International Journal of Smart Vehicles and Smart Transportation
Volume 6 • Issue 1

An Agile Approach for Lifecycle Integration in Personal Rapid Transit Systems Engineering

Nicholas Davenport, Deloitte, UK
Theo Tryfonas, University of Bristol, UK*
https://orcid.org/0000-0003-4024-8003
Alan Peters, Connected Places Catapult, UK
Stylianos Karatzas, University of Cambridge, UK
Anastasios Ioannis Karameros, University of Patras, Greece

ABSTRACT

Dependable systems pose particular challenges to system developers who try to implement agile approaches to tackle the problem of requirements scope creep. However, legislation compliance, safety case development, and other strong contextual influences may be seen to inhibit the implementation of any approaches other than the traditional linear life cycles, even though agility may be able to improve the development process in parts. This article discusses key success factors when integrating agile with structured systems development life cycle approaches. The authors adopt an empirical approach and analyse a historical case study of a personal rapid transit (PRT) system, reflecting on key factors and relating those to the relevant literature. Based on these experiences, a model for the integration of agile with structured systems lifecycle models in dependable systems is developed. This model addresses the challenge of integrating multiple lifecycles of potentially conflicting objectives within a single programme.

KEYWORDS

agility, lifecycle integration, personal rapid transit, systems engineering

INTRODUCTION

Agile software development methodologies have become prevalent in IT companies seeking to develop software in high-risk, fast-changing commercial environments (West & Grant, 2010). These methods help manage risk, uncertainty, and unforeseen change via early and frequent releases of working software, collaborative development in small teams, customer involvement, and adaptive processes.

DOI: 10.4018/IJSVST.324063
*Corresponding Author

Agile is an incremental and iterative software development approach. In Agile methodologies planning is done at the initial stage; however, throughout the project, changes are accepted along with constant feedback provided by users for the improvement of the project (Kumar, 2019; Cockburn, 2001, 2008; Highsmith, 2002; Boehm, 2002; Boehm & Turner 2003a, 2003b; Schwaber, 2004; Augustine, 2005). This process is a significant departure from conventional, plan-driven methods that attempt to foresee change through rigorous up-front requirements gathering, analysis, and design. However, in many cases, there is a need to balance Agile methods with plan-driven methods to provide both high flexibility and high assurance (Boehm, 2002). Increasing attention has been given to the use of Agile methods for developing systems comprising software and hardware (Turner, 2007; Smith, 2007, 2009; Rothman, 2007; Hugos, 2009). Projects with high levels of uncertainty and high rates of unforeseen change are likely to benefit from iterative design, frequent integration, and active involvement of stakeholders to establish, prioritize, and verify requirements as they become better understood. However, plan-driven methods are widely regarded as essential for systems that need up-front architecture to support hardware integration and early safety assurance. Novel system development projects are likely to benefit from combined elements of both plan-driven and Agile methods in achieving high flexibility and high assurance. Recent literature reports on projects in which Agile methods have been applied even in projects where the traditional plan-driven approach used to be mainstream (e.g., regulated and safety-critical environments) because they offer more efficient and flexible ways to produce software (Mansoor et al., 2019; Van Waardenburg & Van Vliet, 2013; Górski & Łukasiewicz, 2013; Jonsson et al., 2012; Mostashari et al., 2012). Baskeville et al. (2011) have shown that companies can successfully combine Agile and plan-driven approaches to achieve the benefits of each method.

This paper examines the feasibility of multiple life cycle approaches integration under a single program. Based on the experience of application of Agile methods used to enhance the development and delivery of a major U.K. airport's personal rapid transit (PRT) system, this work contributes new ideas on how agility may benefit new system development projects, in cases where compliance and certification requirements (e.g., safety cases) dictate the use of structured systems development approaches for certain subsystems. In this paper we synthesize state-of-the-art literature to suggest how Agile principles and practices can be used to provide a degree of agility in new hardware/software development projects. We also describe our research approach and present a case study on the delivery of the world's first commercial PRT system at a major U.K. airport. We conclude the paper with a discussion and our final conclusions.

BACKGROUND AND EXISTING WORK

Agile Versus Structured Systems Development

There is ample discussion in the literature of systems engineering (SE) on the question of how structured approaches compare against Agile methods. Boehm (2002) distinguished between Agile and plan-driven methods for software development projects and also suggested how a degree of agility can be brought to plan-driven methods. This research extended this thinking to broader systems development methods that integrate both hardware and software. "Plan-driven methods" refer to traditional systems engineering and project management practices and are considered as the conventional way to develop large complex systems. Plan-driven methods adhere to specific, predefined processes in moving the system through a series of planned development phases (system development life cycle). Broadly, these processes focus on establishing requirements, establishing an architecture, decomposing the system into logical subsystems, designing the subsystems, building the subsystems, testing the subsystems, integrating the subsystems, and testing the system; that is, the V model.

There is a concern for completeness of documentation at every step of the way to provide thorough verification of the system, traceability between requirements and design specifications, and verification of the processes themselves for quality assurance purposes. The intended outcome is a fully operational capability matched to the needs of clients, end users, and stakeholders (Stevens, 1998). The original tendency was to view the system development cycle as a waterfall from concept through to the end product (Schlager, 1956; Hall, 1962; Goode, 1957). However, incremental and iterative processes have been used for several decades in software engineering (Larman & Basili, 2003). In the mid-1980s incremental and iterative processes were adopted to deal with the increasing participation of software in system developments, but still with strong documentation and traceability mandates across all development activities (Boehm et al., 2010; Boehm & Łukasiewicz, 2013).

In contrast, Agile methods are more lightweight processes that employ short iterative cycles; actively involve users/customers to establish, prioritize, and verify requirements; and rely on tacit knowledge within a team as opposed to documentation (Ramesh et al., 2010; Boehm & Turner, 2003a, 2003b). They employ iterative and incremental development in which planning is on the next iteration/increment only, and each iteration/increment is planned according to the needs of the customer or end-user representative. Agile methods exploit several properties of software in allowing continuous integration and test-driven development. Software can be cost-effectively redesigned and refactored to accommodate changing requirements. Agile methods employ Lean principles, such as minimizing waste, doing what is sufficient, and delaying decisions to the last feasible moment. Teams are self-organizing and development practices are emergent. Processes, principles, and work structures are recognized during the project, rather than being predetermined, and systems are developed in an incremental (small software releases, with rapid cycles), cooperative (customer and developers working constantly together with close communication), straightforward (the method itself is easy to learn and to modify, well documented), and adaptive (able to make last moment changes) way (Abrahamsson et al., 2017).

The need for a balance between Agile and plan-driven methods in organizations and individual projects is well understood in the software domain (Boehm, 2002; Vinekar, 2006) because companies need to create value quickly as well as provide high assurance of their product. Boehm (2002) suggested that projects can be positioned on a continuum from purely Agile to purely plan-driven, indicating their reliance on plans, and offers a risk-based strategy for deciding the level of planning required. Karlström (2006) considered the need for embedding Agile methods in conventional stage-gate project management models. Assuming that a project is either traditional or Agile, Vinekar suggested that two distinct, separate cultures must be maintained if an organization is to successfully undertake both types of projects in parallel (Vinekar, 2006). Nerur (2005) suggested that Agile methods are attractive for highly innovative projects in which the customer places high value in the outcome; however, this researcher suggested that organizations should be circumspect in integrating Agile methodologies with existing plan-driven cultures. Examples of where one approach has necessarily been replaced by the other can be identified, such as in the development of Boeing Commercial Airplanes' wiring system design software tool (Bedoll, 2003). Turner (2007) argued that although it is obvious that there are large differences between the largely plan-driven Capability Maturity Model Integrated (CMMI) and Agile methods, both approaches have much in common. He believes that neither way is the "right" way to develop software, but that there are phases in a project where one of the two is better suited. Turner (2007) also suggested that one should combine the different fragments of the methods into a new hybrid method.

Smith (2007) advocated the use of Agile principles in the development of non-software systems, showing how many of the characteristics of software that Agile methods exploit can apply equally to systems that also contain hardware. Smith emphasized the importance of modular system architectures and the benefits of software-intensive design in creating flexibility and thus managing risk and uncertainty. He discussed experimentation and set-based design, along with the Lean concept of delaying decisions to the "last responsible moment." Jorgensen's Project Analyzer (Smith, 2007)

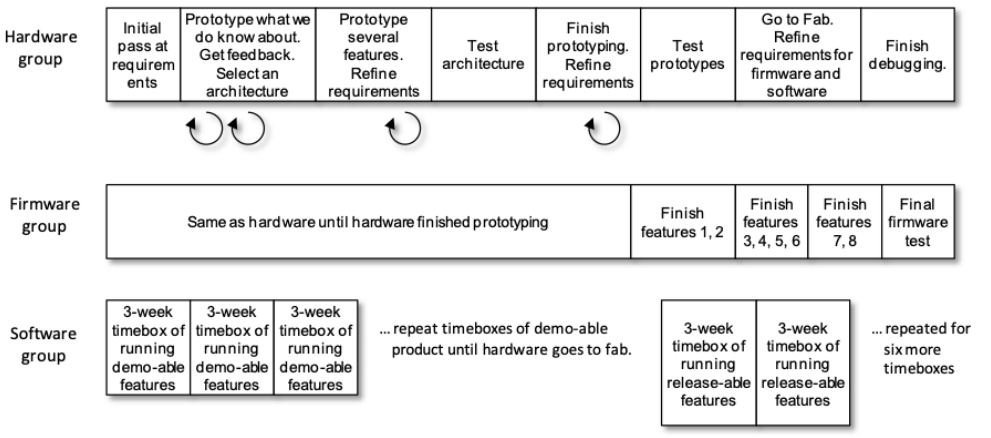
showed how different aspects of a project lend themselves to particular development approaches, suggesting the simultaneous pursuit of Agile and plan-driven methods. This indicates the amount of project overhead relative to the residual effort left for actual product development, thus giving an insight into the "agility" of a given project. Turner (2007) recognized the potential benefits Agile principles can bring to systems engineering, given that systems are becoming increasingly software intensive. Turner suggested that iterative and flexible development, Lean concepts, continuous integration, and test-driven development are all possible for a system that relies predominantly on software. Boehm (2005) agreed that system architecture needs to start with the software architecture, and Turner (2007) suggested that software provides capability, enables flexibility, and represents the majority of the value of modern systems.

Rothman (2007) considered different project life cycle approaches for hardware/software systems on a continuum from "serial" to "iterative," "incremental," then "iterative/incremental," or "Agile," where each approach is intended to manage different levels and sources of risk with different levels of feedback. The first three categories are considered as plan-driven methods because, in systems engineering, they tend to have strong documentation and traceability mandates. However, incremental or iterative approaches are thought to provide a degree of "agility" over serial approaches, allowing requirements and system architecture to change and evolve through the use of iterations or increments. Rothman showed how a hardware/software project can combine these different development life cycles for hardware, firmware and software (Figure 1).

Life Cycle Integration Related Work

Agile practices have gained considerable popularity in industry and research communities. The latest State of Agile Survey 2018 (Agile State Report, 2018) revealed that the top three reasons that drive organizations to adopt Agile were to accelerate product delivery, to manage changing requirements, and to increase productivity. Examples of the emergence of Agile methods in large-scale projects can be found in companies such as Lockheed Martin (Boehm, 2006). Furthermore, companies have started to successfully combine Agile and plan-driven approaches to receive the benefits of each approach (Baskeville et al., 2011). Millard et al. (2014) demonstrated how the development of a large command-and-control (C2) system can benefit from the combination of Agile practices and plan-driven approaches. Similarly, Hallberg et al. (2010) proposed an Agile architecture framework for developing C2 systems, and the use of their proposed architecture framework was examined in three

Figure 1. A combination of life cycles in a hardware/firmware/software project
(Adapted After Rothman, 2007)



case studies based on Swedish military projects. Hallberg et al. (2010) argued for the framework's appropriateness for developing C2 systems, noting that it can be extended to support the development of other systems.

Agile software development provides benefits, but also introduces challenges with regard to project risks (Ramesh et al., 2010). Baskeville et al. argued (2011) that the term Agile practices is vague and can be attributed to anything. Górski and Łukasiewicz (2013) focused on the risk of integrating Agile practices in critical software, while satisfying software assurance requirements. Górski and Łukasiewicz (2013) conducted a user study and found that to conform to the requirements of safety-critical projects, different approaches of Agile practices should be employed. Jonsson et al. (2012) investigated if Agile practices conform to EN 50128, a standard that regulates safety-related software for railway applications. They found that Agile practices can be tailored to fit in this regulated development environment, offering, for instance, more efficient development, but can also cause conflicts with this standard.

Fontana et al. (2014) attempted to define and explore the maturity of software development in teams comprising Agile practitioners, by conducting a user study and statistical analysis. They found that maturity in Agile software development should be people centered (i.e., based on Agile practices such as flexibility and agility), rather than prescriptive processes. Boehm (2013) discussed how recent trends (such as big data and cloud computing) influence the integration of systems and software engineering processes. Machine learning approaches can affect the software life cycles with the ability to integrate early models and simulation results from historic or synthetic data (Kumar, 2019). To address these challenges, Boehm proposed an incremental commitment spiral model process framework and a set of process strategies.

RESEARCH APPROACH

For this paper we used inductive case study research as a method to develop theory (as per Marshall, 1999) by examining the extent to which Agile methods and principles can be integrated with structured systems development and how. We used previous experiences of the project, together with the Agile concepts developed from the literature, as a basis for inquiry into the development processes before, during, and after our involvement. This involvement includes in summary participation in the project management team, developing related systems, conducting interviews with relevant stakeholders during and after the development period, and reviewing transcripts and deliverables. One of us had direct involvement with the management and delivery of the project, another had academic oversight of their output, and two others contributed to the retrospective analysis of the material and the development of the theoretical framework.

Data collection for this case study was based on one-to-one semistructured interviews, focus groups and informal discussions with company personnel, project meeting minutes, and documentation reviews. Much of this data was compiled into a reflective diary, which later formed the basis for analysis techniques, including basic coding, analytic induction, critical incident analysis, and narrative analysis. These techniques were used to identify similarities to the Agile project management and Agile software development literature, which in turn led to further inquiry and assimilation of data. As such, our results cannot be considered generalizable without taking into consideration our specific assumptions because they depend on features of the PRT project and the company that implemented it. However, we believe that our experience in delivering the PRT project can help other new systems developers who use Agile principles and practices, especially where convention dictates an entirely plan-driven approach.

We developed the case study after the primary contributor's involvement to the project. It focused on the same events, from a developer and even an executive viewpoint, helping to corroborate findings from each research method and contrast between developer culture and management culture. Through the course of reflection and analysis of the evidence as discussed above (basic coding, analytic

induction etc.), we looked at decisions and events primarily from a management viewpoint and did not fully consider many aspects of development (it was not intended to do so). Emergent requirements and tasks may well have been by-products of a development process not yet acknowledged by the management team. Further literature was then found addressing the need for Agile methods and principles and their application to conventional SE practice. We used these concepts to identify mechanisms behind the successes and difficulties seen during the period of involvement and to suggest changes to the management approach, such as the introduction of Agile project metrics. In particular, our work in managing early integration and test of new communications equipment and the development of operator interfaces was deemed to have played a crucial part in the success of the overall program.

CASE STUDY: AN AIRPORT PRT PROJECT

Project Overview

This case study examines how a degree of agility was achieved during the implementation of a PRT system at the business car park of a major U.K. airport, complementary to the need for plan-driven approaches to support hardware and safety-critical aspects. We used Boehm's project characteristics (Boehm, 2003) in the following sections to point out how a degree of agility was achieved in terms of application, management, technical, and organizational attributes. We examined characteristics originally set forth by Boehm and Turner (2003a) to point out how a degree of "agility" can be brought to new systems development projects, suggesting how the use of Agile methods, or techniques from Agile methods, could exploit properties of software-intensive systems. These comparators were used in this case study to support these ideas with evidence (see Table 1). These attributes include the following:

- Application characteristics, including primary project goals, criticality, project size, and application environment
- Management characteristics, including customer relations, and planning and control
- Technical characteristics, including approaches to requirements definition, development and test
- Organizational characteristics, including customer characteristics, developer characteristics, and organizational culture (Note that the original model's "personnel characteristics" have been broadened to include team working and process.)

The airport PRT system is a relatively isolated system that interacts primarily with humans and simple external information systems. The developers did not have major external interfaces to worry about other than the passenger interfaces and an operator human-machine interface (HMI). It is expected that future generations of PRT systems will integrate with other transport equipment and more sophisticated information systems, promoting the need for more rigorous up-front planning and control of external interfaces. In terms of systems involved, the PRT project can be thought of as consisting of the following elements:

- Contracted civil works, including guideway construction, and station and operation and maintenance (O&M) facility construction
- PRT system equipment and systems, including PRT vehicles, control and communications systems (including track-side equipment/systems), station and berth equipment and systems, and O&M facility equipment and systems

Most contractor civil works were completed before the installation of PRT equipment and systems. However, PRT equipment and systems began development well before the design and construction

phase, with parallel development and testing undertaken at a separate test facility in another city. The primary focus of this case study is on the final development and testing of PRT systems once the PRT equipment had been installed in the target environment. The basic architecture of this system is shown in Figure 2. The system essentially consists of automated guided vehicles managed via software-based real-time control systems, including vehicle-based guidance-navigation-and-control systems, central control, station control and berth control (berth doors, vehicle sensors, destination panel, vehicle charger, etc.). The central control system manages routing and scheduling of vehicles, empty vehicle management, fault detection and response, and display of system information to the operators in the control room. The central control system communicates with station controllers over a wired network, and both central and station controllers communicate with vehicles via a wireless communication system. Both the central control and station control systems run on customized PCs. Berth controllers manage berth door controllers, vehicle presence sensors, vehicle chargers, and the passenger destination selection panels. These run on single board computers and are developed in Microsoft's .NET environment. Ethernet connections between station and berth controllers allow these applications to be built and tested over the network from a single location (i.e., the control room).

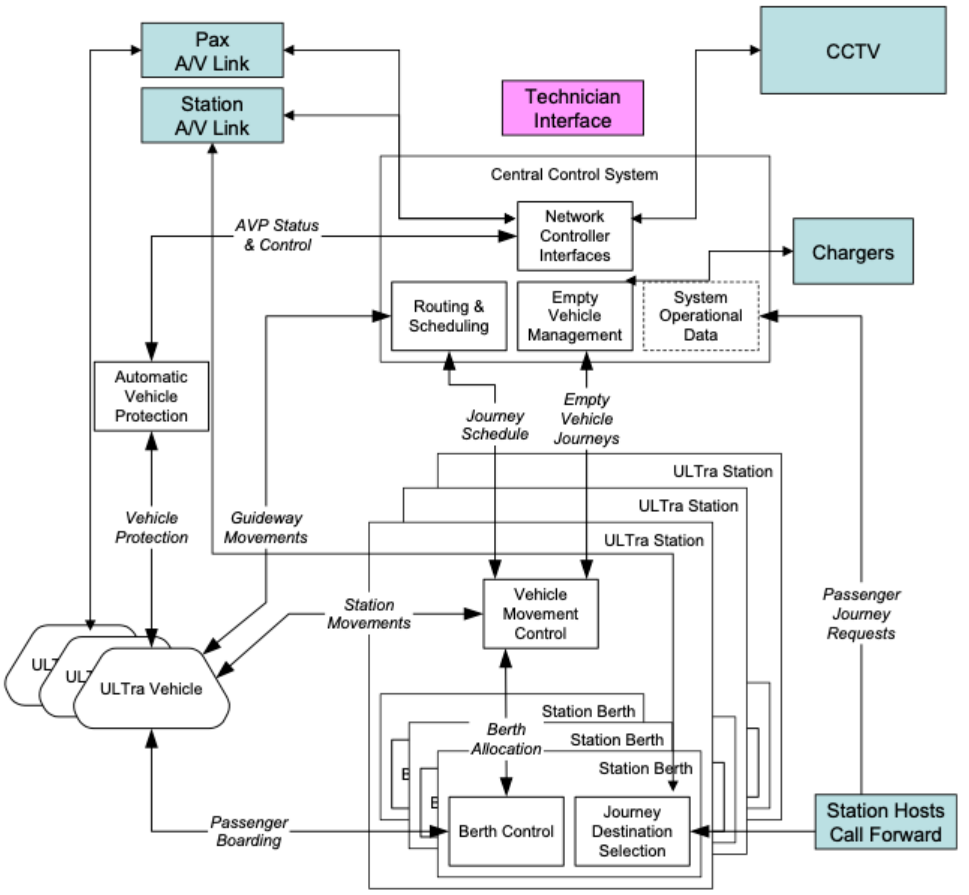
Vehicle control functions are managed by two hardware controllers. Vehicle guidance-navigation-and-control, health-use-and-monitoring, and other peripheral controls are developed in the MathWorks dSPACE development environment and implemented on a MicroAutoBox (MABx) rapid prototyping platform. Traction drive, motor brake control, and safety interlock functionality are implemented in Vehicle Control Language on an industrial motor control unit. Safety-critical functions, including automatic vehicle protection (AVP) and vehicle door control functions, are replicated on each hardware controller for dual redundancy. The AVP system is a fixed-block signaling system that ensures safe vehicle separation (it is similar to that used on railways and uses inductive coupling between sensors on the guideway and in vehicles). The central control system server runs AVP monitoring software to continually monitor the health of AVP sensors and critical functionality. The majority of system functionality is achieved with embedded system application software.

Control system applications comprise approximately 300 KSLOC (thousand lines of code). Project complexity has been viewed relative to other typical software projects in Figure 3. Technical complexity is regarded as high, given that control systems are custom built, unprecedented, embedded, real time, distributed, and fault tolerant. However, external interfaces are limited to four HMIs (for passengers, operators, and technicians), and the system does not interface with other systems in its operating environment (this is likely to change if the system is extended to other areas in the airport campus).

Management complexity is regarded as above average. There were on average 25 personnel in the engineering organization, five of which were software engineers. Project duration is high, with the system not yet in commercial operation. However, the number of stakeholders was limited to a single client (the client airport) and the safety verification team, reducing complexity in this respect. The client assigned a design-build-operate-maintain (DBOM) contract to the PRT system developer for the scheme at the beginning of the project. The subsequent program was conceived as a series of phases: development, demonstration, construction, installation, commissioning, confidence trials, initial operations, and then full operations. The PRT developer commenced further design and development to meet the expanded requirements of the full production system. Changes were also made at a remote test facility to allow testing to be more representative of the airport application. The idea was to develop much of the system's core capability at the test facility before moving to the target environment.

High-level planning, including user requirements definition, success criteria, and assurance processes, were necessarily plan driven. There was a clear and definite understanding of the customer's transit capability needs, which could not perceivably be met with a partially complete system, and a rigorous assurance process required comprehensive documentation to demonstrate adherence to process standards. Rigorous up-front planning was necessary for contract pricing and agreeing on payments contingent on successful completion of development milestones. Thus, in the beginning

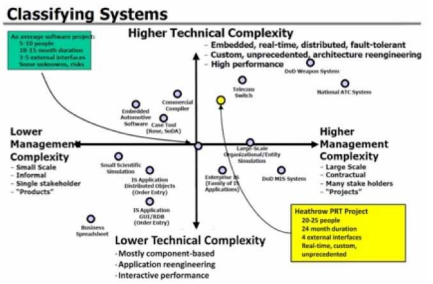
Figure 2. Airport PRT system architecture



system-level development was necessarily plan driven. Detailed plans were drawn up, identifying all major tasks, durations and work products, and these details became part of the contractual agreement between the airport and the developer who were required to issue management plans, including those relating to requirements management, configuration management, test management, information management, and system assurance. User requirements and success criteria set out functional requirements (e.g., vehicle interior, passenger interface), performance requirements (e.g., capacity, response time, availability), safety feature verification, regulation and standards compliance, and key system demonstration requirements. These requirements and success criteria evolved over time as both organizations' understanding of the system matured. In general, Agile methods cannot be used where there is a need for a large portion of system capability up front (Boehm, 2003). However, the use of early software versions that were loaded onto the vehicles and control systems, as well as the use of pure-simulation and hardware-in-the-loop (HIL) environments, enabled developers to demonstrate reduced system capability from an early stage of the project.

Plans and specifications were necessary for supporting an early safety case, integrating off-the-shelf components, and coordinating contractors and suppliers. Project planning was conducted centrally through one-to-one interaction between a dedicated planner and each person in the development team. The customer did not participate in planning or development at this level, but instead relied on the developer to demonstrate compliance and report progress against plan. Although system-level

Figure 3. Airport PRT perceived project complexity



development began with such thorough plans and specifications, the developers themselves did not use a lot of this information because they relied on group planning, tight collaboration, and shared tacit knowledge. Because of the novelty of the system, specifications were emergent and under continual adjustment. Most plans and specifications tended to be updated post process for assurance or compliance purposes. During implementation at the airport, plan-driven methods were unable to foresee lengthy integration processes, emergent system requirements, and emergent operational requirements; thus, the project suffered from schedule delays (Figure 4). Schedules were dominated by fault finding and fixing activities, test development activities, and supplier lead times. Emergent system requirements (in both hardware and software components) led to unforeseen developments, undermining the customer's understanding of progress. The project suffered from the application of plan-driven methods because over-specification of performance requirements early on led to confusion and renegotiation further down the line.

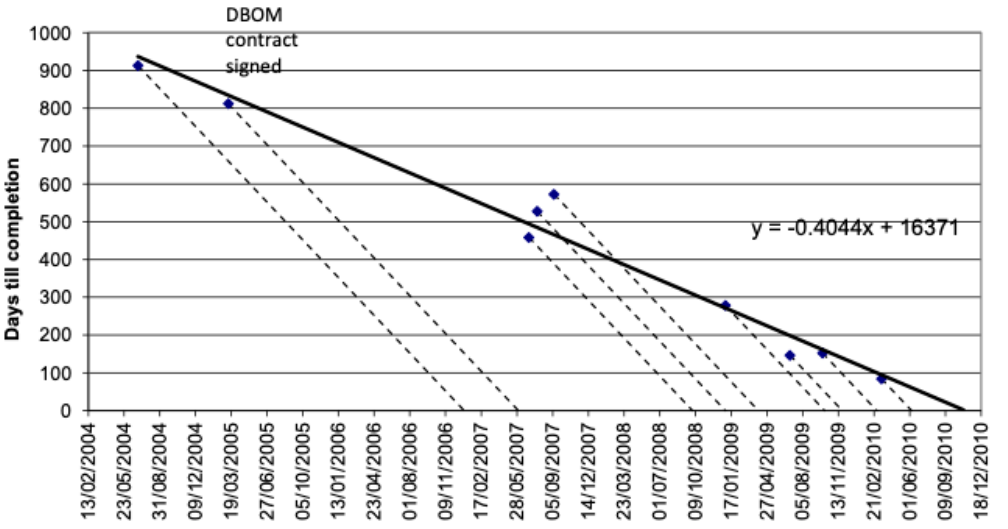
In practice, development continued well into later stages of the program. After various activities were migrated to the target environment, numerous emerging development and test activities undermined the customer's understanding of progress, even leading to growing concerns over the contractor's ability to deliver. The PRT developers then shifted their management approach to a more Agile one, using short-term planning and frequent and early releases to demonstrate progress and build confidence. This approach then exploited the software-intensive nature of development, with new functionality incorporated primarily through iterations of application software and with integration difficulties resolved effectively through the use of rapid cycles of development and automated testing in both pure-simulation and HIL environments. Deliberate group planning sessions with end-user participation allowed developers to operate on the basis of shared tacit knowledge, minimizing their documentation overhead. Evidently, a degree of agility was achieved in a variety of management, technical, and organizational respects.

A Critical Reflection of Application, Management, Technical, and Organizational Characteristics

The core system design, including the basic concept of operations, was established early in the project. This step incorporated a number of key safety features, such as unidirectional, segregated guideway, offline stations, small vehicles, a limited maximum speed, and an independent AVP. Vehicles use fail-safe electromagnetic "hold-off" braking systems and interlocks to safeguard passengers in moving vehicles. The guideway curbs retain vehicles and minimize the angle of incidence in the event of a steering failure. Features such as these together provided an inherent "layer" of safety, allowing higher flexibility in the subsequent development approach.

Safety was a key consideration from the start of the development of the system concept. The PRT had no safety process standards specific to it because this was a new form of transport system. As a result, developers devised their own process, which emphasized the need to incorporate safety

Figure 4. Shifting program completion estimates



in every stage of the project. An independent safety body—the Safety Verification Team (SVT)—was specified, reviewing the project with regard to safety and ensuring that this review would take place at well-defined milestones. This review process relied on a substantial amount of up-to-date and comprehensive safety documentation being made available to the SVT and client organization. These reviews had to be planned well in advance. The feedback from the reviews helped satisfy the end-client that the system was fulfilling all the aspirations and technicalities of the safety philosophy. Developers also specified that a safety case would be developed alongside the rest of the product. The main format of the safety case would remain unchanged as the project developed, but detail would be added as more information about the system became available.

Because the airport PRT was a novel system, no directly relevant safety statistics on which to base a safety assessment were available. Therefore, the developers adopted a diverse approach to the safety assessment that included a hazard analysis and risk assessment against risk criteria agreed with His Majesty's Railway Inspectorate (HMRI); assessments against the Railway Safety Principles and Guidance; assessment against the US Automated People Mover (APM) Standards ASCE 21; and a quantified risk assessment (QRA) allowing direct comparison with the safety targets agreed on between developers, the airport, and the SVT. Design standards such as IEC 61508 were followed to ensure robust design and safe development of safety critical functions. The SVT did not perform safety approval or assurance on the system; rather, the relevant assurance groups within the client organization completed this step. The key difference is that assurance groups looked at the documentation and process standard compliance, whereas the safety verification process focused on assessing the safety of the system based on evidence. Developers were making their own case to the SVT and worked with the SVT to "steer" the safety verification process. This evidence-based approach is considered as the reason the SVT was the focus of all safety-critical developments, and why a more flexible development approach was allowed in many of the subsystems. Furthermore, the use of the IEC 61508 standard in the development of the system promoted iterative life cycle approaches to manage the impact of frequent and late requirements changes on the safety case.

The project began with formally defined change control processes for requirements change, design review, configuration control, test procedure change, reporting and sign-off. Any changes related to the safety critical system (i.e., the AVP), or hardware changes expected to impact the program or budget were managed according to these processes. However, much of the control system

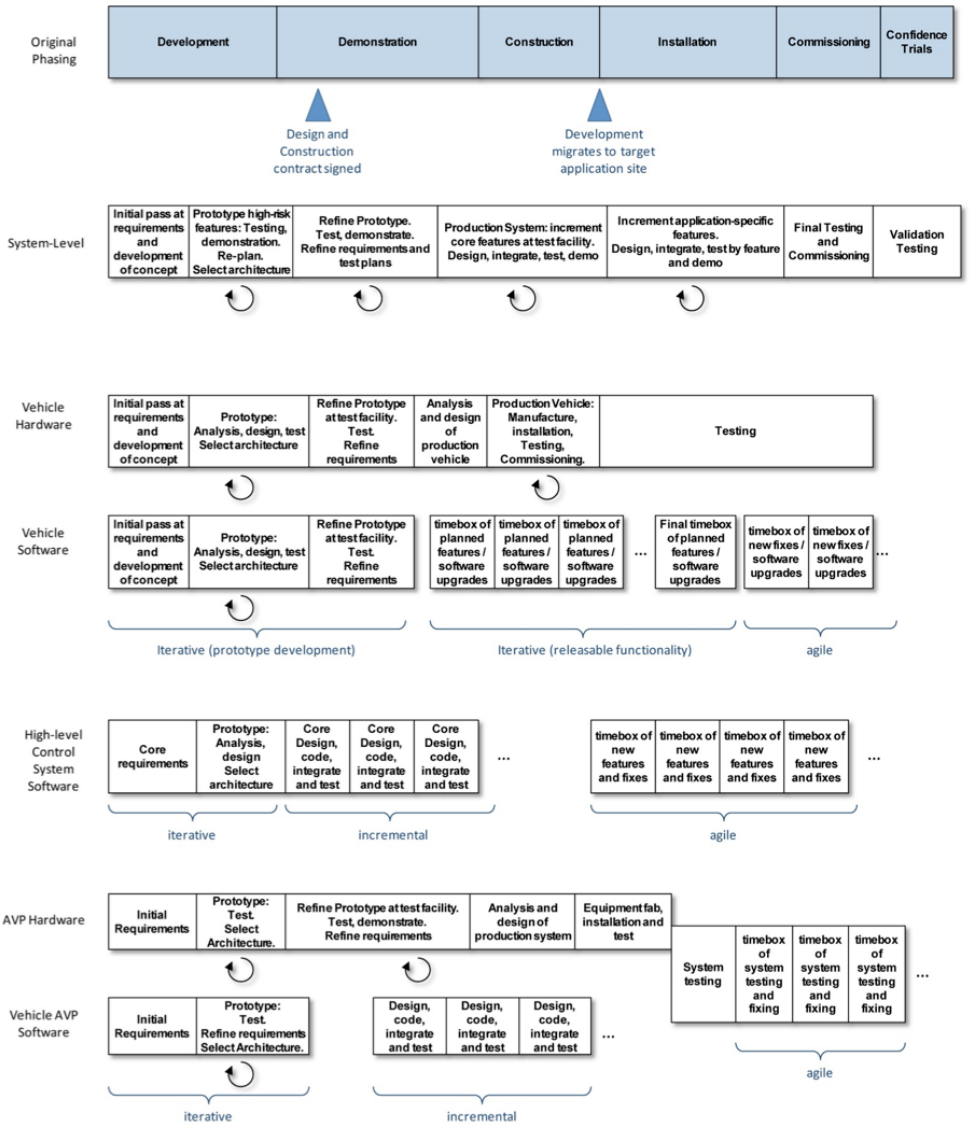
software used informal design reviews, together with configuration management and subversion repository functionality within their integrated design environments (IDEs). Furthermore, many of the requirements for control system functionality were codified in the system's test modules, so changes to requirements and test processes required minimal documentation overhead. With only one or two people developing each control system module, much of the software-related functionality could be changed quickly and with little process overhead.

The early development activities resembled the spiral and evolutionary prototyping life cycle models (Rothman, 2007), using risk-driven iterative prototyping and continual customer engagement to progressively establish requirements and objectives (Figure 5). The development of a prototype system at a remote test facility was used to garner an understanding of the key enabling systems, such as autonomous vehicle control systems and central control system functionality. Development to production standards involved several component and software upgrades, as well as many new features to enable final operations at the airport. After development activities had been migrated to the airport site, the remaining developments were planned by feature and in stages from interface testing to single-vehicle testing and then multiple-vehicle testing. Testing was also planned by feature. Although many of these features were preplanned, many low-risk features concerned with high-level control logic and HMIs were left until later in the process when their requirements would be better understood. Preplanned testing activities spawned new development and test activities. Incremental development approaches were replaced by more Agile development approaches, responding to newly emerging requirements, fault-fixing activities, and test development activities (Figure 5). Specifically, the vehicle software development life cycle used early prototyping to establish architecture and determine the final approach to automatic guidance, navigation and control (Figure 5). Other features, such as battery charge management and health, use, and monitoring (HUMS) functionality, were developed in iterations. Finally, Agile iterations/increments were used to fix and test features in response to faults emerging from system-level integration and testing activities.

The use of a more Agile planning approach assisted the incorporation of operational requirements late in the process. Rapid iteration cycles, group planning, and end-user participation allowed operations personnel to reconcile their needs with the emerging capability of the system, without the need for heavy documentation or process standards compliance. This approach exploited the use of graphical user interface (GUI) design tools. While managers focused on short-term planning as a means of adapting to unforeseeable change, it was also necessary to forecast and negotiate development milestones and generally avoid an "open-ended project." The client's reaction to this shift in management style confirmed their intentions for the project as a showcase for PRT technology around the world. The project was not time critical; however, it had to be shown to be under control. The managers' objective was to ensure that, once in operation, the system worked as intended and gave the best level of service possible to end-users of the system. Schedules were expected to slip, given the amount of uncertainty and risk associated with the novelty of the system. Progress metrics were supplemented with quality metrics, such as defects versus test cases (an Agile project management metric), to give the customer an idea of progress in ensuring a stable, working system throughout the integration process.

As seen in Figure 5, the development of high-level control system iterations finally became more Agile. Software development was planned in short iterations/increments to deal with unforeseen debugging and testing support. New requirements emerging from operational planning, including the control system's fault response and system recovery functionality, also could be easily incorporated into the workflow with the use of short planning cycles. These requirements emerged primarily through an Agile-like development process used for operator GUI software. Safety-critical systems, such as the AVP system, were prototyped to determine requirements and architecture up front. This aspect of system development was necessarily plan driven with lengthy design review processes that the organization wanted to avoid repeating if possible. However, final integration and testing activities were coordinated in a more Agile fashion, with rapid cycles of analysis, fixing, and testing activities, planned one iteration at a time. Automated test support was provided through AVP monitoring software.

Figure 5. Life cycle approaches observed during the case study PRT system development



The system architecture was designed to accommodate late decision-making, such as with the system's wireless communication system. This system was designed as a number of alternative solutions, each providing different levels of redundancy. Design decisions could be delayed until sufficient research had suggested the desired level with respect to the overall reliability needs of the system. Through "option-based design" serious rework was avoided. The core system architecture was designed to complement the small size of the development team and allow for a degree of agility in subsystem development. The system was divided into modules, with each module owned by a single developer or small team. The idea was that one person would be able to understand and manage everything in their module. Interfaces between modules were made as simple as possible and were tightly controlled to allow developers to work autonomously while ensuring that core subsystems

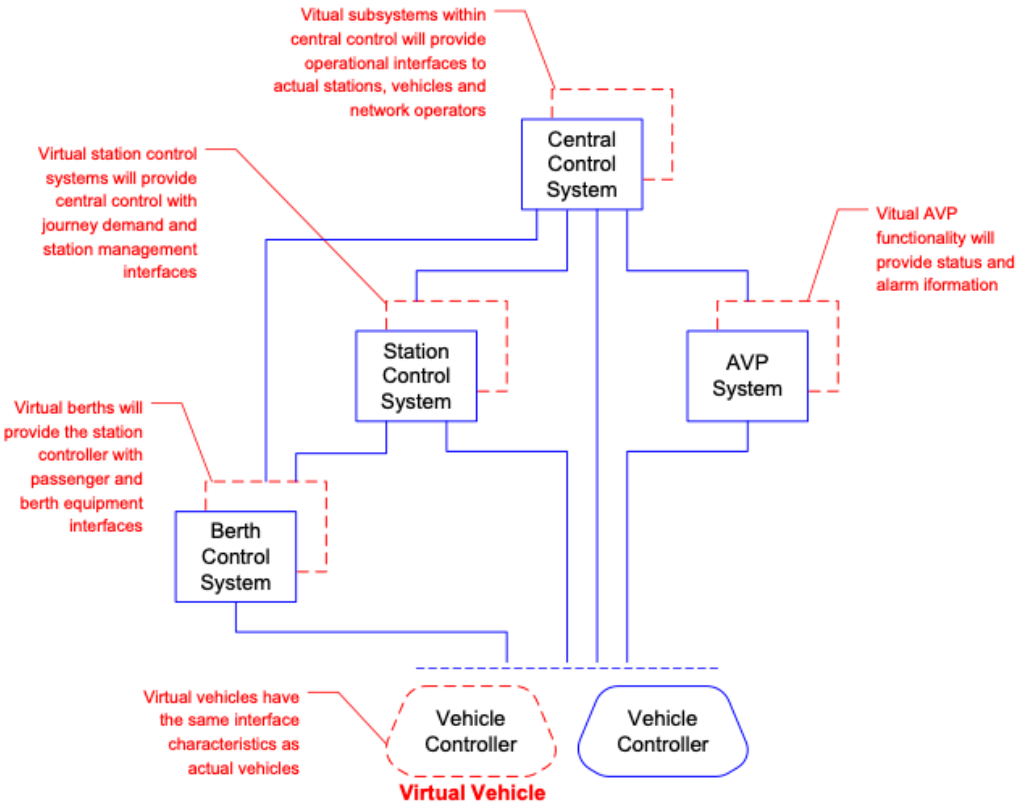
remained broadly compatible. Developers adapted their own processes in developing each subsystem quickly and efficiently and collaborated to ensure that interfaces remained stable. With the majority of developers based in different locations around the U.K., this also enabled developers to work remotely for relatively long periods of time. When subsystem development was slowed by efforts to manage system configuration for system-level integration testing, developers had to strike a balance between delaying system-level integration testing to allow for more efficient subsystem development and integrating subsystems early and often to provide continual verification of the system.

Early consideration of software in the design process allowed developers to better integrate commercial-off-the-shelf (COTS) technology and mitigate late-in-the-process compatibility issues. Software allowed developers to get around the problem of integrating COTS technology that was not designed specifically for use under PRT operating conditions. One such example involved the use of standard lead-acid batteries and battery chargers. These are conventionally used for recharging vehicles over a fixed amount of time, whereas PRT systems use them for fast, opportunistic charging. Although the supplier of charging devices could customize charger behavior to some degree, this approach was not sufficient for managing a battery charge in such a dynamic operating environment. Consequently, the majority of control was left to battery charge management algorithms in the vehicles' control software. The use of COTS-based hardware and software systems allowed developers to streamline their workforce and concentrate on core competencies. The focus was more toward software development, COTS supplier management, and system integration and testing, as opposed to low-level design, manufacture, and unit testing. This approach avoided numerous potential integration difficulties and lack of testing/fixing resources. The use of software-intensive COTS-based systems has provided a means of dealing with change with a small organization and relatively complex new system development.

The system's HMI drew developers and operating personnel into a process more akin to Agile methods. It initially served as a testing interface that had evolved with developers' testing needs. However, this was far from sufficient for the needs of operations personnel. Furthermore, the operations team had minimal knowledge of system capability on which to base their operating procedures, and thus, were unable to provide complete, clear requirements up front. Management initially assumed that there was sufficient basis for designing the full operations-oriented HMI in a single iteration. However, the managers soon recognized that an iterative and incremental development process was required to (1.) foster an understanding of system capability and the necessary degree of control required in the HMI from a developer's point of view; (2.) provide the operator with early versions to understand what they needed from the interface; and (3.) allay concerns of the customer, who had a bad experience with past HMI developments with early working versions. A new version of the software was released every month. In each iteration, a full cycle of operational analysis, planning, development, and testing was accomplished.

The majority of high-level control system functionality was developed in a completely simulated or HIL environment at the test facility. The same practice was used for integrating new features and new vehicles in the target environment at the airport. Simulated or "virtual" subsystems were used to provide a virtual environment in which key subsystems could be tested (Figure 6). For example, using virtual station controllers and virtual vehicles provided central control with a simulated environment that provided repeated automated testing under all envisaged operating scenarios. Then, real station controllers and real vehicles were introduced one by one to allow progressive integration and testing. Integrating and testing of core features of the system were performed early and often to identify errors not only in the implementation and specifications but also with the integration, testing, and validation processes themselves. Developers quickly learned that the tests used to validate operational capability, interoperability, and interface quality would evolve as fault finding/fixing revealed new situations to test for new emergent behaviors and more efficient ways to test for them. Therefore, formal system integration testing was supplemented with exploratory testing, fault finding, and fixing. Frequent and early integration was key to developing systems and practices to support testing.

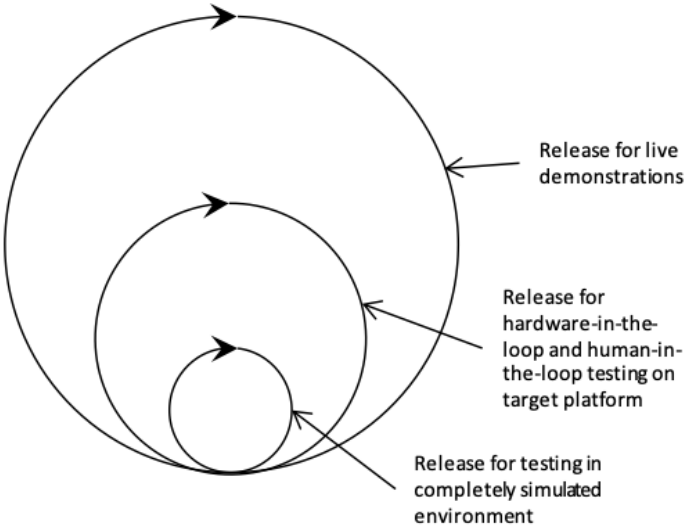
Figure 6. Use of HIL environments to support continuous integration and testing



The use of completely simulated and HIL environments with automated testing allowed control system developers to integrate and test on a frequent basis, including both the software and the hardware. Developers added new features/fixes with frequent (daily) cycles of build and test in a completely simulated environment. This step provided rapid feedback to developers, helping them to correct faults as they arose and allowing developers to quickly reprioritize features/fixes to support system integration activities. Less frequent, but more stable releases were used for HIL and human-in-the-loop development and testing on the target platform. Feedback cycles allowed test engineers to detect hardware-dependent faults early and subsequently re-task developers (or add to their feature/fix backlog). Periodic demonstration releases allowed the customer to examine progress made (Figure 7).

The customer's understanding of the system was critical to the airport PRT project. With insufficient understanding of the system from an operational point of view, the customer originally established performance requirements that in some cases were either contradictory or not verifiable with an operational system. Modeling and simulation techniques were used to develop both the customer's and operations team's understanding of the system and set more relevant validation criteria. The customer organization was inherently plan driven; thus, the developer's management had to adapt their approach to satisfy customer needs/concerns, while also managing the relatively turbulent development environment within. Immense frustration was found within the customer organization initially as development fell behind schedule, and new tasks and new tests appeared. These were in essence inherent properties of a novel system development for which the more R&D nature of the developer was suited. Developers of the core system were accustomed to an Agile culture in which they owned their own processes and coordinated development activities through

Figure 7. Continuous integration cycle, test release cycle, and demonstration release cycle



group planning with face-to-face communication. Therefore, when the management approach was shifted to a more Agile approach, each person was expected and trusted to do whatever work was necessary to the success of the system.

Most individual developers could be considered as Cockburn level 2 or Cockburn level 3 people (Boehm, 2003). Because of the level of expertise and experience among developers, the management approach was always a "light touch" with a focus on maintaining order rather than control. Managers focused on listening to developers so that they could provide better suggestions for improvements and report progress and issues more clearly to senior management/executive level. Managers entrusted developers with an overall directive and let the individuals prioritize tasks and manage their own day-to-day commitments. Developers had a variety of automotive and aerospace backgrounds. Their combined knowledge and experience allowed them to tailor development processes to their needs. With no direct PRT track record, they relied on the expertise and experience of personnel to instil trust in their ability to deliver. Developers operated on the basis of shared tacit knowledge Therefore, most of the knowledge surrounding the system and its development could not be accessed without engaging with all developers and working with them over a period of time. This process frustrated the customer to some degree, but the customer was never in any doubt regarding the developer's technical ability.

There could be endless further reflections upon a number of factors pertaining to application, management, technical, and organizational characteristics. We will, however, gather everything discussed so far, as well as other points, in Table 1, where using Boehm and Turner's framework (2003a) we make explicit the features of agility that enabled the delivery of the PRT project in a way that surpassed the challenges faced and how these were integrated with the original plan-based approach for the system.

How Agility Made a Difference

Plan-driven methods were necessary for developing safety-critical features and providing continual safety verification of the system. However, many aspects of the system were subject to ongoing iterative development with continuous integration and testing. Safety-critical aspects were bounded to make safety provable and uphold the system's safety case throughout development. From the beginning of the project, developers recognized that this project was a novel system requiring a new safety regulation and engaged in discussion with the regulator for its development. They worked with an

independent SVT to devise a safety verification method appropriate for safety assurance of the PRT system. Although this process relied on comprehensive documentation, it also relied on a degree of trust between them and the SVT that could exist only through sustained collaborative development.

The airport's approach to engaging with developers was inherently plan driven, using contracts as a basis for relations. Having validated critical aspects of the system through successful demonstrations, the customer was confident that it was possible to foresee the remaining issues and work through them in advance, formalizing plans and specifications into a contractual agreement. This was necessary for pricing the contract, agreeing on milestones and payments in advance, and avoiding an "open-ended" project. In practice, plan-driven methods were undermined from continually evolving requirements, developments, testing, and unforeseen changes (e.g., fault fixing) surrounding the various integration difficulties. Therefore, a more Agile approach to managing development activities was followed—one that focused planning efforts on particular testing/demonstration goals in the short-term, while using high-level plans to forecast and agree on key project milestones. This approach allowed for a degree of autonomy among developers and allowed operational requirements to be incorporated late in the process. The documented plans and specifications were subject to continual modification to a point where they could not be maintained given the limited resources. Evidently, developers tended to rely on tacit shared knowledge more than documented knowledge.

The developer company did not have the resources to expand; thus, it needed an architecture that would allow a small development team to manage the entire life cycle. Therefore, the core system architecture was designed around the needs of a small organization, allowing individuals to own a subsystem and own their own processes. Developers would be given sufficient autonomy with minimal process or documentation mandates. Interfaces were made extremely simple and were tightly controlled to allow for a degree of autonomy within each area of subsystem development. The decision was made early to minimize the amount of data passing through control system interfaces, ensuring that, while a degree of autonomy was given to each module developer, modules remained broadly compatible. A risk-driven approach to defining requirements introduced a further degree of agility in the development process. Safety requirements and requirements relating to the core enabling systems were defined from the beginning and were subject to rigorous change control. But for noncritical features such as human interfaces, the pervasive use of software throughout the system allowed requirements to be subsequently set and evolve as developers' understanding of the requirements matured.

Developers were intent on delaying formal system-level integration until as late as possible to avoid being slowed down by configuration management and other managerial constraints. Later in the project, management began to coordinate subsystem development to achieve frequent and early system-level integration and early system-level testing, allowing frequent demonstrations of reduced functionality to the customer. The ability to deliver in short iterations was well suited for the company's need for a risk-driven development approach, allowing developers to quickly understand the best technology approach and hardware/software architecture needed to support the system's envisaged life cycle.

The airport PRT project initially saw a clash between the developer's R&D-driven culture, which thrived on informal and emergent working practices, and the customer's plan-driven one, which depended on up-front planning and process maturity. The developer's management adapted their management style to cope with the inherent working practice, while ensuring that high level plans were established and maintained with the customer. Specifically, managers were forced to use short planning cycles, or a rolling-wave planning approach, to establish and prioritize developer's activities. Key integration milestones required cross-discipline, collocated teams performing fault-finding, and fault-fixing activities for particular features. When schedules began to slip owing to continually emerging development, integration, and testing activities, the focus turned to specific demonstrable milestones and customer participation in planning each demonstration requirement. This shift focused all development on demonstrable outcomes, which the customer could verify on sight/inspection.

Table 1. Analysis of the airport PRT case study project along the plan-driven versus agile methods characteristics comparative framework of Boehm and Turner (2003a)

Characteristics		Plan-Driven Method	Agile Method	Situation	Response	Impact
Application	Primary goals	Predictability, stability, high assurance	Rapid value; responding to change	Need for high assurance as well as a degree of flexibility to cope with unforeseen changes in a novel system development	Combination of proven off-the-shelf technology where possible, integrated and operated with in-house developed software	High assurance and high flexibility
	Criticality	Up-front architecture is necessary for proving safety. Safety assurance process requires rigorous up-front planning and adherence to process standards.	Safety can be achieved with refactoring to safety standards; however, safety may be compromised by subsequent changes. Continuous integration and "continuous certification" may be possible.	Transit system safety integrity needed to be upheld and provable despite design changes and adjustments occurring throughout system.	Architecture set early to support a safety case. Safety-critical functionality was bounded (i.e., the AVP system). Separate life cycle approach used for safety-critical subsystem.	Safety was provable and safety-critical functionality was safeguarded from design changes elsewhere in the system.
	Size	Large teams and projects	Small teams and projects	Small team of engineers/ developers—10 personnel, including five software developers	Architecture designed around the organization to allow for high autonomy in areas of subsystem development.	Minimized management and communication overhead, maximizing efficiency
	Environment	Required when there is a large portion of system capability needed up front.	Suited for systems that can provide business value to the customer in small increments/iterations	Full operational capability was not so important to the customer as early feature demonstration to manage risks and allay concerns.	Early demonstrations of reduced functionality	This approach was vital to early procurement and building confidence in the system as its development moved on-site.
		Up-front architecture necessary for managing interfaces with external systems	Few external interfaces to control	The system only interfaces with operations personnel and passengers. External interfaces are software-based HMIs.	Interfaces could be designed and developed late on in the process, with no external compatibility issues.	High adaptability to changes in operational requirements and customer preferences
		Need to plan for hardware manufacturer, civil contractors, off-the-shelf hardware suppliers	Short planning horizon used in Agile project management excludes contractors, manufacturers, suppliers.	Hardware/software system. Need for flexibility and rapid feedback in the software development process	Separate life cycle approaches for hardware and software	Hardware development risks managed separately from software development risks
		Need to plan for any hardware test and verification support systems	Relies on automated test and verification	Proven off-the-shelf technology avoids low-level hardware development and testing. PRT is fully automated.	Hardware and software testing was achieved at a higher level and could be achieved on-site and under automatic control.	Rapid cycle of development and testing was achieved for many hardware/ software systems.
		Organization spread across multiple locations	Collocated teams	Individuals/small teams developing different subsystems in different locations	Architecture designed to allow a degree of autonomy in subsystem development; however, integration events required all developers working together on-site.	Cross functional, cross-discipline team working together to support integration, test, and fix activities. Minimizes integration risk.
Management	Customer relations	Plans contractually agreed on up front	Iterations allow customers/end-users to actively establish and prioritize require-ments on the system.	Progress against plan suffered from emergent development, integra-tion, and test support activities	Focus turned to specific demonstrable milestones and customer participation in planning each demonstration requirement.	Progress and issues better understood and more visible

continued on following page

Table 1. Continued

Characteristics		Plan-Driven Method	Agile Method	Situation	Response	Impact
Management	Customer relations	Customer/ end-user not involved in the development process	Relies on customer/end-user participation	Customer's lack of understanding surrounding development issues	Customer representative brought in to help steer the planning process	Customer representative could understand the challenges in more detail and mediate between PRT developer and the customer.
		Process maturity as a means of instilling trust	Experience, expertise, and shared track record as a means of instilling trust	Small start-up and novel system; therefore, little process maturity	Customer relied on experience and expertise of developers. PRT developers focused on building a track record with the customer.	The project became as much a confi-dence and trust-building exercise as it was a system development exercise.
	Planning and control	Relies on documented plans, managed centrally	Relies on group planning and tacit interpersonal knowledge	Schedules and specifications required continual rework, as new activities continually emerged	Developers used documentation only where absolutely necessary to their progression. Management maintained higher-level plans.	Minimized documentation rework while maintaining a level of progress reporting
		Long-term planning	Plan only for the next short iteration (or use rolling-wave planning)	Emergent developments, fault-finding and fault-fixing activities undermine the schedule	Planning switched to a rolling-wave approach, with detailed planning for the next iteration and high-level plans for subsequent iterations.	Ensures that development activities are trained on a particular feature or outcome, and that related integration issues are dealt with before the next iteration. Emergent developments can be incorporated into subsequent development iterations.
		Progress measured as "progress against plan"	Progress is measured by how many requirements are turned into capability, or simply how satisfied the customer is.	Progress against plan was not seen to reflect "real progress."	Detailed planning was focused on the next demonstration milestone, marking progress toward the original user requirements specification.	Successful completion of milestones gained confidence in the plans.
		Quality is assured through adherence to process standards.	Quality is assured via a continually working system.	Lean documentation approach delayed the assurance process.	Focus shifted toward empirical reliability, proven through continual running of the system throughout testing and commissioning.	Continual assur-ance provided through regular demonstrations of a working system.
Technical	Requirements	Up-front complete, consistent, traceable requirements	Rapid iteration cycles used to establish and prioritize new requirements with the customer	High-level transit capability understood by the customer. System requirements were only understood on a high level at the outset.	Risk-driven, evolutionary requirements. Iterative prototyping used to select an appropriate architecture	Hardware design and safety architecture defined early. Software development used Agile approaches to manage late changes, debug, and test activities.
	Development	Up-front architecture to support the envisaged life cycle of the system	Simplest design given the current requirement set; requirements too unpredictable to plan for in the current design	Developers could foresee many late design decisions, component upgrades, and also aspects that needed to remain flexible (e.g., vehicle equipment configuration was expected to change; vehicle control software development was evolutionary)	Up-front architecture to support redesign, upgrades, and extension of the system. Option-based design was used to delay decision-making where possible. Architecture designed to allow flexible development at subsystem level through the use of simple subsystem interfaces and strict interface control.	COTS component upgrades supported. Design decisions feasibly delayed until requirements were better under-stood. Adaptive processes and rapid develop-ment at subsystem level.

continued on following page

Table 1. Continued

Characteristics		Plan-Driven Method	Agile Method	Situation	Response	Impact
Technical	Development	Serial, iterative, and incremental life cycles well suited for managing hardware devel-opment and safety-critical func-tionality	Agile life cycle approach well suited to noncritical software	Different levels of criticality across hardware and software components, including a safety-critical hardware/software system.	Separate life cycle approaches used for hardware and software components of each subsystem. A combination of life cycles was used in most instances (e.g., iterative, incremental, and then Agile).	Life cycle approach commensurate with the risks associated with each subsys-tem. Multiple life cycle approaches required a signi-ficant program management effort.
		Architecture designed to accommodate possible redesign/ refit	Flexibility of software allows for unforeseen design changes.	Unforeseen hardware/ software integration issues	Software was used to overcome late COTS compatibility issues (e.g., vehicle battery charger).	Flexibility to support unforeseen hardware/ software integration issues
	Integration and testing	Late-in-the-process integration, or use of prototypes to rehearse the integration and testing process	Continuous integration and test-driven development	Distributed autonomous control system being developed and integrated with control system and communications equipment	Continuous integration of vehicle control system functionality with early hardware prototypes. Continuous inte-gration of control system functionality in a completely simulated envi-ronment. Progressive integration and testing on target platform and in target environment using HIL simulation.	Early progress before target hardware was available and before target environment was ready. Agility in the software development process to support the emerging needs of system integration activities.
Organizational	Customers	Up-front planning with the customer allows customer to continue to operate in their own organization.	Dedicated, representative, committed customer to steer development	Customer was not involved in the development process and did not have expertise with PRT systems.	Customer worked to manage interfaces between relevant work groups. Expert consultants, both from the customer organization and external, were used to support decision-making where the client organization lacked expertise/experience.	Customer's interests were fully represented in all management and technical areas.
	Developers	Can manage the organization around system architecture, allowing for less experienced/ skilled people to contribute	Richer mix of higher-skilled people	PRT developer had to work with a minimal number of personnel throughout the project	Every developer brought a wide range of skill and experience and had to manage a complete subsystem. There was no room for less skilled or inexperienced people.	The project was at risk from personnel leaving the project, or underperforming.
	Culture	Clear policies and procedures that define people's roles in the enterprise. People are expected to perform their roles.	Adaptive processes and self-organizing teams. People are expected and trusted to do whatever work is necessary to the success of the project	Emerging number of development and managerial tasks required personnel to assume several roles and responsibilities.	Roles were initially defined, with clear responsibilities. Subsequently, people took on several roles and began to self-organize. People were trusted to do what it takes to successfully complete the project.	Multitasking tended to disrupt the Agile approach because personnel were never fully committed to a single activity.

This approach reduced reliance on documented plans and instead focused on demonstrations of a working system. Collocation, deliberate group planning, and client participation were all seen as characteristics of a more Agile project management process. Although this was true, managers also had to maintain a sense of progress with the client, focusing on high-level planning for the long term and using demonstrations to convey progress and instill trust.

DISCUSSION

This work presented a case in which a degree of agility was crucial to the successful development and delivery of an airport PRT project. Agility was needed to cope with the following challenges:

(a.) emergent risks owing to the novelty of the system development; (b.) the need for rapid customer value, given the level of uncertainty surrounding the system's technical and commercial feasibility; (c.) the small size of the organization and limited overhead to support heavily documented processes; and (d.) the need for quality assurance despite the lack of process maturity. We used previous experience, together with the Agile concepts developed from the literature, as a basis for inquiry into the development processes before, during, and after our involvement in the project over a period of 12 months. This explored the need for plan-driven approaches as well as the need for a degree of agility in both technical and management respects, and suggested, for example, the reasons for the shift in management style, the need for frequent and early integration as a means of managing risk, and how the system was designed to promote late flexibility and Agile approaches in its core subsystems. This inquiry has contributed to ideas on how Agile methods may significantly enhance first-time system developments. Thus, a key contribution of this paper is in the understanding of the mechanisms in support for, and benefits of, agility in first-time systems development.

The concept of the system that was delivered emphasized software as a means of achieving the majority of system capability. Therefore, the system naturally facilitated a necessary degree of agility in the development process. The pervasive use of embedded systems in the project allowed software to manage everything from automatic door control to vehicle guidance and navigation to automatic routing, scheduling, and autonomous PRT network control. Flexibility was created in many of these systems with the use of standard highly programmable computing platforms such as the Microsoft .NET framework and the dSPACE MicroAutoBox rapid prototyping platform. The company minimized the need for hardware support with the use of proven off-the-shelf technologies where possible and by reducing mechanical complexity with the use of rubber-tired vehicles running on simple open guideways (Figure 8). These properties together allowed for early and rapid iterations of a prototype system, with system capability delivered primarily through iterations of application software development.

Another key aspect of the developed system is that its operational capability can be evolved through software developments made during commercial operation. Initially, the system could run a

Figure 8. PRT Podcar on guideway system
(Original photo courtesy of D. Rhodes, 2009)

simple service with a limited number of vehicles (i.e., three), offering the possibility of immediate commercial service. Through iterations of control system software, together with improved HMIs and operational strategies, more and more vehicles were added until the system was capable of operating at full capacity. Given that the airport system was a showcase for PRT, a minimum level of capability was required before the system was opened for commercial operations. We believe that this feature will become important for PRT networks of increased size and complexity, where initial commercial operations can steer developments to, for example, the control systems, station call-forward systems, and operating procedures, all funded by operating revenue. This evolutionary approach remains exclusive to non-safety-critical functionality, where Agile approaches to verification and validation, and change control/release are acceptable. Developments could allow the system to learn commuter behaviors, integrate with other transport networks' scheduling systems, integrate with mobile applications, switch to scheduled operations during the peaks (and demand-responsive operations off-peak), dynamically reroute vehicles around blocked guideway links, prioritize vehicle availability to particular stations depending on service-level agreements, or isolate vehicles from the rest of the fleet and crowded stations if they are a security threat. The ability to evolve transport capability in this way, with no disruption to commercial operations, distinguishes PRT from other transport systems.

Agile development is promoted by the studied PRT system through its use of repeatable design. Vehicle and station control systems, as well as berth control modules were developed as a single module and then installed on each respective system. Furthermore, each of these systems could be installed remotely over the wired network (for station and berth control) or over the wireless network (for vehicle control). This method allowed the development and testing of new module versions in real time, allowing for rapid iterations of coding and testing in the target environment. Much of the system's capability could be developed and tested in pure-simulation or HIL environments, allowing developers to move toward continuous integration and test-driven development. A pure-simulation platform was used for integrating new capability in short iterations, switching between periods of development and stabilization, and building up an automated test suite that would subsequently be used for HIL testing. HIL simulation was used to provide an effective test platform on which modules could be independently and rigorously tested, with a mixture of simulated or real subsystems in the loop. Test engineers could test before vehicles or other hardware became available, integrate hardware progressively to reduce cycle times, and manage integration difficulties more effectively. Short planning cycles also allowed integration testing activities to quickly feed back into the software development cycle, allowing developers to quickly respond to hardware-dependent faults, new requirements, and test support needs. This process became especially important as migration to the target environment incurred new developments, most of which could be absorbed into the existing architecture, with some requiring a degree of refactoring. This approach also assisted the incorporation of operational requirements that emerged with the addition of peripherals and user interfaces. The use of human-in-the-loop simulation assisted the development of operator interfaces, allowing operations personnel to develop operational procedures and the interfaces concurrently.

Agile methods rely on maintaining a continually working version of the system and using demonstrations to the client to convey progress and quality. This approach relies on the client having sufficient understanding of the system's capability, which in the case of some systems (e.g., avionics systems), would be rarely seen. More intangible systems require more formal compliance processes to convey progress and quality. The studied PRT system allows the level of functionality, performance, subsystem interoperability, and total operational capability to be easily deduced upon visual inspection, test participation, and user trials. This is perhaps why the project converged on demonstration milestones as a means of measuring progress.

Documented process plans and specifications were required to satisfy safety requirements and for quality assurance purposes. Safety, quality, risk, configuration, and test management plans were required early in the project. Safety assurance was critical to gaining regulatory approval and

satisfying the customer's safety requirements. However, for many of the subsystems, there was little overhead to support the continual adjustment and modification of documentation given the limited number of personnel, the number of emergent development activities, and increasing time pressures. Documentation tended to be updated periodically, or post-process, and was usually aimed specifically at assurance activities. The dependence on documentation was further reduced by the use of automated test environments, allowing new requirements/test cases to be maintained in executable form. Developers evidently focused on a continually working system as a means of assuring quality.

Although the relevance of Agile principles and practices was realized only late in the project, the organization had naturally converged on many of the same ideas. However, it is believed that making the use of Agile principles and practices explicit from the outset would improve project performance. The organization could have further exploited the use of automated test environments, driving continuous integration and test-driven development. Managers could have driven development by feature, allowing for more effective testing, more visible progress, and improved forecasting. An awareness of Agile principles could have shifted the focus of quality assurance teams, placing trust in a continually working system as opposed to compliance with documentation-heavy assurance processes. Finally, the use of Agile principles could have prompted the client to view progress in terms of their goals and immediate values, allowing managers to set out a more progressive, yet flexible delivery approach, focused on delivering value to the customer early and often.

CONCLUSION

This paper presented a case study where a degree of agility was crucial to the successful development and delivery of a novel safety-critical system in a regulated environment. We examined the extent to which Agile methods principles were used to enhance the delivery of an airport PRT project, which provides evidence in support of Turner's thesis on SE projects that suffer from "schedule-busting" integration processes (2007). The increasing involvement of software allows for increasing agility, as systems become much more software intensive. The project used option-based design for a number of subsystems, allowing design decisions to be better informed through better research/quality of information. The discussed PRT project used multiple life cycle approaches for different subsystems/modules in the way motivated by Rothman (2007), managing hardware and software development risks separately, with Agile software development and iterative prototyping of hardware. Early software releases on hardware prototypes were used to ensure that the system could be developed by feature as early as possible. There are strong parallels with Grenning's approach to embedded Agile development (2004), where quick releases are combined with more stable releases to tackle software bugs and hardware compatibility issues while minimizing resources/costs. The project relied on pure-simulation and HIL environments to allow for early release and more exhaustive testing. This technique is already used extensively in the automotive industry, and in addition, PRT can also benefit greatly from HIL simulation, allowing operators to train and realize operational requirements along the way.

Because the original contractual relationship between the PRT developer and the client suffered when schedules and deliverables slipped, an Agile project management approach was needed to deal with the various nested life cycle approaches within the development team. This ties in with Smith's reflections on Boeing 777 and their use of "rolling-wave" and "loose-tight" management approaches (2007). The lack of success in shifting the client's plan-driven attitude confirms Boehm's thinking on the degree of trust required to facilitate more Agile project management approaches (2003). In terms of the conflict of Agile methods with safety requirements, the airport PRT concept has inherent safety built in. This has allowed for more flexibility to deal with the relatively turbulent development process. Safety integrity was required in much embedded software. However, a long development period with continual demonstrations of a working system provided the necessary confidence in some critical aspects, such as the vehicle guidance-navigation-and-control system.

This study has identified properties of a novel automated transit system that promote the use of principles and practices from Agile development and Agile project management. A degree of agility has been seen to occur naturally; however, explicit consideration of Agile principles from the outset of a project led to significant increases in project performance in terms of the level of value perceived by the customer. We hope that this study prompts other real-world projects where new systems development have benefited from the use of Agile principles and practices, especially where convention dictates an entirely plan-driven approach.

REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. arXiv:1709.08439 [cs.SE]. https://doi.org/10.48550/arXiv.1709.08439

Augustine, S., Payne, B., Sencindiver, F., & Woodcock, S. (2005). Agile project management: Steering from the edges. *Communications of the ACM*, *48*(12), 85–89. doi:10.1145/1101779.1101781

Baskerville, R., Pries-Heje, J., & Madsen, S. (2011). Post-agility: What follows a decade of agility? *Information and Software Technology*, *53*(5), 543–555. doi:10.1016/j.infsof.2010.10.010

Bedoll, R. (2003). A tail [*sic*] of two projects: How 'Agile' methods succeeded after 'traditional' methods had failed in a critical system-development project. In F. Maurer & D. Wells (Eds.), *Extreme programming and Agile methods - XP/Agile universe 2003. Third XP and Second Agile Universe Conference.* Springer. doi:10.1007/978-3-540-45122-8_4

Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64–69. doi:10.1109/2.976920

Boehm, B. (2005). Some future trends and implications for systems and software engineering processes. *Systems Engineering*, *9*(1), 1–19. doi:10.1002/sys.20044

Boehm, B. (2006). *Product and process architectures for integrating Agile and plan-driven methods* [Keynote address]. *7th International Conference on Extreme Programming and Agile Processes in Software Engineering*, Oulu, Finland.

Boehm, B. (2013). Skating to where the puck is going: Future systems and software engineering opportunities and challenges. In J. Münch & K. Schmid (Eds.), Perspectives on the future of software engineering: Essays in honor of Dieter Rombach (pp. 299–333). Springer. doi:10.1007/978-3-642-37395-4_19

Boehm, B., Lane, J. A., Koolmanojwong, S., & Turner, R. (2010). Architected agile solutions for software-reliant systems. In T. Dingsøyr, T. Dybå, & N. Moe (Eds.), *Agile software development* (pp. 165–184). Springer., doi:10.1007/978-3-642-12575-1_8

Boehm, B., & Turner, R. (2003a). *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional. https://www.oreilly.com/library/view/balancing-agility-and/0321186125/

Boehm, B., & Turner, R. (2003b). Using risk to balance agile and plan-driven methods. *Computer*, *36*(6), 57–66. doi:10.1109/MC.2003.1204376

Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*, *22*(5), 30–39. doi:10.1109/MS.2005.129

Cockburn, A. (2001). *Agile software development*. Addison-Wesley Professional.

Cockburn, A. (2008). Using both incremental and iterative development. *Crosstalk*.

CollabNet VersionOne [Digital a.i]. (2018). *12th annual state of Agile report*. Digital a.i. https://digital.ai/ https://explore.versionone.com/state-of-agile/versionone-12th- annual-state-of-agile-report-2.

Fontana, R. M., Fontana, I. M., da Rosa Garbuio, P. A., Reinehr, S., & Malucelli, A. (2014). Processes versus people: How should agile software development maturity be defined? *Journal of Systems and Software*, *97*, 140–155. doi:10.1016/j.jss.2014.07.030

Goode, H. H., & Machol, R. E. (1957). *Systems engineering: An introduction to the design of large-scale systems*. McGraw-Hill.

Górski, J., & Łukasiewicz, K. (2013). Towards agile development of critical software. In A. Gorbenko, A. Romanovsky, & V. Kharchenko (Eds.), *5th International Workshop, Software Engineering for Resilient Systems, SERENE 2013. Lecture Notes in Computer Science, 8166*, (pp. 48–55). Springer. doi:10.1007/978-3-642-40894-6_4

Grenning, J. (2004). Progress before hardware. *The Agile Times*, *4*(1), 74-79. https://wingman-sw.com/papers/AgileAllianceNewsletterVol4.pdf

Hall, A. D. (1962). *A methodology for systems engineering*. van Nostrand.

Hallberg, N., Andersson, R., & Ölvander, C. (2010). Agile architecture framework for model driven development of C^2 systems. *Systems Engineering*, *13*(2), 175–185. doi:10.1002/sys.20141

Highsmith, J. (2002). *Agile software development ecosystems*. Addison-Wesley Professional.

Hugos, M. H. (2009). *Business agility: Sustainable prosperity in a relentlessly competitive world*. J Wiley & Sons.

Jonsson, H., Larsson, S., & Punnekkat, S. (2012). Agile practices in regulated railway software development. In *Proceedings of 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW)*, (pp. 355–360). IEEE. doi:10.1109/ISSREW.2012.80

Karlstrom, D., & Runeson, P. (2006). Integrating agile software development into stage-gate managed product development. *Empirical Software Engineering*, *11*(2), 203–225. doi:10.1007/s10664-006-6402-8

Kumar, P. S. (2019). PSK method for solving mixed and type-4 intuitionistic fuzzy solid transportation problems. [IJORIS]. *International Journal of Operations Research and Information Systems*, *10*(2), 20–53. doi:10.4018/IJORIS.2019040102

Kumar, R., Maheshwary, P., & Malche, T. (2019). Inside agile family: Software development methodologies. *International Journal on Computer Science and Engineering*, *7*(6), 650–660. doi:10.26438/ijcse/v7i6.650660

Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, *36*(6), 47–56. doi:10.1109/MC.2003.1204375

Mansoor, M., Khan, M. W., Rizvi, S. S. H., Hashmani, M. A., & Zubair, M. (2019). Adaptation of modern agile practices in global software engineering. In M. Rehman, A. Amin, A. R. Gilal, & M. A. Hashmani (Eds.), *Human factors in global software engineering* (pp. 164–187). IGI Global. doi:10.4018/978-1-5225-9448-2.ch007

Marshall, V. W. (1999). Reasoning with case studies: Issues of an aging workforce. *Journal of Aging Studies*, *13*(4), 377–389. doi:10.1016/S0890-4065(99)00016-X

Millard, W. D., Johnson, D. M., Henderson, J. M., Lombardo, N. J., Bass, R. B., & Smith, J. E. (2014). Embedding agile practices within a plan-driven hierarchical project life cycle. In *Proceedings of INCOSE International Symposium*, 745–758. Pacific Northwest National Lab. (PNNL). https://www.osti.gov/biblio/1194328

Mostashari, A., McComb, S. A., Kennedy, D. M., Cloutier, R., & Korfiatis, P. (2012). Developing a stakeholder-assisted agile CONOPS development process. *Systems Engineering*, *15*(1), 1–13. doi:10.1002/sys.20190

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, *48*(5), 72–78. doi:10.1145/1060710.1060712

Ramesh, B., Cao, L., & Baskerville, R. (2010). Agile requirements engineering practices and challenges: An empirical study. *Information Systems Journal*, *20*(5), 449–480. doi:10.1111/j.1365-2575.2007.00259.x

Rothman, J. (2007). *Manage it! Your guide to modern, pragmatic project management*. Pragmatic Bookshelf.

Schlager, K. J. (1956). Systems engineering—Key to modern development. *IRE Transactions on Engineering Management*, *EM-3*(3), 64–66. doi:10.1109/IRET-EM.1956.5007383

Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft Press.

Smith, P. G. (2007). *Flexible product development: Building agility for changing markets*. Jossey-Bass.

Smith, P. G. (2009). Flexible product development for a turbulent world—Is "Agile" NPD the answer? *PDMA Visions Magazine, 33*(2), 20–21. https://www.strategy2market.com/wp-content/uploads/2014/05/Flexible-Product-Development-Turbulent-World.pdf

Stevens, R., Brook, P., Jackson, K., & Arnold, S. (1998). *Systems engineering: Coping with complexity*. Prentice Hall.

Turner, R. (2007). Toward agile systems engineering processes. *Crosstalk*, 11–15. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b807d8efe4acd265187a98357b19c063b5a9ad5e

Turner, R., & Jain, A. (2002). Agile meets CMMI: Culture clash or common cause? (2002). In D. Wells & L. Williams (Eds.), Lecture Notes in Computer Science: Vol. 2418. *Extreme programming and agile methods—XP/Agile universe 2002* (pp. 153–165). Springer. doi:10.1007/3-540-45672-4_15

van Waardenburg, G., & van Vliet, H. (2013). When agile meets the enterprise. *Information and Software Technology*, *55*(12), 2154–2171. doi:10.1016/j.infsof.2013.07.012

Vinekar, V., Slinkman, C. W., & Nerur, S. (2006). Can agile and traditional systems development approaches coexist? An ambidextrous view. *Information Systems Management*, *23*(3), 31–42. doi:10.1201/1078.10580530/46108.23.3.20060601/93705.4

West, D., & Grant, T. (2010). *Agile development: Mainstream adoption has changed agility—Trends in real-world adoption of agile methods*. Forrester Research. https://www.forrester.com/report/agile-development-mainstream-adoption-has-changed-agility/RES56100

Nicholas Davenport is a strategy and operations consultant at Deloitte, U.K. He holds an engineering doctorate from the University of Bristol and has long worked on infrastructure projects, including intelligent transport systems and autonomous vehicles.

Theo Tryfonas is a chair of infrastructure systems and urban innovation at the Department of Civil Engineering, University of Bristol, U.K. He is a computer scientist with expertise in urban data, the IoT, and smart cities. He is a chartered fellow of the BCS, The Chartered Institute for IT, and a fellow of the Royal Society for Arts, Manufactures and Commerce (RSA).

Alan Peters is the ecosystem director at Connected Places Catapult. He is an intelligent mobility specialist and holds an engineering doctorate in systems from the University of Bristol. He has extensive experience working with emerging technologies in mobility and infrastructure sectors.

Stylianos Karatzas has received a diploma in electrical engineering and computer technology and a master of engineering in automation and control systems (School of Engineering, University of Patras). He holds a master of sciences in operations management (University of Bath, U.K). He has received a PhD from the Civil Engineering Department, University of Patras, in the field of smart cities infrastructure. He has worked as operations and project manager in a variety of construction projects in Greece and abroad. He is currently a research fellow at the University of Cambridge, Engineering Department, and coordinator of a number of European research projects.

Anastasios Karameros is a graduate civil engineer from the Department of Civil Engineering, University of Patras, and holds a master of sciences from the same institution. He is also a PhD candidate working on energy systems and their nexus with mobility and the built environment. He is a senior associate at PwC Greece with responsibilities for EU affairs and funding.