extreme programming and feature driven reuse development. *International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)* (pp. 1–6). Pudukkottai, India: IEEE.
- Eden, C., & Ackermann, F. (2001). SODA— The principles. In J. Rosenhead & J. Mingers (Eds.), *Rational analysis for a problematic world revisited: Problem structuring methods for complexity* (2nd ed., pp. 21–41). New York, United States: John Wiley & Sons.
- Frakes, W., & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, 35(7), 529–536. doi:10.1109/TSE.2005.85
- Friend, J. (2006). Labels, methodologies and strategic decision support. *Journal of the Operational Research Society*, 57(7), 772–775. doi:10.1057/palgrave.jors.2602089
- Gligor, D., Esmark, C., & Holcomb, M. (2015). Performance outcomes of supply chain agility: When should you be agile? *Journal of Operations Management*, 33–34, 71–82. doi:10.1016/j.jom.2014.10.008
- Haefliger, S., von Krogh, G., & Spaeth, S. (2008). Code reuse in open source software. *Management Science*, 54(1), 180–193. doi:10.1287/mnsc.1070.0748
- Hunt, J. (2006). Feature-driven development. *En Agile Software Construction* (Primera ed., pp. 161–182). Londres, Inglaterra: Springer-Verlag London Limited.
- IEEE Computer Society. (2014). *Guide to the software engineering body of knowledge* (3rd ed.). IEEE.
- ISO/IEC/IEEE. (2011). *42010:2011 – Systems and software engineering – Architecture description*. In C. I. Engineering (Ed.), Switzerland: International Standardization Organization.
- Kneuper, R. (2017). Sixty years of software development life cycle models. *IEEE Annals of the History of Computing*, 39(3), 41–54. doi:10.1109/MAHC.2017.3481346
- Livermore, J. (2007). Factors that impact implementing an agile software development methodology. *IEEE SoutheastCon*. (pp. 82–86). Richmond, VA, USA: IEEE.
- Melo, C., Cruzes, D., Kon, F., & Conradi, R. (2011). Agile team perceptions of productivity factors. *Agile Conference (AGILE)* (pp. 57–66). Salt Lake City, UT, USA: IEEE.
- Midgley, G., Cavana, R., Brocklesby, J., Foote, J., Wood, D., & Ahuriri-Driscoll, A. (2013). Towards a new framework for evaluating systemic problem structuring methods. *European Journal of Operational Research*, 229(1), 143–154. doi:10.1016/j.ejor.2013.01.047
- Mingers, J. (2011). Soft OR comes of age—but not everywhere!. *Omega*, 39(6), 729–741. doi:10.1016/j.omega.2011.01.005
- Mingers, J., & White, L. (2010). A review of the recent contribution of systems thinking to operational research and management science. *European Journal of*

- Operational Research*, 207(3), 1147–1161. doi:[10.1016/j.ejor.2009.12.019](https://doi.org/10.1016/j.ejor.2009.12.019)
- Ormerod, R. (2008). The transformation competence perspective. *Journal of the Operational Research Society*, 59(11), 1435–1448. doi:[10.1057/palgrave.jors.2602482](https://doi.org/10.1057/palgrave.jors.2602482)
- Ormerod, R. (2014). The mangle of OR practice: Towards more informative case studies of ‘technical’ projects. *Journal of the Operational Research Society*, 65(8), 1245–1260. doi:[10.1057/jors.2013.78](https://doi.org/10.1057/jors.2013.78)
- Project Management Institute. (2017). *A guide to the project management body of knowledge (PMBOK® guide)* (6th ed.). USA: Project Management Institute, Inc.
- Qumer, A., & Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology*, 50(4), 280–295. doi:[10.1016/j.infsof.2007.02.002](https://doi.org/10.1016/j.infsof.2007.02.002)
- Ranyard, J., Fildes, R., & Hu, T. (2015). Reassessing the scope of OR practice: The influences of problem structuring methods and the analytics movement. *European Journal of Operational Research*, 245(1), 1–13. doi:[10.1016/j.ejor.2015.01.058](https://doi.org/10.1016/j.ejor.2015.01.058)
- Royce, W. (1970). Managing the development of large software systems. *Proceedings of the IEEE WESCON* (pp. 328–338). IEEE.
- Salience. (2018). *FeatureMap Product Tour*. Retrieved from <https://www.featuremap.co/en/tour>.
- Schramm, V., & Schramm, F. (2018). An approach for supporting problem structuring in water resources management and planning. *Water Resources Management*, 32(9), 2955–2968. doi:[10.1007/s11269-018-1966-9](https://doi.org/10.1007/s11269-018-1966-9)
- Schwaber, K. (1995). SCRUM development process. In J. Sutherland, C. Casanave, J. Miller, P. Patel, & G. Hollowell (Eds.), *Business object design and implementation* (1st ed., pp. 117–134). Austin, Texas, USA: Springer, London.
- Smith, C., & Shaw, D. (2019). The characteristics of problem structuring methods: A literature review. *European Journal of Operational Research*, 274(2), 403–416. doi:[10.1016/j.ejor.2018.05.003](https://doi.org/10.1016/j.ejor.2018.05.003)
- Tarhan, A., & Yilmaz, S. (2014). Systematic analyses and comparison of development performance and product quality of Incremental Process and Agile Process. *Information and Software Technology*, 56(5), 477–494. doi:[10.1016/j.infsof.2013.12.002](https://doi.org/10.1016/j.infsof.2013.12.002)
- Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2011). The agile requirements refinery: Applying SCRUM principles to software product management. *Information and Software Technology*, 53(1), 58–70. doi:[10.1016/j.infsof.2010.08.004](https://doi.org/10.1016/j.infsof.2010.08.004)