

Teaching How to Select an Optimal Agile, Plan-Driven, or Hybrid Software Development Approach: Lessons from Enterprise Software Development Leaders

Gary Spurrier and Heikki Topi

Recommended Citation: Spurrier, G., & Topi, H. (2023). Teaching How to Select an Optimal Agile, Plan-Driven, or Hybrid Software Development Approach: Lessons from Enterprise Software Development Leaders. *Journal of Information Systems Education*, 34(2), 148-178.

Article Link: <https://jise.org/Volume34/n2/JISE2023v34n2pp148-178.html>

Received: January 3, 2022
Revised: March 14, 2022
Accepted: July 14, 2022
Published: June 15, 2023

Find archived papers, submission instructions, terms of use, and much more at the JISE website:
<https://jise.org>

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Teaching How to Select an Optimal Agile, Plan-Driven, or Hybrid Software Development Approach: Lessons from Enterprise Software Development Leaders

Gary Spurrier

Heikki Topi

Department of Computer Information Systems

Bentley University

Waltham, MA 02452, USA

gary.spurrier@gmail.com, htopi@bentley.edu

ABSTRACT

Over 20 years after introducing and popularizing agile software development methods, those methods have proven effective in delivering projects that meet agile assumptions. Those assumptions require that projects be small and simple in scope and utilize small, colocated teams. Given this success, many agile advocates argue that agile should replace plan-driven methods in most or all project contexts, including those projects that deviate significantly from agile assumptions. However, today's reality is that a diversity of agile, plan-driven, and hybrid approaches continue to be widely used, with many individual organizations using multiple approaches across different projects. Furthermore, while agile advocates argue that the primary barrier to agile adoption is the inertia of traditional organizational cultures, there are, in fact, many rational motivations for utilizing plan-driven and hybrid methods based on individual project characteristics. For information systems students, this creates confusion in two ways: 1) understanding that there is no single best way to develop software in all circumstances but, rather, teams should choose an optimal project approach based on project characteristics, and 2) unpacking and analyzing the wide range of project characteristics – including multiple dimensions in functional requirements, non-functional requirements (NFRs), and team characteristics – that impact that choice. This paper addresses both sources of confusion by utilizing case studies from 22 interviews of enterprise software development leaders. The paper analyzes each case utilizing a “home grounds” model that graphically portrays key project characteristics and their impact on the optimal choice of software development project approach.

Keywords: Enterprise systems development, Agile, System development life cycle (SDLC), Case study

1. INTRODUCTION

It is now well established that agile software development methods can effectively deliver successful project outcomes. Indeed, these outcomes often can be superior to those from traditional, plan-driven methods, such as the traditional Software Development Life Cycle (SDLC) or “waterfall” (Hastie & Wojewoda, 2015; The Standish Group, 2015). This is especially true in projects where key agile assumptions are met – most importantly, software with a small, simple scope and is supported by small, cohesive, and colocated IT and business team members. Traditionally, these are often low criticality projects, executed in highly dynamic environments, with organizational cultures thriving on rapid, highly responsive delivery of value (Boehm & Turner, 2004; Williams & Cockburn, 2003).

As such, it makes sense that, since their introduction in the 1990s, agile software development methods such as eXtreme Programming (XP) (Beck, 1999, 2000), Scrum (Schwaber, 1995), and many others (Leffingwell, 2007) have come into widespread use. However, agile's popularity does not mean it has been universally adopted. For example, a major, global, multi-industry survey of agile software development practices

(Digital.ai, 2021) found that 98% of respondents report using agile practices in at least some of their organizations' teams. Importantly, however, that does not mean all teams in these organizations are agile. Rather, that same survey found that an approximately 50/50 split exists between organizations in which less than half of teams are agile versus those in which more than half are agile. While the exact meaning of *agile* in this context may be debated, overall, this points to software development organizations today continuing to employ a wide range of approaches, including plan-driven methods and techniques. Furthermore, the Digital.ai survey is consistent with other recent research (Nelson & Morris, 2014), indicating that a significant number of practitioners are still utilizing plan-driven techniques, such as the waterfall SDLC.

Overall, we see that, even though the Manifesto for Agile Software Development (2001) was published over 20 years ago, and key agile methods such as XP and Scrum have been available even longer than that, many teams are still using plan-driven methods and hybrid approaches that combine plan-driven with agile techniques (e.g., Baird & Riggins, 2012; Batra et al., 2010; Gemino et al., 2021).

Viewing these issues from an information systems education perspective, we note that agile methods have also

become highly popular in information systems education, both as a topic of study and as a foundation for project-based courses (e.g., Adkins & Tu, 2019; Sharp & Lang, 2018; Sharp et al., 2020). However, the reality that a wide range of approaches continue to be used in real-world systems projects today creates questions and challenges for teachers and students in information systems courses. For example: Should we teach both agile and plan-driven approaches? Are there hybrid approaches available, and if so, what are their characteristics? If we teach multiple approaches, how do we instruct students how to choose between these options, including what project characteristics to focus on and how those characteristics impact the choice of optimal project approach? Given that systems students are, after all, budding practitioners, we can see that these issues also have a direct impact and importance for experienced systems professionals.

In this light, this article's primary purpose is to explore the relationship between project characteristics and the systems development approach selected for the project, including the ways to address this question effectively in computing courses. Our goal is to provide instructors of courses in systems analysis and design, software project management, software engineering, and related areas (including project-based capstone courses) with case descriptions of systems development approach choices made in real-world projects. Using a relatively broad, primary data set of practitioner-based case studies is important for at least two reasons. First, it provides a level of realism that cannot be obtained in other studies based on small, student-based projects (e.g., Dhir et al., 2019). Second, it provides a greater breadth of contexts and, therefore, explanatory power than studies focusing on a single project or team context (e.g., Almudarra & Qureshi, 2015).

These case descriptions will be explained via a conceptual framework utilizing an effective visual representation to understand salient project dimensions and determine the optimal software development approach in each case. This framework is an effective teaching tool and a resource for organizational practice.

1.1 Explanations for Why Agile Is Not Universally Used: Stubbornness vs. Intelligence

If agile represents a major step forward over traditional, plan-driven approaches, why has it not largely displaced those older methods? There are at least two key possibilities. First, it could be that traditional, non-agile organizational cultures – including both information technology departments and the overall organizations they belong to – are stubbornly and irrationally slow to relinquish their traditional values and processes. This explanation is often associated with the viewpoint that agile approaches are uniformly superior to traditional approaches in all organizational and project contexts. See, for example, McKendrick (2020), in which the author argues that efforts to implement agile approaches widely are being “crushed by organizational inertia.” This reflects a vocal, uncritical belief in the near-universal superiority of agile methods – a position that Boehm and Turner (2004, p. 4) describe as exhibiting “near-messianic stridency.” (To be fair, Boehm and Turner indicate this can be true of both agile advocates and resistant traditionalists.)

In contrast, the second key possibility is that teams that do not adopt agile but utilize either plan-driven techniques or both plan-driven and agile techniques in a hybrid fashion do so for

rational, well-justified reasons. Leaders of these teams recognize that, given the characteristics of their projects, doing so represents a better process choice leading to better project outcomes than a “pure” agile approach.

In this viewpoint, there is no “single best way” to develop software. Rather, agile techniques will work best in specific circumstances where agile assumptions are met. At a summary level, these include:

- **Ability to engage in effective informal, verbal communication:** This implies a small team that is colocated, including not only IT professionals but also business customers who are highly available to work with IT team members (Boehm & Turner, 2004, pp. 32, 34-35, 37). Additionally, team members should all exhibit a high level of capability and cohesiveness (Boehm & Turner, 2004, pp. 46-47).
- **Small, simple functionality:** This lends itself to the appropriateness of informal communication and the consequent lack of comprehensive requirements documentation (Boehm & Turner, 2004, p. 28).
- **Low criticality:** This pertains to the costs of errors and failures – applications that are not mission-critical, do not manage sensitive customer data, are not subject to regulation and audit, and do not impact human safety can more readily implement the informal, low-documentation techniques of agile (Boehm & Turner, 2004, p. 39).

When these assumptions are violated, agile techniques become less effective. Seminal authors have noted these and other issues focused on plan-driven techniques (Cockburn, 2001, p. 187; Jacobson et al., 2016, pp. 120-121) and agile techniques (Cohn, 2004, p. 188; Leffingwell, 2011, p. 247). More generally, contingency frameworks have been developed for assessing project characteristics to strike an optimal balance between agile and plan-driven approaches (Boehm & Turner, 2004; Spurrier & Topi, 2017).

1.2 Focus on Systems Relevant to MIS Pedagogy

In this paper, we focus on business application software systems, and in particular enterprise-level software systems. While this excludes a wide range of other system types (e.g., embedded systems, Internet of Things, system software), it does concentrate on the kinds of systems that are at the heart of management information systems (MIS) curricula. We term the corresponding projects that MIS teams undertake to create those large-scale business application software systems *enterprise software development*, or ESD. Enterprise software refers to systems focused on transaction processing or data analysis processes in business domains, such as retail order processing, financial trading, insurance claims processing, supply chain, customer service, and the like (Fowler, 2003). As such, enterprise software tends to exhibit the following characteristics:

- **Strategic and mission-critical:** Software key to strategic differentiation and the ability to operate and compete.
- **Complex, organization-specific functionality:** Typically, not attainable solely from a single commercial off-the-shelf (COTS) product. If COTS software plays a major role in an enterprise software solution, it will typically be supplemented with custom

integrations (“glue code”) with internally developed applications or extensions, such as custom user interfaces.

- **Large amounts of diverse, persistent data:** Generated by transaction processing and/or subsequently analyzed for reporting. Large numbers of entity types create a need for many corresponding user interface screens.
- **Serving many users:** Across the organization, often in multiple roles and units.
- **High risk:** From handling customer data, facing the public internet, being subject to audit, or, in some cases, supporting human safety.
- **Needing robust architecture:** To support industrial-strength reliability, scalability, extensibility, and criticality.
- **High budget, large IT staff, and high visibility:** Arising from all the factors above.

These are broad descriptors, but they do narrow our focus to the kinds of transactional and analytics systems most frequently taught in MIS courses. Furthermore, note that we do not specify ESD or large-scale agile solely based on the number of developers or number of teams (as, for example, Dingsøyr & Moe, 2014, and Dikert et al., 2016, do). Also, we are not focused on the exact structure of teams nor on the path to implementing those structures (as, for example, in Gerster et al., 2020). Finally, to be clear, in this context, “enterprise software” is distinct from COTS enterprise resource planning (ERP), customer relationship management (CRM), or similar third-party systems. While our definition may include project contexts that are primarily built using COTS products, it generally focuses on contexts in which another team outside of COTS vendors needs to engage in significant, additional custom software construction, integration, or configuration unique to that organizational context. Alternatively, it could refer to the custom development of the COTS product itself, by the software vendor’s own IT team.

1.3 Fundamental Project Approaches and Project Characteristics

Agile pundits and authors have acknowledged that challenges exist when scaling agile. But they have strongly asserted that, by implementing certain adaptations that retain both the core character and benefits of agile, it is possible and, in fact, desirable to scale up agile approaches for large-scale, complex projects – that is, in projects we define as ESD. These assertions are evidenced by the publication of a significant number of “scaled agile” techniques and approaches, including Agile Project Management (APM) (Highsmith, 2010), the Scaled Agile Framework (SAFe) (Leffingwell et al., 2018), Large Scale Scrum (LeSS) (Larman & Vodde, 2017), Disciplined Agile Delivery (DAD) (Ambler & Lines, 2012), and others. However, while these frameworks indicate support for scaling agile, the need for them to supplement base approaches such as Scrum implicitly also acknowledges the challenges of doing so. This need raises key questions. First, in what specific ESD circumstances should base agile techniques be augmented with scaled agile techniques or even replaced with plan-driven or hybrid techniques? Second, in response to those circumstances, how should agile techniques be modified in terms of fundamental project approach dimensions, including functional

requirements, non-functional requirements, implementation processes, and team structure?

This paper addresses these questions by leveraging primary data from 22 case studies. Via this data, we make sense of practitioner systems project approach choices and their motivations for those choices. Additionally, we explain those choices in terms of the fundamental project dimensions just mentioned rather than arguing the relative merits of any of the scaled agile frameworks that we have noted or, for that matter, other non-agile frameworks. In doing so, we provide a fundamental understanding of system project approach dimensions, choices, and motivations. One of our fundamental goals is to offer information systems educators teaching courses related to systems development a rich set of material that they can use to illustrate the importance of and the practical challenges related to these decisions.

To lay the foundation for these arguments, we begin with a conceptual discussion of the fundamental dimensions of software development projects:

- **Project Approaches:** We span the two most general characteristics of traditional plan-driven versus new agile systems project approaches: requirements and construction. This discussion includes how those two fundamentally different approaches can be melded into a hybrid approach. It also focuses on fundamental project approach characteristics and thereby avoids becoming mired in the details and terminology of any specific framework.
- **Project Characteristics:** With project approaches in hand, we then turn to key project characteristics that impact the choice of project approach in any particular project. Note that we will map those characteristics to three higher level characteristics categories: functional requirements, non-functional requirements, and team characteristics. This schema will provide conceptual clarity for understanding how each individual characteristic impacts the choice of project approach.

1.3.1 Fundamental Project Approaches and Project Characteristics. By *project approach*, we define the essential characteristics of a systems project, and, in particular, whether it is an agile approach, a plan-driven approach, or a hybrid approach that combines aspects of the other two. Rather than focus on particular methods and frameworks (e.g., Scrum, eXtreme Programming, traditional SDLC, SAFe, APM), we define project approach in terms of two key dimensions:

- **Requirements:**
 - When are software requirements captured during the project? In particular, are they captured before construction begins or while construction is happening?
 - How much formal requirements documentation will be created during the project?
- **Construction:**
 - How do we structure software construction – in a single long phase or a series of iterative cycles (called “sprints” in the Scrum methodology)?
 - When do we review working software with business customers – only at the end of the project or periodically during construction?

By *project characteristics*, we mean to identify, organize, and interrelate key factors that influence the optimal choice on the spectrum between agile and plan-driven approaches, where the points between agile and plan-driven represent a variety of hybrid approaches. As explained below, given the complexity of software projects, numerous factors impact this choice. We organize them into three main categories:

- **Functional Requirements:** Define the behavioral features of the system – what the system does and how it does it to support the needs of a specific type of user in accomplishing a business goal or obtaining business value. Specific characteristics of functional requirements may determine the suitability of an informal, agile approach to the requirements versus the need for formal, up-front functional requirements in the form of Big Requirements Up Front (BRUF).
- **Non-Functional Requirements:** Pertain to the qualities of the system that are not specific to a business or industry, including familiar “FURPS ilities” such as usability, reliability, performance, and supportability (Grady, 1992). As we will demonstrate, these requirements drive the need for intentional architecture in the form of Big Design Up Front (BDUF).
- **Team Characteristics:** Include the size, skill sets, and other characteristics of both the IT team and the customer team. In general, team characteristics should fit with both functional and non-functional requirements. For example, a large set of functional requirements would likely drive the need for a large IT team, while complex non-functional requirements would likely drive the need for a diverse set of IT skill sets (such as architects and security specialists). Furthermore, some aspects of team characteristics may independently influence requirements needs; for example, an IT team that includes offshore team members would likely need more BRUF, all other things being equal than a colocated IT team.

1.4 Review of Prior Relevant Research

To position this paper in the context of information systems (IS) research on systems analysis and design (SA&D) education and, specifically, research on pedagogical questions related to the selection of the SA&D approach between agile, hybrid, and plan-driven, we reviewed relevant IS education research published during the last 20 years. Our core focus was on journal articles and conference papers published in IS education journals (*Journal of Information Systems Education*, *Information Systems Education Journal*, *Communications of the Association for Information Systems*) and conferences (EDSIGCON, AIS SIGED, and AMCIS and ICIS Education tracks), but we also included relevant papers published in general peer-reviewed IS outlets. The purpose of this section is not to provide a comprehensive, systematic literature review of SA&D education, or a specific area within it, in the way Feng and Salmela (2020), Sharp et al. (2020), Sharp and Lang (2021), and Niederman et al. (2018) have recently done. Instead, our purpose here is to demonstrate the connections between this paper and prior IS education literature. We are indebted to the authors of those literature reviews for their excellent work in identifying potentially relevant articles.

This paper addresses questions regarding agile software development. Therefore, we want to clarify its positioning

using the framework proposed by Sharp and Lang (2018) – also featured in Sharp et al. (2020) – in which the authors specify the distinction between agile pedagogy and agile content. This paper focuses on the topical content of a typical SA&D course (project approach selection on the agile-hybrid-plan-driven continuum) instead of following a specific set of pedagogical principles (such as agile pedagogy).

In the context of the IS 2020 competency model (Leidig & Salmela, 2021), this paper addresses SA&D competency DEVP.SADN.3 “Identify SDLC Models.” We recognize the importance of this competency and advocate for its inclusion in the coverage of IS textbooks, which was found lacking in Sharp and Lang (2021).

The primary question in this paper is how to select the optimal approach to executing a specific SA&D project. Prior IS education research has recognized the importance of this question, but not at the same level of breadth and focus. McAvoy and Sammon (2005, p. 409) want to “highlight the importance of certain adoption factors to the adoption of an agile method ... [and] ... to determine the **viability of an agile method for a specific software project.**” Based on prior research, the authors identify 11 critical adoption factors categorized into four groups (Project, Team, Customer, and Organization). McAvoy and Sammon focus entirely on methods within the agile approach and do not specify the alternatives. Harb et al. (2015) acknowledge the value of project approach selection but focus only on agile methods without addressing whether agile is the right choice in the first place. Landry and McDaniel (2016) propose a way to integrate the coverage of agile concepts within a traditional project management course. They acknowledge at least implicitly that the agile approach is not always right for a project by introducing a case discussion question, “Is agile right for my project?” (p. 28) and by outlining seven conditions under which “agile is right for a project” (p. 29). They do not, however, discuss the non-agile options. These three papers demonstrate that the project approach or methodology selection is an important topic, even though they all focus on selecting one of the agile methods.

Prior IS education research includes a small number of studies that recognize the important role of the hybrid approach on the continuum between agile and plan-driven approaches. Baird and Riggins (2012) directly recognize the existence of agile, traditional, and hybrid IT project management methodologies in a paper that focuses on using a hybrid methodology in an undergraduate capstone project management course. They emphasize the importance of teaching the hybrid approach because of its popularity among industry practitioners. Rush and Connolly (2020) propose a pedagogical approach for teaching IT project management that is based on agile Scrum. They acknowledge that in organizational practice, the choice is not (only) between agile and plan-driven; instead, “companies can and do manage projects in a hybrid manner, mixing these methods, depending on the size, type, and needs of the project” (p. 198, quoting widely known Forrester report by West et al., 2011). Several general IS papers demonstrate the importance of the hybrid approach, including Vinekar et al. (2006), Fitzgerald et al. (2006), Batra et al. (2010), and Baskerville et al. (2011). Still, no scholarly work known to us addresses the focus areas of this paper, that is, the specific questions of how to determine the

optimal project approach in the hybrid space between plan-driven and agile and how to cover this issue in an SA&D course.

In sum, prior research on SA&D education has 1) identified the role of the hybrid approach to systems development that includes elements of both agile and plan-driven approaches and 2) recognized that the right selection of a specific development methodology or approach is important for project success. No prior study has, however, provided a detailed model for selecting an optimal approach on the agile-hybrid-plan-driven continuum. In this paper, we propose such a model and provide pedagogical guidance for helping students understand and apply the model.

We have now set the stage for the remainder of the paper. In Section 2, we describe our model of systems projects and key decisions that the leadership of each project (and/or organization) needs to make regarding the software development approach chosen, whether that choice is between agile, plan-driven, or hybrid. The model consists of 14 project characteristics organized into these three main categories; it substantially modifies and expands an agile versus plan-driven “home grounds” model from Boehm and Turner (2004). Section 2 also serves as a tutorial for instructors interested in incorporating the hybrid approach, the home grounds model, and the project characteristics impacting approach selection into their courses. Section 3 describes the general characteristics of 22 in-depth interviews, which, in turn, provide the case material in Section 4 for a much more detailed illustration of the home grounds model, including how each of the 14 characteristics affects the development approach choice. For instructors of IS courses, Sections 3 and 4 provide a rich selection of descriptions of organizational practices related to systems development approach selection. Section 5 notes exceptions to the extended home grounds model. Section 6 discusses the use and selection of project approaches, the home grounds model, and the radar chart method for both practitioners and information systems courses. Section 7 ends the paper with a summary.

2. THE FRAMEWORK

2.1 The Essence of Agile, Plan-Driven, and Hybrid Software Development

Before providing illustrative examples of the factors affecting software development approach choice in Section 4, we must provide an organized framework to describe some of our key assumptions regarding software development projects. For this purpose, we utilize a two-part framework developed previously (Spurrier & Topi, 2017).

Per the discussion in Section 1, the first part of this framework argues that the essence of agile and plan-driven approaches can be expressed in a simple matrix utilizing only two dimensions:

- Requirements: When and how features and designs are determined.
- Construction: When and how software is coded, tested, and deployed.

As portrayed in Figure 1 (first published in Spurrier & Topi, 2021), for plan-driven approaches such as waterfall, the choices for requirements and construction are Big Requirements Up Front (BRUF) coupled with single-phase, noniterative construction. In contrast, for agile, the choices are emergent

requirements, meaning documenting only high-level stories up front, with detailed requirements emerging during iterative construction (i.e., sprints).

Figure 1 allows us to pay less attention to the details of specific methodologies that are not essential for understanding the high-level differences (such as eXtreme Programming’s use of pair programming).

It is essential that this framework posit the possibility of a third, hybrid software development approach combining a high degree of plan-driven BRUF with agile, iterative construction. Even though it is not necessarily identified as such, many projects use this hybrid approach, which combines the advantages of up-front planning and requirements specification with iterative construction. In hybrid projects, only a portion of the detailed requirements is defined up front, with the rest emerging during implementation within minimum and maximum scope “guardrails” defined in the BRUF. Furthermore, iterative construction allows for frequent customer contributions, evaluation, and feedback, including revisions to requirements and priorities based on customer reviews at the end of each iteration.

This suggests a balance of BRUF and emergent requirements that could vary based on project characteristics. Instead of considering agile as “evolving and emerging” and plan-driven as “staid and traditional” (Cram & Brohman, 2013), this model suggests that the optimal choice of project approach is often hybrid, varying significantly between agile and plan-driven, depending on a project’s characteristics. We believe that articulating a spectrum of hybrid approaches between the agile and plan-driven approaches will be important for all SA&D and other systems development course students to learn. Section 2 serves as a tutorial in this important topic area.

2.2 A Home Grounds Model for Understanding Practitioner Motivations

Additionally, to understand the motivations for selecting a software development approach, we built on a contingency “home grounds” model first created by Barry Boehm and Richard Turner (Boehm & Turner, 2004), describing the fit of plan-driven versus agile approaches based on organizational and project characteristics. Our extended model is shown in Table 1 (first published in Spurrier & Topi, 2021), showing the 14 project characteristics mentioned previously. Note that, as the answer to “The degree to which...” for any given project characteristic increases, plan-driven approaches become more appropriate (plan-driven home ground). On the other hand, as any given project characteristic decreases, agile approaches become more appropriate (agile home ground). As described above, enterprise software development (ESD) frequently aligns with the plan-driven home ground.

Dimension	Plan-Driven	Agile
Requirements	Big Requirements and Designs Up-Front (BRUF and BDUF) <ul style="list-style-type: none"> Up-front fixed scope Comprehensive, detailed requirements documents 	Emergent Requirements <ul style="list-style-type: none"> Up-front flexible scope in form of prioritized user stories Detailed requirements created (mostly informally) during construction sprints
Construction	Non-Iterative <ul style="list-style-type: none"> Single, long phase of software construction Software demonstration and user testing only at end 	Iterative (Sprints) <ul style="list-style-type: none"> One- to four-week sprints Software demonstration at end of each sprint Revise each sprint based on customer feedback and updated requirements

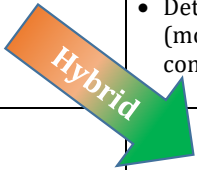


Figure 1. Agile, Hybrid, and Plan-Driven Approaches: Essential Characteristics (Spurrier & Topi, 2021; Copyright © 2021 Prospect Press)

2.3 The Primacy of Iterative Software Construction

Note that we expect that doing software construction iteratively will nearly always be an inherently better approach than traditional plan-driven, noniterative construction. This is because iterative construction enables frequent customer feedback and requirements course corrections during construction (Leffingwell, 2007). We therefore expect that, as project characteristics move toward ESD, organizations today that are engaged in coding new features would move to the hybrid approach, rather than a “pure” plan-driven approach (e.g., traditional SDLC or “waterfall”). Thus, for software construction, today the real choice is between agile and hybrid approaches, rather than agile versus “pure” plan-driven. This distinction is reflected in the column headings of Table 2 – “Agile Home Ground” versus “Hybrid Home Ground” – which portrays prototypical home grounds characteristics for each approach. (Table 2 was first published in Spurrier & Topi, 2021.)

In contrast, we expect that some organizations solely implementing a commercial off-the-shelf (COTS) software application (without significant new software coding) would follow a “pure” plan-driven process, given that these packages are often at least expected to be configured using a well-understood, repeatable series of configuration steps (Rubin, 2013, p. 8). However, based on our definition of ESD earlier, which specifies at least some new software development, such instances are outside the scope of the extended home grounds model. We note this as an exception to the model near the end of the paper in Section 5.6.

We next turn to the issue of making sense of the home grounds model in relation to scaled agile frameworks – are these conceptual frameworks fundamentally at odds with each other, or is there an underlying commonality?

2.4 The Home Grounds Model in Relation to Scaled Agile Frameworks

How can we relate the home grounds model to the agile versus scaled agile frameworks introduced in Section 1.3? Note that many of the home grounds model dimensions and their impacts

on projects would be familiar to students of the scaled agile frameworks. A good example of this is the Agile Project Management (APM) framework from Highsmith (2010), who is a leading agile author and a signer of the Agile Manifesto (2001). A careful reading of the APM framework, and in particular of the Agile Enterprise Framework included in the second edition (2010), reveals a concern with the challenges and impacts on the appropriateness of agile techniques that arise with respect to the following functional requirements characteristics:

- High numbers of features (p. 271)
- High levels of feature complexity (p. 276)
- High levels of feature interdependence (pp. 139, 276, 282)
- Low levels of feature uncertainty (pp. 129, 210, 276)
- Low levels of feature requirements change (pp. 110, 139)

Similar challenges and impacts can be found with respect to several of the non-functional characteristics (see pp. 87, 271, 286) and team characteristics (see pp. 142, 276, 282, 286, 287) listed previously.

Based on these issues, Highsmith acknowledges that a “multiplicity” of software development approaches may be warranted and that, furthermore, projects need to employ “*situationally* specific strategies, processes, and practices” (p. 77, italics in the original) that impact “organization, architecture, documentation, process” (p. 271). This causes a need to strike a balance between agile and plan-driven characteristics based on each project’s characteristics (pp. 70, 112).

Project Characteristic	Project Characteristic Description “The degree to which...”
FUNCTIONAL REQUIREMENTS CHARACTERISTICS	
Number	<ul style="list-style-type: none"> • Project includes many new features • Features include multiple functional areas
Complexity	<ul style="list-style-type: none"> • Individual features are complicated • Requirements vary across different users, departments, or offices • Goals require multiple projects and/or multiple systems
Interdependence	<ul style="list-style-type: none"> • Existing application is difficult to update because of high coupling • New features build on each other; must be built in a specific order
Clarity	<ul style="list-style-type: none"> • Current state of business and software is clearly understood • Future state can be clearly understood up front
Stability	<ul style="list-style-type: none"> • Requirements change slowly over time
NON-FUNCTIONAL REQUIREMENTS CHARACTERISTICS	
Performance	<ul style="list-style-type: none"> • Software must support large numbers/amounts of users/transactions/data • Planned need for high performance contrasts with current low-performance needs
Supportability	<ul style="list-style-type: none"> • Software needs to easily support future extensions to functionality • Software needs to be easily maintainable
Criticality	<ul style="list-style-type: none"> • Software is mission critical • Software security needs to protect sensitive data • Software impacts human safety • Software is subject to legal audit or requires formal requirements
Integration	<ul style="list-style-type: none"> • Software must integrate or interface with many other systems
Technology	<ul style="list-style-type: none"> • Project needs to use or integrate new or unproven technologies • Technology is obsolete and needs to be updated or replaced
TEAM CHARACTERISTICS	
IT Team Size	<ul style="list-style-type: none"> • Many IT team members • Multiple IT teams to coordinate
IT Team Locations	<ul style="list-style-type: none"> • Multiple locations • Multiple time zones • Multiple native languages and cultures • Multiple organizations (e.g., internal team working with a vendor)
IT Team Skills	<ul style="list-style-type: none"> • Many skill sets with high level of team member specialization • Team needs training on new or existing technologies • Team needs training on software development process • Team needs better cohesiveness and communication
Customer Team	<ul style="list-style-type: none"> • Many Subject Matter Experts, sponsors, other stakeholders • Multiple areas of expertise • Multiple locations, time zones, languages, cultures • Customers face diverse laws, regulations, and market practices • Customers value formal project planning and management

Table 1. Extended Home Grounds Model Showing Dimensions for Selecting Development Approach (Spurrier & Topi, 2021; Copyright © 2021 Prospect Press)

This leads Highsmith to describe a project that for good reasons utilized non-agile practices (p. 272), including “heavyweight” processes and documentation, albeit delivered iteratively – in essence, it was hybrid, although Highsmith does not use that word. However, Highsmith argues that the project was still agile because the team utilized an agile “mindset” in changing baseline agile practices to those “heavyweight” practices (p. 272). For students of information systems, describing an objectively non-agile project as “agile” because of the subjective mindsets of its team members is likely to be merely confusing: a distinction without a meaningful difference. It would be better to teach those students to pay attention to how the project is actually planned and executed –

in this case, using a hybrid approach, as described in Figure 1 in Section 2.1.

In reviewing Highsmith’s work in relation to this paper, we note the following:

- He acknowledges the need to balance a systems project’s approach based on what we call a project’s home ground characteristics.
- However, he does not provide a systematic, comprehensive model that students or professionals can employ to understand each of those project characteristics and their particular impacts on optimal project approach. Referring to the page numbers cited in this section, one can see that the arguments are scattered throughout his book.

Project Characteristic	Agile Home Ground	Hybrid Home Ground
FUNCTIONAL REQUIREMENTS CHARACTERISTICS		
Number	Small number of new features Focused on one function Small budget	Many new features Focused on multiple functions Large budget
Complexity	Simple features Single project Simple data schema Single version of requirements	Complex features Multiple interacting projects Complex data schema Many requirements variations
Interdependence	Brand-new software application Enhancements to a modern, well-designed existing application User stories are independent	Enhancements to an existing, “legacy” application that is poorly designed User stories must be built in a specific, logical order
Clarity	Start-up business New product, service, or function Responding to confusing, turbulent environment	Current business and software well understood (or can be) New requirements well understood (or can be)
Stability	Requirements changing rapidly	Requirements changing slowly
NON-FUNCTIONAL REQUIREMENTS CHARACTERISTICS		
Performance	Small number of users Low transaction/data volume	Many users High transaction/data volume
Supportability	Tactical application Proof of Concept/“throwaway code”	Strategic application High future investment
Criticality	Nonessential application Public data only Internal access only No safety risks No regulations or auditability	Mission-critical application Protect sensitive/confidential data Facing public internet Impacts human safety Subject to regulation or audit
Integration	Software operates in isolation from other systems	Software integrates or interfaces with many other systems New approaches (e.g., web services)
Technology	Continuing to use existing, proven technologies	New tech to learn, prove, or update Integrate with existing tech stack
TEAM CHARACTERISTICS		
IT Team Size	Under 10 team members Single team	Many team members Organized into multiple teams
IT Team Locations	Single location (single room) Common language and culture Team all from same organization	Multiple locations and time zones Multiple languages and cultures Multiple organizations
IT Team Skills	Strong technology skills Strong, existing project approach Long-standing, cohesive team	New, unfamiliar technologies Adopting new project approach Multiple new team members
Customer Team	Single Product Owner or SME Single department and function Customers in single location Single version of requirements	Multiple Product Owners or SMEs Multiple departments or functions Multiple locations, time zones, languages, and cultures Requirements vary significantly

Table 2. Home Grounds Model of Project Characteristics for Agile and Hybrid Project Approaches (Spurrier & Topi, 2021; Copyright © 2021 Prospect Press)

- Nor does he provide clarity in defining agile versus non-agile techniques and approaches applying to how a project is objectively planned and executed.

We offer this critique not to pillory Highsmith (or any other author advocating scaled agile techniques). Rather, we offer him as a prominent and, in fact, a typical example of agile literature that acknowledges that agile techniques are often inappropriate but fails to systematically address all the reasons

why agile techniques are not appropriate. Such literature thereby attempts to obscure the reality that in such circumstances, the optimal process is not agile but, rather, a complex hybrid of agile and plan-driven techniques. It was, in fact, this gap in the literature that motivated us to engage in this study.

On the other hand, we acknowledge that there is some validity in Highsmith’s point that an agile mindset can, to a limited extent, help teams adapt to objective project

characteristics that make agile less appropriate. Such team adaptations may ameliorate some of the impacts of those project characteristics. For example, as we will explain, a team operating in multiple physical locations will face more challenges to operating in a highly agile manner than a colocated team. However, a team that has operated in multiple locations for an extended period of time may, to an extent, be able to adapt to those circumstances, enabling them to operate in a relatively more agile manner than a team that has not previously had to operate in those circumstances. We will return to this point in Section 6, in which we describe approaches to utilizing our models for practitioners and students.

Having established that the scaled agile frameworks, in fact, advocate for hybrid approaches – albeit obliquely – we now turn to address an issue that they do not consider: how to systematically make sense of the many project characteristics that impact the optimal choice of project approach.

2.5 Visualizing Project Characteristics via a Radar Chart

The home grounds model explained earlier is complex, involving 14 separate dimensions. This reflects the reality that software projects are extraordinarily complex activities. Furthermore, it reflects that software projects are also difficult to understand in a summary fashion because of that complexity. Boehm and Turner (2004, pp. 54-56) introduced the use of radar charts to portray various project characteristics in a manner that could be comprehended at a glance.

In Boehm and Turner’s approach, a project that graphs near the outer edge of the radar chart is consistent with the plan-driven home ground, while a project that graphs near the center of the radar chart is consistent with the agile home ground.

We utilize this same approach, albeit with some key changes that we initially outlined in Section 1:

- **Number of major categories and more dimensions:** Boehm and Turner’s model utilizes five dimensions – Size, Dynamism, Personnel, Criticality, and Culture. As described earlier, our model, in contrast, expands these dimensions to 14, organized into three categories. Note that we merely introduce these categories and dimensions here but wait until our case studies to fully explain them and their impacts on project approach:
 - **Functional Requirements:** Driving the need for detailed functional requirements documentation, also known as “Big Requirements Up Front,” or BRUF. Note that functional requirements are the primary responsibility of the business analyst (BA). Within this category are five dimensions – Number, Complexity, Interdependence, Clarity, and Stability. Each of these drives the need for BRUF in a unique way.
 - **Non-Functional Requirements:** Driving the need for detailed non-functional requirements planning, also known as “Big Design Up Front,” or BDUF. Note that non-functional requirements tend to require support from specialists in areas such as architecture, security, and infrastructure. Within this category are five dimensions – Performance, Supportability, Criticality, Integration, and Technology.
- **Team Characteristics:** These generally should be driven by and fit with the functional and non-functional

requirements categories, but which also may, to an extent, drive the other two. Within this category are four dimensions – IT Team Size, IT Team Locations, and IT Team Skills, and Customer Team.

Figure 2 illustrates the model as a radar chart. The plotted line tracks near the outer edge for most dimensions, indicating a project that generally aligns with the plan-driven home ground (or hybrid approach, in the case of a software construction project). A line plotting near the center of the chart would be consistent with the agile home ground.

For each case study we discuss, we help explain the choice of project approach by providing a project characteristics radar chart to illuminate the case study text. We propose the radar chart approach as an effective pedagogical tool that enables students to understand better the joint impact of the complex set of factors affecting the fit between the systems development approach and project characteristics. In Section 4, we use the radar chart as a tool for illustrating our case studies. In that context, we also will discuss at a more detailed level how the radar chart can be used to determine the best systems development approach for a project. Let us first, however, describe the sources of the case study data and the process through which it was collected.

3. CASE STUDY DATA

The following case studies are based on one-hour, semistructured interviews with 22 software professionals, each of whom had leadership responsibility for enterprise software development projects. The sample included representatives from internal software development, COTS product development, and custom consulting development. We recruited them from our personal contacts, including industry colleagues, who tended to be older and more experienced, as well as former students, who tended to be younger. While this was not a random sample, it did tap into a broad range of experiences, including those who grew up in a mainly plan-driven environment and those who grew up in an environment in which agile was especially prominent.

Figure 3 presents the industry background of these professionals. Healthcare and insurance industries are disproportionately represented; this reflects the influence of our own professional backgrounds.

Figure 4 presents the perspective of each interviewee:

- **Internal IT Group:** Indicates interviewee was operating inside an organization directly supporting that organization’s enterprise software application needs. To the extent COTS software was part of the application mix, the interviewee was a consumer of that COTS software, not a vendor.
- **Vendor-Custom Development:** Indicates interviewee was operating as a consultant or within a firm offering custom software development to other organizations. Note that some of these include COTS products or components, but only in a highly unique, single-client manner emphasizing a high degree of custom software development.

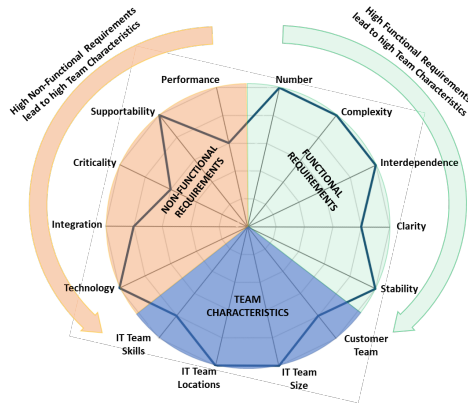


Figure 2. Sample Project Characteristics Radar Chart

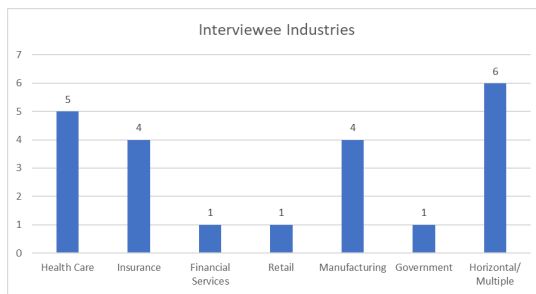


Figure 3. Industry Background of Interviewees

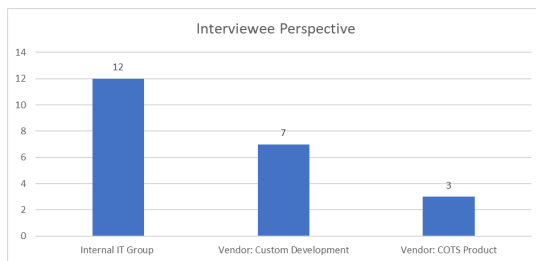


Figure 4. Perspective of Interviewees: Internal IT, Vendor-Custom, or Vendor-Product

- Vendor-COTS Product:** This indicates the interviewee was operating as a member of a vendor organization focused on creating and deploying highly configurable software products for multiple customers. As such, the software development considered was for the COTS product itself.

Table 3 briefly summarizes the case examples covered in Section 4 and their alignment with the corresponding radar chart figures.

Both authors participated in each interview. The interviews were semistructured, and each began with a review of the software development framework described earlier. General questions included the following:

- Does the software development framework make sense to you?

- What kind of enterprise software are you focused on?
- What is your software development approach, including your approach to requirements and to development?
- Do you have a label for your approach?
- What determines your selection of a given software development approach?

Our original plan was to focus each interview on a specific software project. However, it became clear at the outset that several respondents naturally wanted to provide insights from multiple software projects. It was especially the case that older, more experienced respondents could provide insights across many projects and software development eras, so we allowed each respondent to do so based on their breadth of experience.

All interviews were conducted via videoconferencing, although some respondents joined using only audio. Both authors took written notes, and all interviews were audio-recorded and transcribed for analysis. Each author then coded the interview data independently using categories based on the framework in Table 2. For example, Scope Size, Scope Clarity, and Software Development Interdependence categories were included as likely determinants of the software approach utilized. Each of the authors then extended those categories based on detailed transcripts reviews. For example, Architecture and Application Type emerged as an important determinant of the approach employed. Given this analytical method, the findings were grounded in preexisting theory and emergent concepts from the data. Finally, the authors compared their independent findings, with one of them combining the findings and the other finalizing them.

This sets the stage for the next section, in which we examine each of the case studies and utilize radar charts to enable understanding the motivations of each practitioner in adopting a particular software development project approach or, in some cases, multiple approaches.

4. ILLUSTRATIVE CASE STUDIES

This section discusses case examples, using them as examples illustrating the impacts of various project characteristics on the project approach. We start with functional requirements – the traditional province of the business analyst. We then proceed to non-functional requirements – typically the province of technical specialists, such as architects, security specialists, and infrastructure specialists. We finish with team characteristics, including considerations of fit with functional and non-functional requirements and the team’s independent impact on the need for planning. The purpose of these case examples is twofold. First, they demonstrate that even a small sample of 22 interviews includes examples of each factor affecting project approach choice, lending support for and illustrating the conceptual model. Second, as discussed earlier, they provide educators teaching SA&D courses with material for illustrating these factors and their impact on systems development approach selection with practical cases.

Company Type	Key Point Demonstrated by the Example	Figure
Health care provider	Large portfolio of integrated information systems led to high Number and high Complexity	Figure 5
IT consulting; single IT team per project (max 12 weeks)	Intentional choice of small Team Size, small Number, and low Complexity	Figure 6
IT consulting; single IT team per project (max 12 weeks)	Impact of back-end components leading to high Interdependence and high Criticality	Figure 6
Manufacturing; complex modifications to trucks	Initially low Number, Complexity, and Interdependence in greenfield projects, but these three characteristics will increase when the systems mature and their value is established	Figure 8
Insurance broker	Two project types: “Enhance the Business”: High Stability and high Clarity; “Research & Development”: Reduced Stability and Clarity	Figure 10
Health care software	Less focus on requirements because of lower Stability than anticipated	Figure 11
Large-scale survey solution	Lower planning need despite very high Performance requirements because of use of existing third-party components	Figure 12
Insurance company	Rewrite of a legacy claims processing system with high Criticality, Performance, and Supportability requirements – use of agile led to “disaster” in terms of Supportability	Figure 13
Expert architect with experience over number of solution contexts	Moving toward the use of a microservices architecture to enable high Supportability	Figure 14
Online retailer	Rare example of a system with low Criticality: Sending order status messages to customers	Figure 15
Manufacturing company	Systems integration project with high Integration, Technology, and IT Team Skills requirements	Figure 16
Major financial services company	High Technology coordination needs across multiple divisions; overall, high values for all characteristics	Figure 17
Multinational insurance company	Different Customer Team characteristics in the United States and Europe, leading to different project approaches	Figure 18
Big data analytics COTS product vendor	Very high feature requests lead to a need to utilize a formal requirements management tool; different client requirements regarding speed of changes and documentation requirements	Figure 19
Technology consulting firm building customer websites	Impact of Team Size on development approach selection	Figure 20
High-tech manufacturer	Back-office administrative systems developed by teams in the United States and India; impact of IT Team Locations and IT Team Skills on approach selection	Figure 21

Table 3. Summary of Cases Featured in Section 4

4.1 Functional Requirements and Big Requirements Up Front

When creating functional requirements, BAs, product owners, and other requirements-focused team members must choose between the plan-driven technique of creating comprehensive, formal documentation up front versus the agile technique of deferring those requirements so that they can emerge (Agile Manifesto, 2001), likely in an informal manner, just before the feature is created. The former is commonly termed “Big Requirements Up Front (BRUF)” (Ambler, 2014) – the term we use in this paper – or “Big Up Front Design (BUFD)” (Leffingwell, 2007, p. 30). The latter is called “emergent” requirements (Boehm & Turner, 2004, p. 29).

Importantly, as noted earlier, the choice is not binary – teams can explicitly choose to do a portion of the requirements upfront while deferring the remainder to iterative construction cycles. The following examples illustrate how real practitioners assess project characteristics in striking that balance.

4.1.1 Number and Complexity: Impacts of Largeness. The first two factors – Number and Complexity – and their impacts on projects are easy to understand. Still, they set the stage for the more complex impacts of Interdependence, Clarity, and Stability, considered below.

Together, Number and Complexity account for the “largeness” of a project – the number of features to build and how difficult each feature is to build. Not surprisingly, as either of these factors increases, it becomes more and more difficult to handle requirements in an informal, agile way. Many features need to be formally tracked, and individual complex features need to be documented in detail.

An excellent example is a health care organization with multiple hospitals and physician clinics. It employs over 4,000 employees in various functions – doctors, nurses, administrative staff, etc. Furthermore, it serves tens of thousands of patients, many of whom access the organization’s large portfolio of integrated information systems, including a COTS system for electronic health records interacting with numerous internally developed applications. The radar chart for

most projects in this organization is portrayed in Figure 5, in which the blue line represents the profile of the project characteristics, and the red stars highlight the two factors of interest, Number, and Complexity. Not surprisingly, these projects plot near the outer edge of the radar chart. In particular, for this example, the IT group receives 600 to 700 requests for coding changes each month. Furthermore, many of those requests are individually complex. This necessitates what their Director of Applications called a “highly defined process” for handling requirements, including highly formal functional requirements and “documented, iterative builds.”

At the other end of this spectrum is an example from an IT consulting group “culturally committed” to agile principles and practices, including eXtreme Programming (XP). In addition to providing custom software development services to its clients, this group also educates IT team members from those clients in agile development practices by having those clients temporarily join their team as pair programmers. From the perspective of illustrating the impacts of Size and Complexity, the interesting thing about this example is that all of this team’s projects are kept small – none targeting more scope than a single, small IT team can accomplish in 12 weeks. This organization will not accept projects larger than this because they become difficult to execute in an agile manner. In Figure 6, the blue line indicates the project characteristics of a typical web development project for this group. The red stars highlight the small Number and Complexity of features necessary to keep these projects small. (Note: We will examine the red line and yellow stars in the next section on Interdependence.) A project significantly larger than described would plot progressively farther out on the Number and Complexity, consequently requiring more BRUF.

4.1.2 Interdependence: When One Thing Depends on Another. Agile approaches assume that features – typically expressed as user stories – can be written in such a way that they are independent. This means they can be developed in any order and will not impact other stories or existing functionality (Wake, 2003).

This is illustrated in Figure 7 (first published in Spurrier & Topi, 2021), which shows that, when features are independent, it is possible to defer detailed requirements (or, alternatively, accept changes to BRUF) late in the project without incurring exponentially increasing costs; conversely, if interdependence is high, then late changes generate exponential cost increases (Beck, 2000), which is the plan-driven assumption.

However, feature independence needs to be met in many situations. Rather, individual features and entire applications are often unavoidably **interdependent** – they must be built in a particular order or coordinated fashion, leading to the need for a more plan-driven approach.

Interdependence can arise in several ways. One way is simply the familiar example of systems that involve multiple application layers, such as a dynamic website leveraging presentation, business logic, and a database. For example, let’s consider the same IT consulting group shown in Figure 6. They indicate that one of their typical small web projects exhibits a high level of independence, per the agile idea. However, they also indicate that when the project involves back-end systems in the build, the level of Interdependence jumps substantially, forcing them to plan the project to avoid critical problems, such as data loss. (Also note that anticipating discussion of the non-functional dimension of Integration later in Section 4.2.4,

Interdependence and Integration here would be elevated, but for different reasons: Interdependence arises because the system designs must be *constructed* in a specific manner in relation to each other. In contrast, Integration arises because those multiple systems must be designed to operationally *interact* with each other.)

Note that this increases the Criticality of the application, as well – a subject to which we will return in the discussion of Non-Functional Requirements. These impacts are indicated for both Interdependence and Criticality by the yellow stars and the arrows pointing outward toward the edge of the graph. The overall, revised red line indicates that even groups that are committed to agile principles and practices may need to deviate from them in relatively common circumstances.

We complete the section on Interdependence with one more example, that of a large manufacturing company that implements complex, aftermarket modifications to large trucks.

Overall, typical projects for this organization plot near the edges of the radar chart, as shown by the brown line in Figure 8. The use of brown reflects that these are “brownfield” projects – meaning enhancements to existing systems that are already large, complex, integrated with other systems, and used across multiple facilities. However, the Senior Manager of this group noted that individual, “greenfield” systems – meaning new systems built from scratch – often start out in a much more agile way: small, simple, and originating independent of other systems within a single facility. This can be especially true of systems that originate with an end user in a single manufacturing facility to solve a single problem. See the green line in Figure 8. However, when such systems are identified as being valuable, they are often extended to other facilities and matured with additional features and integration with other systems. In this way they become enterprise systems, with increases in the Number, Complexity, and Interdependence of features to be constructed. Similar to the previous example in Figure 6, the non-functional requirement of Integration also increases, as the system needs to be designed to operationally interact with other systems in the enterprise environment. Overall, the effect is the same: a system that could be initiated in an agile manner requires more and more planning over time, as reflected in the changes in the chart from the green line to the brown line and arrows highlighting the movement toward the edge of the chart.

4.1.3 Clarity and Stability. The Value of BRUF Over Time.

We next consider two related project characteristics impacting BRUF: Clarity and Stability (see also Spurrier & Topi, 2017). Clarity is the degree to which we understand (or can with a reasonable level of effort understand) software requirements, including features and designs. A central agile belief is that BRUF holds no initial value because it is very difficult or impossible to accurately discover and structure requirements prior to illustrating the ideas in functioning code. Not surprisingly, plan-driven assumes the opposite: BRUF can provide substantial value because it is possible to determine a well-structured set of requirements with reasonable effort and time (Leffingwell, 2007, p. 20).

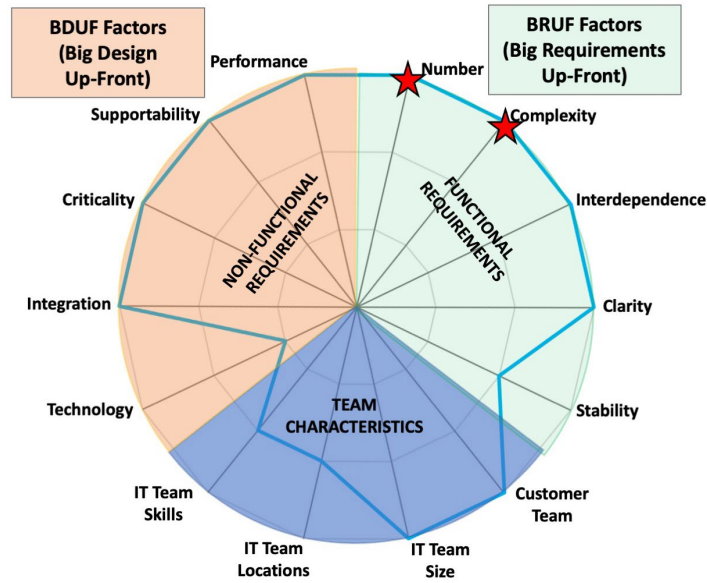


Figure 5. Radar Chart for Typical Projects at a Health Care Provider

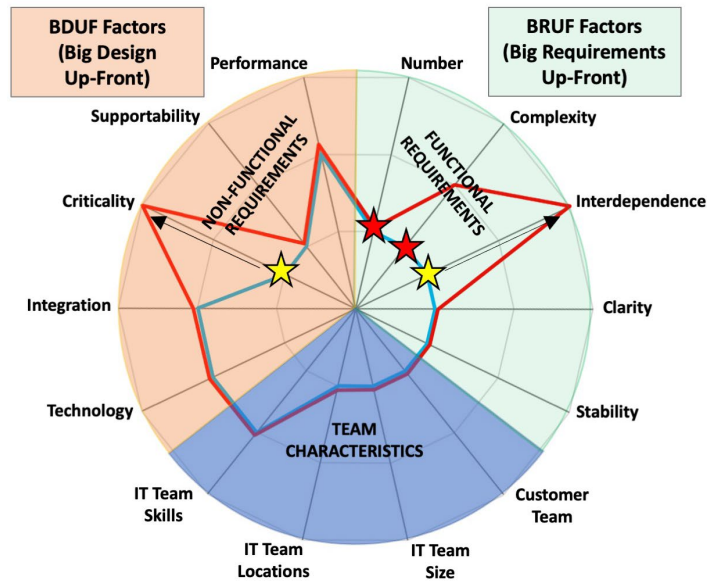


Figure 6. Radar Chart for Typical Projects at an IT Consulting Firm

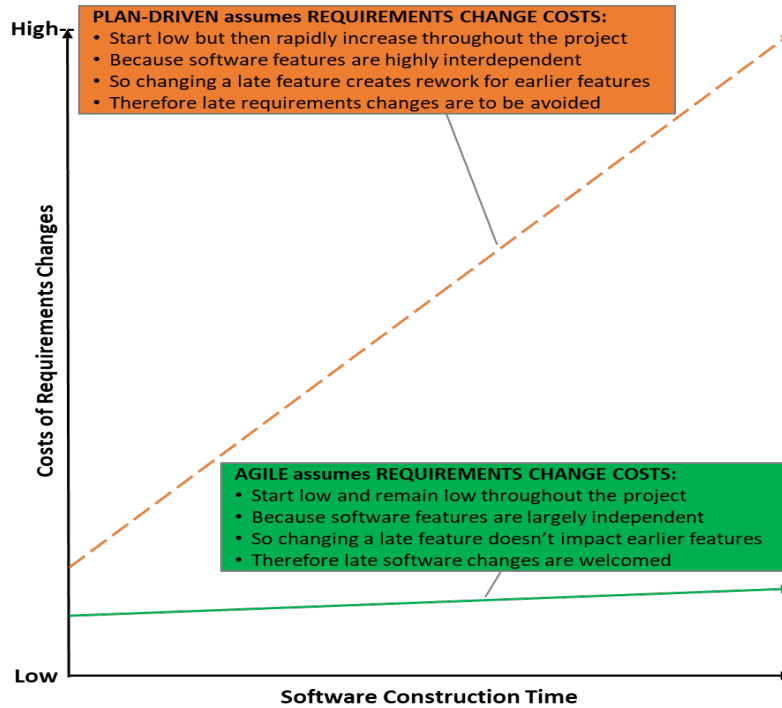


Figure 7. Impacts of Interdependence on Software Change Costs as a Project Executes (Spurrier & Topi, 2021; Copyright © 2021 Prospect Press)

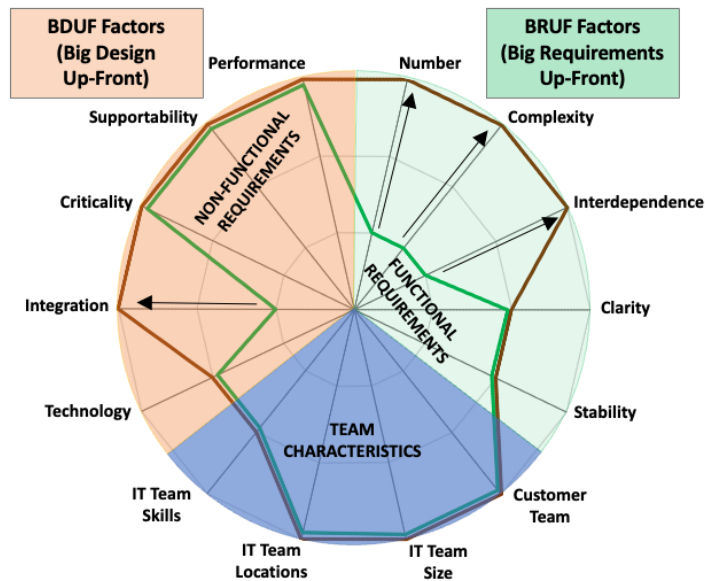


Figure 8. Impacts of Greenfield Systems Evolving into Mature Enterprise-Level Systems

Stability, in turn, is the degree to which software requirements remain unchanged over time. Numerous sources advocating for the benefits of the agile approach (e.g., Augustine et al., 2005; Cockburn & Highsmith, 2001; Sarker & Sarker, 2009) suggest that software requirements continue to change faster than in earlier eras because of the increasingly dynamic nature of the organizational domains within which the systems are used. Consequently, the initial value of BRUF would quickly go down because the change in the organizational environment would reduce the alignment between the initially documented and currently needed requirements. Therefore, agile believes strongly in specifying requirements as late as possible. Not surprisingly, plan-driven takes a different approach and builds on the assumption of requirements stability and suggests that BRUF will continue to be valuable during development.

These dynamics are illustrated in Figure 9 (first published in Spurrier and Topi, 2021). These points are illustrated by a software development manager for a large insurance broker. His software group explicitly distinguishes between “Enhance the Business” (ETB) projects, in which mature enterprise systems are given a relatively small number of enhancements (as brownfield projects), versus “Research and Development” (R&D) projects, in which a team works to develop a completely new and generally unclear set of software requirements (as greenfield projects). He describes these as different “pillars” of development within the same IT group. The ETB projects are thoroughly planned, starting with formal roadmaps and continuing through BRUF that is explicitly adjusted from 80% (complemented with 20% agile emergent requirements during development) down to 40% (with 60% emergent), depending on the Clarity of the requirements. In contrast, the R&D projects exhibit low Clarity and Stability, leading to a highly agile project approach, including requirements.

Figure 10 illustrates this distinction, with the green line representing a typical R&D, greenfield project – agile, except for frequent issues of needing to test new technologies and, consequently, needing expanded IT Team Skills. The brown line represents the situation for ETB, brownfield projects, with Clarity and Stability explicitly considered (and here emphasized with arrows) when determining the optimal level of BRUF. We conclude this section with one final example, specifically highlighting the impacts of lower-than-anticipated Stability. The interviewee was a project leader for a health care software firm focusing on implementing advanced document search capabilities across multiple systems. As shown in the radar chart, the non-functional requirements were daunting, with very high volumes of documents to search, concerns over missing or losing document data, and the need to integrate with an external search technology vendor and many underlying client applications systems. In contrast, the functional requirements for document search were not that large, with relatively few fairly simple use cases. In this case, the IT group still planned on significant BRUF (75% – see green line in Figure 11) because those requirements were understood to exhibit high Clarity and Stability. However, the requirements turned out to be less stable than anticipated, causing the group to revise their level of BRUF down to 50%. (See the arrow reflecting the change to the revised, red line in Figure 11.)

4.2 Non-Functional Requirements and Big Design Up Front

While functional requirements are the traditional focus of business analysts and product owners, non-functional requirements (NFRs) pertaining to the “ilities” – scalability, extensibility, reliability, security, and contractual and legal requirements (Leffingwell, 2007, p. 222) – are equally important to project success. The plan-driven architectural effort to systematically address these NFRs is sometimes called “Big Design Up Front” (BDUF) (Boehm & Turner, 2004, p. 42). This contrasts the agile approaches, which generally do not even define a separate architectural role (Leffingwell, 2011, p. 388). Rather, agile prefers to allow a system’s architecture to emerge organically during the development effort (Agile Manifesto, 2001; Leffingwell, 2011, p. 54), which recognizes that for small systems in uncertain environments, the effort of BDUF may not be worth it (Boehm & Turner, 2004, p. 42). On the other hand, in enterprise systems, the high costs of NFR failures typically justify moving from a purely agile emergent architecture toward plan-driven intentional architecture (Leffingwell, 2011, pp. 386-387). This, in turn, justifies the need to engage the services of technical specialists, such as architects, security specialists, infrastructure specialists, and others, who may live outside the development team (Leffingwell, 2011, p. 54).

As with functional requirements, the choice in relation to NFRs between agile emergent architectures versus intentionally architecting a system via BDUF is not binary. Teams can and do make intelligent choices based on the project characteristics. The following examples illustrate how enterprise development leaders strike that balance.

4.2.1 Performance: Preparing to Carry a Heavy Load.

Preparing to Carry a Heavy Load. We begin with the NFR of Performance, which is the amount of work a system must perform in terms of volumes of users or transactions. This typically includes speed and reliability metrics, such as transaction throughput, response times for a given number of users, uptime during normal business hours, etc. Performance can be viewed as the NFR analog of the functional requirements of Size and Complexity: a system with high Performance requirements is “large” in terms of NFRs, analogous to the way that a system with many complex functional requirements is large.

In this light, it is unsurprising that a system needing to support, for example, millions of simultaneous users would need BDUF to ensure it performs adequately. What is more interesting is to consider situations in which such a project would *not* require high BDUF. Such a circumstance was described by a senior developer working to create an extremely large survey solution for a government entity. Had that solution needed to be created from scratch, that greenfield project would clearly have needed extensive intentional architecture to ensure performance, as shown by the green line in Figure 12. However, in this case, the solution was based on a series of existing third-party components already architected to scale to nearly these performance levels. While this did not reduce the need for BDUF to agile levels, it did reduce the planning level needed for this project’s Performance dimension. (Figure 12 shows the arrow pointing to a lower requirements level with the red line.)

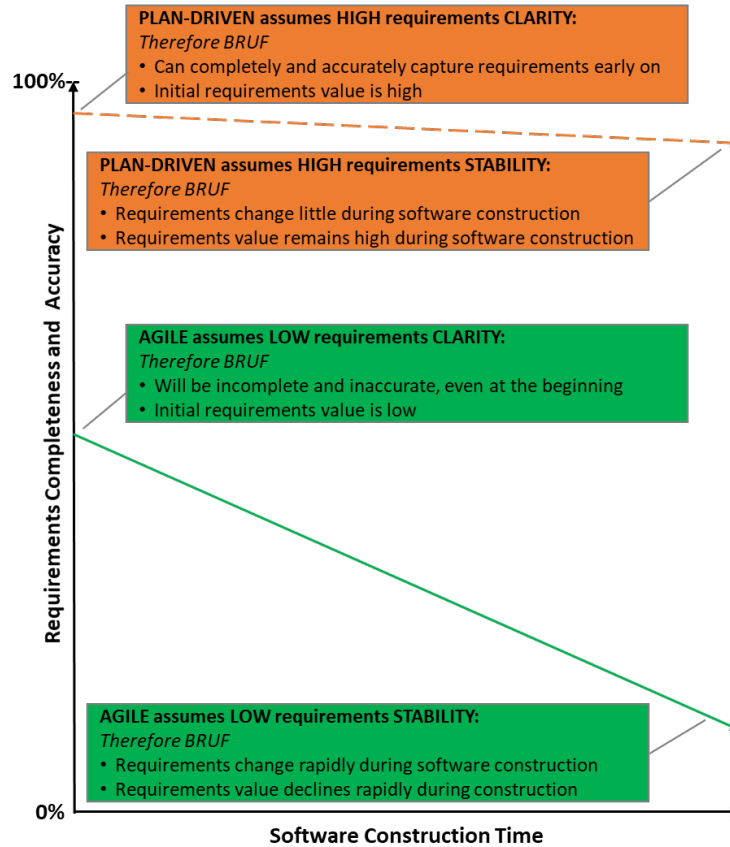


Figure 9. Agile vs. Plan-Driven Assumptions Regarding Clarity and Stability (Spurrier & Topi, 2021; Copyright © 2021 Prospect Press)

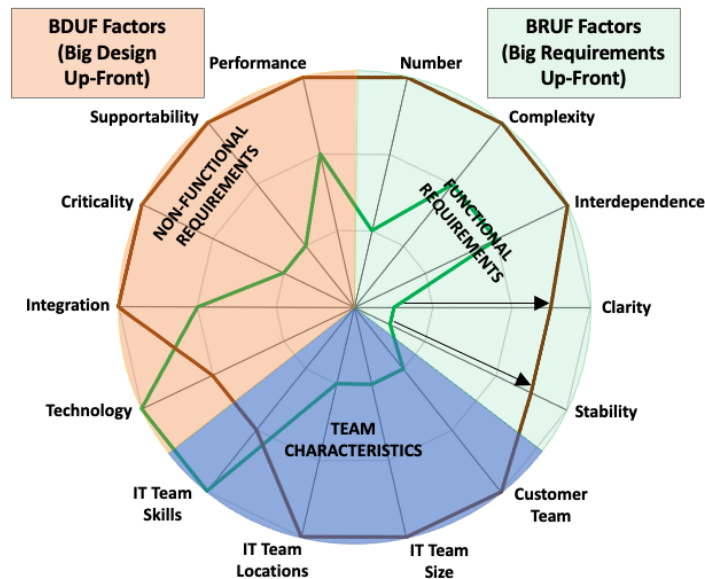


Figure 10. Radar Chart for Comparing “Enhance the Business” and R&D Projects

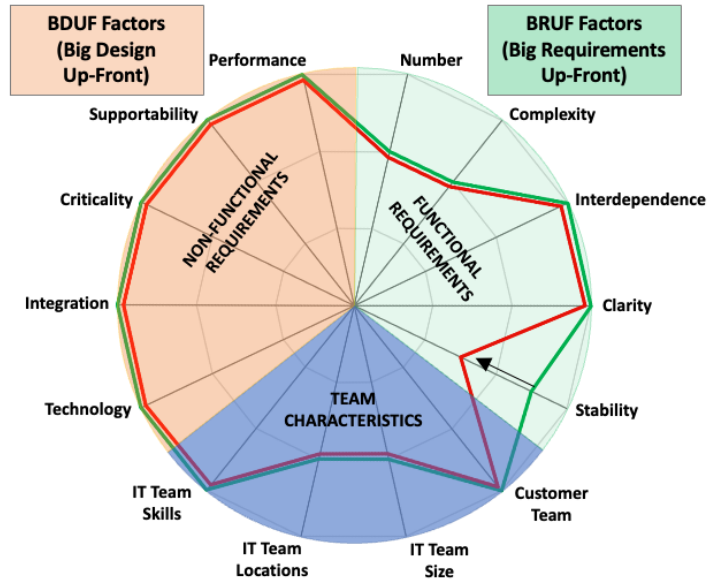


Figure 11. Impact of Stability on Emphasis on BRUF at a Health Care Software Vendor

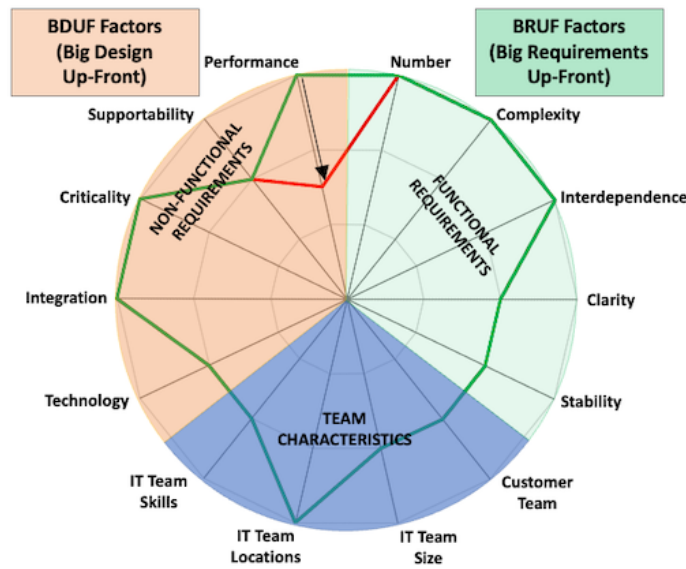


Figure 12. Impact of Existing Capabilities on Performance Planning Needs

Again, this is analogous to Size and Complexity in functional requirements: a project that adds a small number of enhancements to an existing large number of features is smaller than a greenfield project needing to create all those features from scratch.

4.2.2 Supportability: Building for the Long Haul. We turn next to Supportability, which is the degree to which a system needs to be easily reconfigurable and easily updatable with new features over time. We offer two examples in this category: one illustrating the danger to Supportability from using an agile-functional-requirements approach where it doesn't fit, and another illustrating ways to improve Supportability through intentional architecture.

The first example is from the CEO of an insurance organization that was responsible for health insurance claims processing system. This legacy system was functionally very large, containing many complex features. The team was pursuing a project to rewrite the system using a more modern technology stack and extend it with public internet-facing web interfaces. The project, not surprisingly, is at the outside edge of the radar chart (see Figure 13), with high Criticality, Performance, and – especially pertinent to this example – Supportability needs (emphasized with yellow stars and a red star, respectively). Despite this, the project team pursued a highly agile methodology that eschewed both BRUF and BDUF. This resulted in what the CEO termed a “disaster,” a mass of code that no single team member understood and lacked documentation sufficient for others to learn. As such, the system could not be effectively configured or operated. The CEO noted that, even if short-term operational problems were solved, it would be extremely difficult for team members in later years to update and extend the functionality.

The second example is from a deeply experienced software development leader who had created and extended many enterprise-class applications in multiple organizations. This leader noted that he had changed how he architected such solutions over time. Specifically, early in his career, he would plan a large system with many complex features as a single, large application. This is the typical architecture of mainframe systems, often resulting in a mass of millions of lines of code that are difficult to maintain and extend, even though enterprise-class systems typically need to be used in this way for many years (see the green line in Figure 14). Over time, however, he learned to architect such large-scale applications as a series of small, interacting applications, each focused on a specific capability area. This is the architectural concept of microservices, which, at its core, transforms one very large application into a series of smaller ones. While this does not reduce Supportability requirements (highlighted with the red star in Figure 14), it does make it possible to manage each of the microservices in a more agile manner by reducing the Number and Complexity of features in each microservice, as well as reducing team characteristic requirements at the microservice level (see arrows pointing to the red line in Figure 14). There is a degree of increased planning required to coordinate the Interdependence of the microservices, but the net effect is to make it easier to support the overall application environment over time, and especially in a more agile way overall. A key point here is that designing for microservices is fundamentally a BDUF activity to produce an intentional architecture for meeting Supportability requirements.

4.2.3 Criticality: Preventing High-Cost Failures. The Criticality NFR represents the costs resulting from defects and failures (Boehm & Turner, 2004, pp. 55-56). An application can have high Criticality requirements for several reasons: being mission critical to the organization, storing and managing sensitive data (especially when facing the public internet), being subject to legal audit or regulation requiring formal documentation, or impacting human safety. While the latter is not typically a major concern for most enterprise systems, the other three are quite common. For example, news dispatches of security breaches and ransomware attacks crippling the ability of organizations to function have become all too common.

As with the Performance NFR, these factors are so commonplace that pointing to enterprise systems that do *not* require BDUF to handle high Criticality is more challenging than pointing to the great majority that does. In the current examples, we point to one example that requires a relatively low level of criticality: a project for a retail website to send order status messages to customers.

While this project involved large volumes of communications with external customers, there are several reasons why this project needs to exhibit more criticality. First, this capability is not mission-critical because the overall application environment already provides other modes of communication to customers, including emails, calls to customer service representatives, and simply looking up the order on the main website itself. Second, while the customer order data is confidential, the security concerns are relatively low because the shared data is highly summarized (order number, shipping method, and dates). Furthermore, text messaging is one-way, minimizing the risks of hacking. Finally, there are no impacts on human safety.

This set of factors is portrayed in Figure 15, with the red star highlighting the somewhat reduced level of Criticality.

4.2.4 Integration and Technology: Imposing Order on the Unknown. Two NFR dimensions that frequently go together are Integration and Technology. Integration is the degree to which a system needs to operationally interoperate or exchange data with other systems (and, as noted in Section 4.1.2, is distinct from Interdependence, which pertains to the build order dependencies of software construction). Technology is the degree to which a system needs to be built or enhanced using new, unproven, or not previously integrated information technologies. A good example of this is a systems project led by a senior director of IT at a manufacturing company. The project focused on integrating the functioning of two sets of systems that were already familiar to the IT team: a third-party COTS enterprise resource planning system used for administrative functions and internally developed manufacturing systems that ran the factories. These two different sets of systems made Integration a major issue, indeed the primary goal of the entire project. But this integration challenge was magnified by the decision to utilize a new, third-party workflow and data integration platform that the team had never used before. This impacted the team by compelling them to expand their IT Skills – a topic we will explore further below. This situation is portrayed in Figure 16, with Integration, Technology, and IT Team Skills emphasized with the red stars. This example is similar to the more general Integration needs

from the manufacturing company described in Section 4.1.2, Figure 8.

Technology can become a major planning issue aside from the needs of any specific project. An example is that of a major financial services company with well over 100,000 employees in multiple divisions pertaining to banking, investments, and wealth management. Each division has its own large, sophisticated, division-level IT group. This results in the “maximum plan-driven” profile shown in Figure 17. Furthermore, pertaining specifically to Technology (highlighted with the red star), the bank is faced with the need to compete by utilizing multiple, burgeoning emerging technologies, including machine learning, blockchain, quantum cryptography, cloud computing, augmented reality, and many others. The point is that the overall organization needs a top-down evaluation and plan for employing these technologies that is consistent across the divisions – this is a function of one of our interviewees in a Project Management Office (PMO) and Tech Strategy Team.

4.3 Team Characteristics: Driven by Requirements but Driving Approach

Finally, in this section, we come to Team Characteristics, including IT Team Size, IT Team Locations, IT Team Skills, and Customer Team characteristics. While these are not requirements in and of themselves, they can profoundly impact the choice of project approach. Agile methods generally address team characteristics with prescriptive recommendations: teams should be small, colocated (including direct, continuous availability of the customer to the IT team), and highly skilled in the chosen technology tools and development processes (Boehm & Turner, 2004). Deviations from these ideals are seen as incompatible with the effective use of agile, including by agile advocates.

Team Characteristics interact with functional and non-functional requirements in two fundamental ways. First, analogous to the idea that form should follow function, Team Characteristics ideally should fit with requirements. For example, in the previous example, high Technology needs to imply a team needs to increase their IT Team Skills. For another, a large systems project requiring many new, complex functional requirements implies the need for a large IT Team Size – in particular, many developers and possibly testers (if QA is organized as a separate function).

Second, Team Characteristics can directly impact project approach, independent of functional and non-functional requirements characteristics. A classic example is that of IT Team Locations: as noted, agile assumes that the entire IT and customer team will be colocated. But offshoring developers is still highly common, increasing the need for plan-driven BRUF to support team communications regardless of the project’s functional and non-functional requirements.

4.3.1 Customer Team: The Wellspring of All Requirements.

We begin with Customer Team characteristics for the simple reason that all functional and non-functional requirements arise from the needs of business customers. Characterizing the “customer” along a single dimension may seem overly simplified, given that our extended home grounds model allocates three dimensions for the IT Team. This is a choice to keep the model from having too many dimensions to be useful. Furthermore, however, this single dimension can usefully

characterize the Customer Team against the agile ideal: using specifically the Scrum model, a single Product Owner (Schwaber & Beedle, 2002, p. 34) providing a single source of full-time subject matter expertise in the same, single location as the IT Team (Beck, 1999). The Customer Team may depart from this ideal in many ways, with the theme that “single” becomes “many”: many subject matter experts (SMEs); many different areas of expertise; multiple locations; facing diverse business processes, market practices, and regulations.

This is illustrated by a development leader in an insurance context who faced the need to develop two systems that addressed the same fundamental business need – one for the United States and one for Europe. While the details of the requirements certainly varied across the Atlantic Ocean, the bigger issue for this leader was that in the United States the requirements were driven by a single SME with unclear and rapidly changing requirements – essentially a product owner. This SME valued getting a working system to market quickly and was willing to accept relatively few simple features. In contrast, in Europe, seven different SMEs valued capturing and delivering highly formal known and extensive requirements. The situation is graphed in Figure 18. This led the development leader to pursue two radically different project approaches: a highly agile approach for the United States (red line) and a much more formal, hybrid approach for Europe (green line). As a second example, we consider the challenge faced by the solution architect for a COTS product vendor selling big data analytics tools to a wide range of corporate clients. This group faced an extremely high number of feature requests, requiring them to utilize a formal requirements management tool, like the health care organization described in Section 4.1.1. However, the challenge here of tracking and managing requirements was likely even greater because these were external clients, and those clients were highly diverse in their requests and, especially, the speed at which they wanted them fulfilled. In particular, many of the clients on the west coast were tech companies that wanted quick delivery of new features. Other clients on the east coast were highly regulated financial services companies that objected to implementing highly planned updates to the product more frequently than every quarter.

In essence, this product vendor faced the challenge of trying to simultaneously enhance the product in a highly agile and highly plan-driven fashion. This is portrayed in Figure 19, with the green line representing the tech companies and the red line representing the financial services companies. The arrows indicate that this diversity across the five requirements dimensions emanated from this highly diverse and complex set of clients (Customer Team highlighted with the red star). How did the vendor handle this? The answer was to utilize a highly plan-driven formal requirements process, which then identified small, simple, and independent features that could be created and deployed outside of the quarterly release cycle.

In general, this is typical of the challenges that product managers and architects for COTS product vendors face.

4.3.2 IT Team Size: Challenges of Communication and Coordination.

Shifting to the IT Team Characteristics, we begin with IT Team Size. Agile authorities uniformly specify that agile teams should be small. For example, in Scrum, the team should consist of no more than nine members (Schwaber & Beedle, 2002, p. 34-35). Beyond this limit, agile methods do not work well.

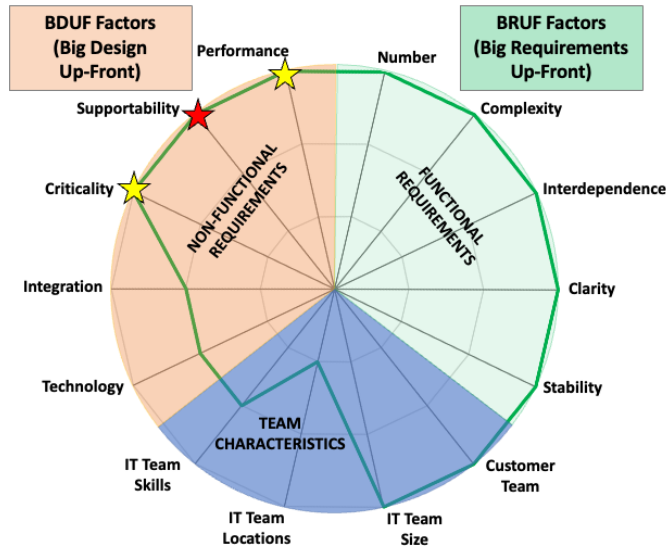


Figure 13. Radar Chart for a Large-Scale Insurance Claim Processing System Project

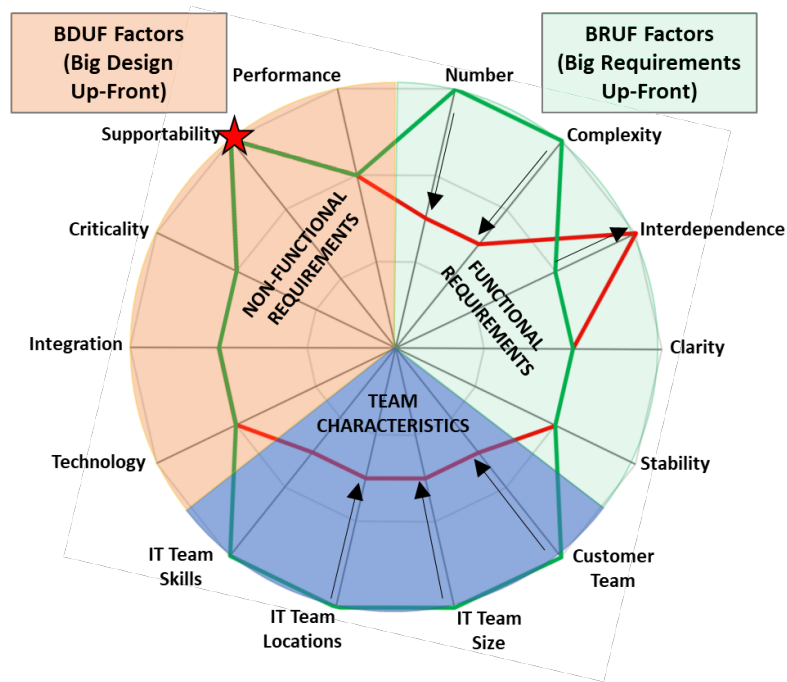


Figure 14. Impact of Moving to a Microservices Architecture Illustrated with a Radar Chart

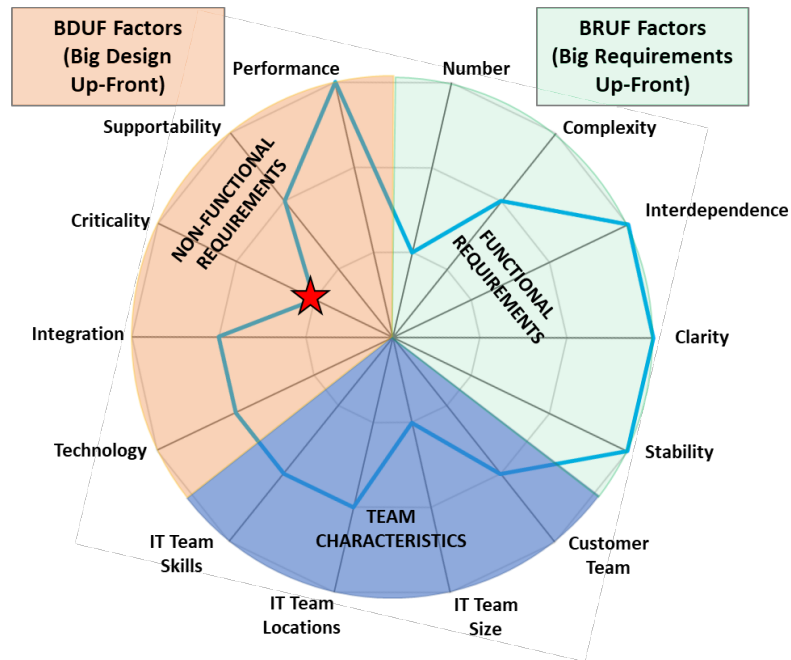


Figure 15. Radar Chart Showing a Project with Low Criticality Requirements

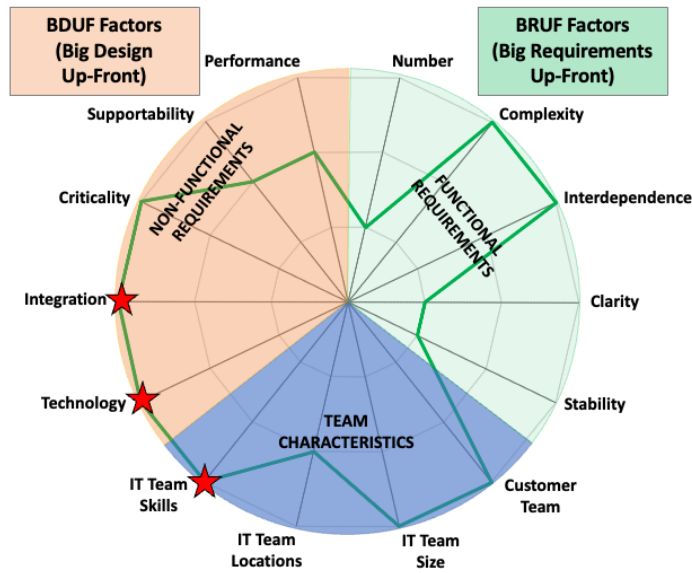


Figure 16. Radar Chart for a Project with High Integration, Technology, and IT Team Skills Requirements

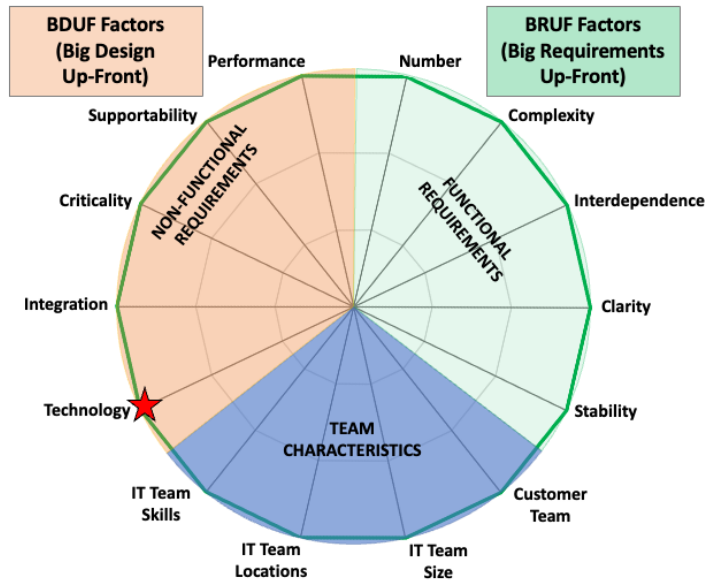


Figure 17. Radar Chart for Technology-Intensive Projects at a Large Financial Services Provider

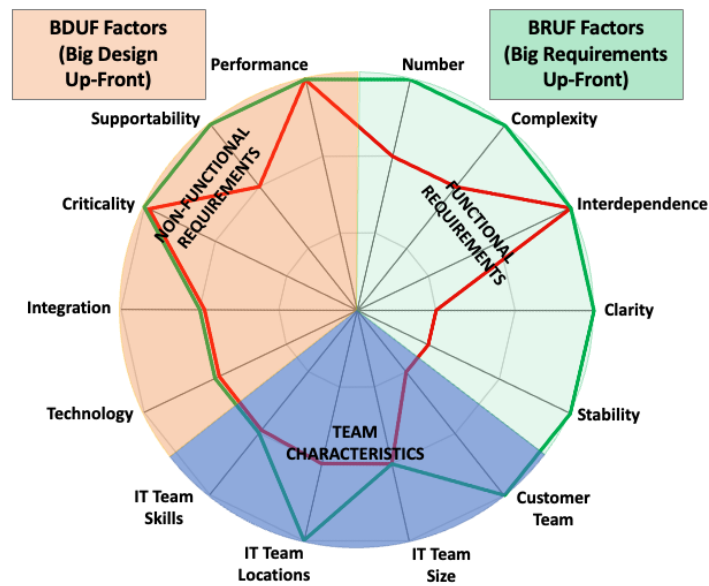


Figure 18. Impact of Heterogeneity of Subject Matter Expert Team on Project Characteristics

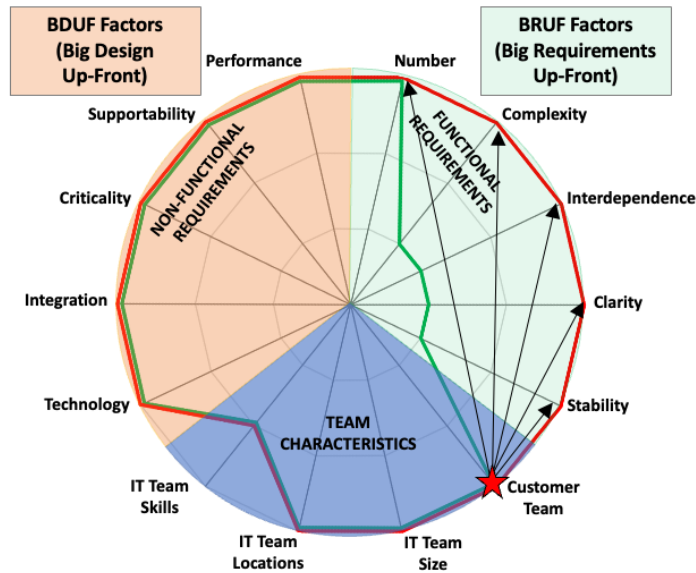


Figure 19. Impact of Customer Culture on Project Characteristics

This position was supported by an enterprise architect in a technology consulting firm that builds custom websites for external clients. This group emphasizes doing small projects that can be done in an agile fashion. These projects can be fairly agile with up to 15 team members, although 8 to 10 is preferable. Unlike the technology consulting group described in Section 4.1.1 Figure 6, this group accepts large projects. But, once team size reaches 25 or higher, they indicated that they have to “step outside the rigid rules of agile,” adapting those rules into a hybrid approach combining plan-driven and agile techniques. See Figure 20, with the arrow indicating the move in IT Team Size from that appropriate for agile to levels requiring hybrid techniques, leading to a profile change from the green line to the red line.

Note that some organizations will try to retain agility in larger projects by splitting a large team into several agile-sized teams. This can work well when the overall solution is structured as a series of small interacting applications (see the discussion of microservices in Section 4.2.2). But structuring that way then requires additional planning and coordination using such techniques as “scrum of scrums” (Rubin, 2013, pp. 218-220) or more elaborate frameworks, such as the Scaled Agile Framework (SAFe) (Knaster & Leffingwell, 2017).

4.3.3 IT Team Locations and IT Team Skills. Finally, we tackle the dimensions of IT Team Locations and IT Team Skills. We discuss these dimensions together because, while, high levels of one can arise independently of the other, and they also may interact with each other.

Starting with IT Team Locations, this pertains not just to the challenges of working in multiple locations – contrary to agile prescriptions – but also the complications of those locations, especially when a local IT team outsources some of its work to offshore resources: multiple time zones, multiple native languages and cultures, and/or multiple organizations (e.g., working with an outside technology vendor supplying consultants or contractors).

All of these make communication and coordination more difficult, forcing teams to become more plan-driven, especially with respect to BRUF. Furthermore, large IT Team Size leads to multiple IT Team Locations, for example, because a local team may struggle to recruit enough skilled IT personnel in a single office or city.

In contrast, high IT Team Skills point to the need for many technical or software development process skill sets, especially when the team currently lacks those skill sets. This often leads to team member specialization. It can arise in a straightforward manner independently of other dimensions; for example, recall the manufacturing team in Section 4.2.4 that needed to learn a new workflow and data integration platform utilized in an integration project.

Pointing to interactions between these dimensions, multiple IT Team Locations can also lead to IT Team Skills challenges, as expanding a local team to other locations can lead to the need to integrate new team members who initially may lack knowledge of the development tools, current application code, business requirements, or the development process that the local team utilizes.

This is illustrated by an IT program manager leading the development of back-office administrative systems for a high-tech manufacturer: his local team was in the United States, but he also was utilizing lower cost, offshore team members in India. He noted that he had to execute in a more plan-driven manner because of this and that his offshore team members were not all “superstars,” leading him to spend more time documenting requirements in a formal manner. The joint impact of offshoring to multiple locations is illustrated in Figure 21. The arrows indicate the increases in IT Team Locations and IT Team Skills, leading to a project profile change from the green line to the red line.

While the outsourcing example points to one way in which multiple IT Team Locations can exacerbate IT Team Skills needs, it is also true that high skills needs can lead to the need to locate the team in multiple locations, even when the IT Team Size is small.

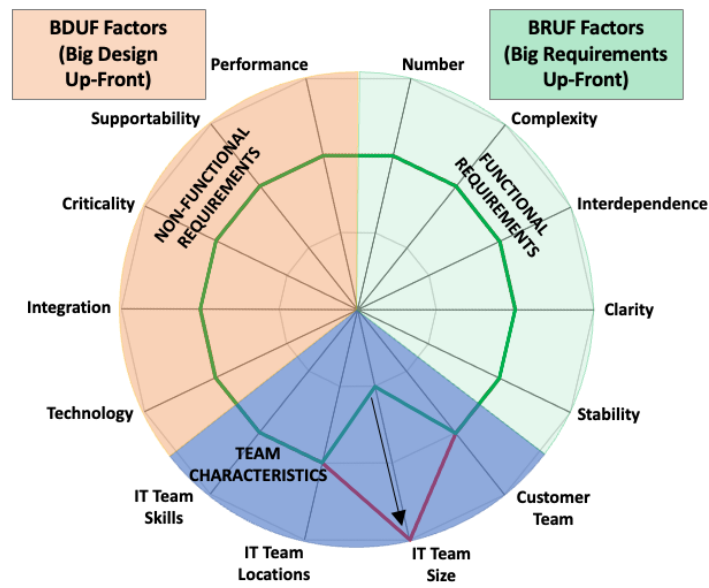


Figure 20. Impact of Team Size Increase on Project Approach Profile

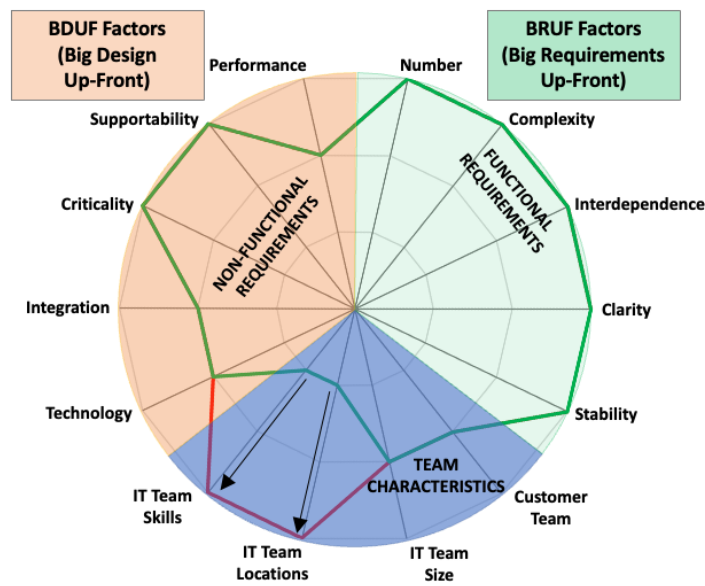


Figure 21. Impact of IT Team Skills and Location Requirements on Project Approach Profile

An example of this is a consultant running a big data project for a client. In this case, diverse, specialized skill set needs led to a small team of only 10 members being located in five different locations because of the need to recruit the “best of the best” in these skills.

5. VARIATIONS AND EXCEPTIONS TO THE EXTENDED HOME GROUNDS MODEL

The examples we have described illustrate that the extended home grounds model provides insights explaining IT professionals’ choices of the project approach in a wide range of projects impacted by numerous dimensions. Furthermore, as one of the respondents, a leader for a software development consulting group, confirmed, the reality today is that the approach chosen needs to be tailored to project circumstances based on a variety of factors, such as large versus small projects, characteristics of the development environment, team and client locations, and so on. The message is clear: there is no one best way to develop software. Rather, the optimal choice should balance agile and plan-driven techniques guided by key project characteristics.

However, that same development leader quipped the familiar aphorism: “All models are wrong, but some are useful.” That is to say, the extended home grounds model we have presented provides significant explanatory power, but there are exceptions, which we detail in this section.

5.1 Data Analytics

The plan-driven-agile continuum was best aligned with environments where transaction processing was the dominant application function. However, in three environments where data analytics was dominant, and transaction processing was nearly absent, neither plan-driven nor agile approaches made sense. Rather, for requirements, respondents did significant up-front planning for creating a data schema and importing data sources. Still, then the actual use of the populated data for analytics became much more ad hoc, typically without even the use of agile user stories of the form “As a <user>, I want/need to <do a software function> to <achieve a business goal>.” Furthermore, these types of projects tended to plan and execute their analytics work on a daily basis, without the use of fixed-cadence iterations or sprints. Here, traditional weeks-long sprints gave way to an “even more organic” approach than traditional agile practices.

5.2 Artificial Intelligence

Similar to the prior point, the model did not have a good fit with two cases with environments emphasizing the implementation of trainable AI. Both were executed in the context of data analytics, but it is worthwhile noting them as raising a more general issue of a need for new project execution frameworks in cases where machine learning methods begin to displace traditional human-driven coding of system functionality. One respondent stated, “There’s a new emerging need in the industry to do nontraditional coding methods, and that whole [regular software development] process gets thrown out the window.” Furthermore, this respondent noted that effectively combining the development of traditional and artificial intelligence systems is a difficult problem.

As such, explaining this is an evolving issue for the field of practice rather than a pedagogical one.

5.3 Making BRUF More Agile

While acknowledging the importance of BRUF in many project circumstances, two of the more experienced development leaders noted innovations in creating BRUF in an iterative manner aligned more closely with the development iterations or sprints in which the features would be constructed. One respondent called these repeated, small BRUF iterations “shadow sprints” – in essence, doing final elaboration of requirements details two to three sprints prior to the sprint in which the features were to be built. Another alluded to the same basic concept, using the phrase “rolling cycles” of requirements.

5.4 Software as the Product

One respondent argued that the framework made sense for systems that support a non-IT product or service. This would be the case for most systems contemplated in the examples. However, when the system itself is, in essence, the product, especially in online retail contexts, then a highly agile approach would be appropriate: “if you’re working for somebody like a Netflix ... I think ... pure agile works better for them because they have one product ... and they have to be up to date with the market constantly.” Note that this could be true of any customer-facing site that is a primary face of the organization and source of competitive advantage in directly selling and/or delivering products and services – for example, social media companies, retail websites, news providers, and so on.

5.5 Creating Commercial Off the Shelf (COTS) Software

Note that the exception described in Section 5.4 is highly distinct from COTS products providing transactional or data analytics functionality to external client companies. While these applications must compete against other COTS products in the same niche, in this type of situation, the need to consolidate feature requests from multiple customers and then deliver those features on a promised release timeline compels a much more conventional, essentially plan-driven approach. In essence, managing COTS software from a vendor perspective intensifies the need for plan-driven techniques by dramatically increasing the number and diversity of business customers.

5.6 Implementing Commercial Off the Shelf (COTS) Software

In addition to challenges of *creating* a COTS product from a software development perspective, we note that *implementing* highly configurable COTS systems, such as ERP and CRM systems, creates specific challenges not directly captured in the extended home grounds model. That is, given that the design of these systems already exists (rather than needing to be coded as part of the project itself) and that the design may be implemented in similar contexts hundreds of times, it is typically the case that such COTS systems are implemented using a well-defined, repeatable series of steps – an approach that is highly plan-driven, including not just requirements but also implementation. This was the case for one respondent who led implementations of a COTS capacity requirements planning (CRP) system.

But functional and non-functional requirements for these systems are already largely addressed by existing technology and application architectures. This makes the extended home

grounds model less applicable for software configuration projects than for software construction projects.

The takeaway is that some COTS implementations are among the most “pure” plan-driven system project approaches seen today. Even agile advocates acknowledge that agile approaches such as Scrum may not be a good fit in these circumstances (Rubin, 2013, p. 8). At the same time, extensive research on COTS-based enterprise systems implementation (as reviewed, e.g., in Ali & Miller, 2017) suggests that despite decades of research and practice in this area, there is still a need for better standardized implementation models (Ali & Miller, 2017, p. 23).

5.7 Organizational Culture

Finally, we address the argument introduced in Section 1.1 that the main barrier to adopting agile is stubborn resistance from traditional, non-agile organizational cultures. While the original Boehm and Turner (2004) home grounds model includes Culture as a key dimension, our extended home grounds model deemphasizes culture. In our model, culture exists as an aspect of Customer Team – “Customers value formal project planning and management” – and as an aspect of IT Team Skills – “Team needs training on the software development process.” In essence, our model focuses on key project characteristics that should rationally impact optimal project approach selection rather than focusing on traditional cultural values that do not provide a rational basis for that choice.

This is not to say that culture is unimportant in selecting and adopting software development project approaches. For example, in Section 4.3.1, we saw a software development leader who needed to tailor different project approaches based on customers' differing cultural values on different continents. We can also point to other examples where cultural values within the IT team impacted the conduct and success of the project.

For example, we recall the CEO in Section 4.2.2 who faced a “disaster” with a team pursuing a highly inappropriate agile approach in updating and extending a complex health insurance system. This CEO noted that, even in the face of project failure, several of his younger development team members in their 20s and 30s were deeply unhappy with a move to a hybrid approach.

From the opposite perspective, the respondent we highlighted in Section 5.6 implementing COTS CRP systems recounted his experiences in another context: a food and alcohol distribution company. This company operates in a market niche that has been relatively stable and somewhat insulated from competitive pressures. As such, the IT staff in this company was dominated by 50- and 60-year-olds who had grown up professionally with and were comfortable with plan-driven techniques. Not only did these mature professionals resist the introduction of agile methods, but their values also led to younger, agile-oriented IT team members leaving after a short period of time.

The message here is clear: culture can and does matter in customer and IT teams. However, based on the broad range of noncultural issues we explored, it is also clear that culture is not the main barrier to adopting agile methods in enterprise systems development. Rather, the objective dimensions of those projects in terms of functional requirements, non-functional requirements, and team characteristics compel IT leaders to choose plan-driven and hybrid approaches over “pure” agile approaches.

6. USE OF THE EXTENDED HOME GROUND AND RADAR CHART MODEL TO SUPPORT SOFTWARE DEVELOPMENT APPROACH SELECTION IN PRACTICE AND TEACHING

The complexity of our model, with its 14 dimensions, arises directly from the need to reflect the complexity of real-world systems projects. Because of this complexity, we have used nearly two dozen case studies to explain it.

In this section, we discuss general approaches for using a project radar chart to determine the optimal development approach for the project in the context of the extended home grounds model. This includes use both by IT practitioners and by teachers and students in MIS courses. We begin in Section 6.1 with a discussion of how to teach the rudiments of the model to both audiences. This includes utilizing models that are straightforward in plotting and interpretation.

With the rudiments of the model in hand, we then turn to how actually to utilize the models in more realistic situations. We begin by discussing use by practitioners in Section 6.2, then turn to use by students in MIS course settings in Section 6.3. We utilize this sequence because students are, after all, budding IT practitioners; thus, practitioner use forms a baseline that students can use in projects and classroom settings.

Throughout this Section 6, we reflect on our experiences in teaching the models to a variety of audiences.

6.1 Learning the Rudiments of the Model

Because of the complexity of the model, our experience with all audiences has been that the model takes some time to learn. For example, the meaning, impacts on the project approach, and interactions of each of the 14 project dimensions will typically require some explanation (for example, explaining the impacts of Clarity, Stability, and Interdependence as portrayed in Figures 7 and 9). Even more fundamentally, many individuals will not clearly understand the key distinctions between agile, plan-driven, and hybrid project approaches (as summarized in Figure 1). Furthermore, before utilizing the model in practical situations, individuals should practice plotting project dimension values on a radar chart using some straightforward examples.

In academic workshops we have run for information systems teachers (Spurrier & Topi, 2020a; Spurrier & Topi, 2020b), we found that communicating this information requires about an hour. In our own systems analysis and design (SA&D) classrooms, we have found that students can also grasp the rudiments of the model in a single 75-minute class; we do note; however, we taught this topic after the mid-point of the semester, after the students had learned the elements of the hybrid approach, including detailed requirements analysis, project planning, and sprint management.

We also note that we have found it useful to portray the radar charts utilizing web-based, shared-experience technologies. For example, we have utilized Google Sheets spreadsheets in which participants learning the model could plot their own project values against each project dimension in a series of project examples, with the plot emerging on a spreadsheet-generated radar chart. Our Google Sheets spreadsheet included multiple duplicate tabs, enabling multiple individuals or small groups to plot their values separately and then easily compare outcomes via screen sharing.

Project examples that we typically use to explain the model include the following:

- **Prototypical agile project:** Building a simple mentoring application designed to connect mentors and mentees, sharing noncritical textual data and supporting about 100 users. This results in low functional requirements (low BRUF), low non-functional requirements (low BDUF), and an expectation of utilizing an agile-style team.
- **Prototypical hybrid project:** The complete rewrite of a complex, legacy COBOL mainframe manufacturing system supporting dozens of factories with large numbers of users. The rewrite would require a new systems architecture based on JavaScript and the employment of new technologies, such as machine learning, unfamiliar to an existing, small systems team. This results in high functional requirements (high BRUF), high non-functional requirements (high BDUF), and high team requirements (from the need to expand team skills to a new tech stack and new technologies, as well as likely requiring a much larger team in multiple locations).
- **High BRUF/Low BDUF project:** Major functional feature upgrades to an existing health insurance claims processing system. The existing architecture is fine in terms of non-functional requirements. This results in high functional requirements (high BRUF) but low non-functional requirements (low BDUF). IT Team Skills are likely low (relative to current team capabilities), but it is likely that IT Team Size will increase, as may IT Team Locations.
- **Low BRUF/High BDUF project:** Creation of a social media platform supporting several straightforward use cases, including posting, uploading files, and “liking” and commenting on posts. The application, however, will contain highly confidential data and need to scale to millions of users. This will require low functional requirements (low BRUF) but high non-functional requirements (high BDUF). IT Team Skills will likely be high, in part because of the need to bring in the expertise of security and infrastructure specialists.

We note that the need to train participants in the rudiments of the extended home grounds model should not be considered a barrier to adopting the model. Rather, this training is valuable in itself, as it expands the thinking and understanding of those participants in making a comprehensive sense of project dimensions relevant to the optimal choice of approach for a given systems project.

6.2 Use by IT Practitioners

For practitioners, we suggest that they first chart their team or project’s situation against each of the radar chart dimensions. For each dimension, the answer as to whether the dimension level should be valued as high (i.e., plotting at the outer edge of the chart), low (plotting near the center of the chart), or medium (in between) should be answered by the practitioner in the context of their team’s situation: “Is this dimension high, low, or in between, by my own judgment, given my team’s circumstances and practices?” For example, for some dimensions, such as IT Team Size, absolute values are helpful; e.g., agile views teams numbered in the single digits as

compatible with agile assumptions (low, in our model), while teams moving upwards from double digits toward triple digits are not compatible (high, in our model). Similarly, whether a team is colocated or not is an absolute distinction.

But, as has been noted in other studies (e.g., Marek et al., 2021), teams do have some ability to adapt to impacts such as distributed and remote work. Thus, the assessment of the impact of (in this example) large IT Team Size and multiple IT Team Locations might be different for a practitioner whose team had become practiced, for example via communications tools and process techniques, at working in these circumstances, versus another practitioner whose team had previously been small and colocated. On the other hand, even with adaptations, we would not expect a team with over 100 members in multiple locations to ever be able to be as agile as a team of less than 10 members in a single location. In general, this recalls our discussion in Section 2.4 of Highsmith and his APM model (2010): teams can try to adapt to non-agile project circumstances using an agile mindset, but they will often still end up with non-agile, “heavyweight” project processes.

Handling the other dimensions would be similar: a “low” number of features for a team used to multimillion-dollar feature budgets (say, a global bank) could be similar to a “high” number of features for a team used to operating at a much smaller scale (such as a small, early-stage start-up). A team that has already implemented and used large-scale, robust architectures could evaluate objectively high non-functional requirements differently from a team that must dramatically expand its architecture in the face of those same requirements. As these examples demonstrate, the numeric values associated with the 14 dimensions will depend on the organization, its level of resources, its experience, its size, and its culture.

Once the chart is plotted, the practitioner should note the shape of the plot overall. In particular, the focus should be in areas where the model suggests that a less-agile approach is warranted. For example, in the upper right sector of the plot, are the functional requirements more aligned with agile emergent requirements (if plotting mostly near the center) versus with plan-driven big requirements up-front (if plotting mostly near the outer edge)? In the upper left sector of the plot, do the non-functional requirements of the project demand big design up front, or are they lightweight enough to justify a fully agile approach? Furthermore, are the Team Characteristics compatible with the Functional and Non-Functional Requirements Characteristics? For example, if Functional Requirements are low but IT Team Size and IT Team Locations are high, then that discrepancy might motivate a change in that team’s staffing posture.

In all of these dimensions, the practitioner can scan our examples for similar situations by visually comparing their team’s chart to examples provided. Note that functional and non-functional plots may be considered independently, given the different impacts those requirements have on requirements: BRUF versus BDUF, respectively. It is important to emphasize that the specific characteristics of the development approach cannot be derived arithmetically from the values in the radar chart – the chart is a visual aide that provides project leadership with a comprehensive overview of project characteristics and supports the ultimate decision process.

6.3 Use by Teachers and Students

The proposed radar chart approach can be used in a variety of courses related to systems development, including a general SA&D course in the core of the undergraduate IS major or an MSIS program, a capstone project course of these same program types, and specialized software project management courses in IS and software engineering. Also, advanced software development courses (particularly project based) can use the approach to support anchoring the projects better to a real or fictitious context.

One of the key goals of this paper is to encourage instructors of information systems courses in SA&D, systems development, and software project management to make sure that the following key topics and associated learning objectives are covered in courses that deal with answering the questions pertaining to selecting the optimal systems development approach:

- The role of the hybrid development approach as an alternative to plan-driven and agile approaches
- The project characteristics that can be used in the analysis of the fit between a development approach and the project
- The process for determining the development approach based on the project characteristics using the radar chart approach
- Project types with which the radar chart frequently does not work

As mentioned, the case studies and the radar chart may be employed in MIS teaching in at least two ways: in senior capstone projects and in lecture-based courses.

6.3.1 In Capstone Projects. First, when students are embarking on senior (capstone) projects – especially in team settings – they can plot and evaluate the characteristics of their project much as a practitioner would. In student settings, it is useful to realize that IT Team Skills may typically be high, given that the students frequently need to learn new technical skill sets, software development process skills, and teamwork. (This recalls the IT Team Skills project characteristic descriptors in Table 1.) On the other hand, IT Team Size will likely be low, while IT Team Locations may be low (for face-to-face settings) or high (for distributed learning situations).

As with IT practitioners, students and their teachers may compare the functional and non-functional requirements sections to the case study examples provided to find ones that generally are similar to their current project.

6.3.2 In Lecture-Based Classroom Settings. When used in a classroom setting – such as in a project management course or SA&D course – the models and case descriptions may be used to explain the factors that drive teams toward agile versus hybrid approaches. Depending on the course subject matter, it may be useful for the instructor to focus on specific major categories of the model. For example, in an SA&D course, the focus on functional requirements modeling may suggest focusing on that (upper right-hand) sector of the model, including case studies from Section 4.1. The instructor can then zero in on specific characteristics of interest and then utilize the case studies that specifically explain those dimensions. Similarly, when focusing on non-functional requirements, the instructor may wish to focus on case studies from Section 4.2.

Note that our own experiences have been in teaching the model after the mid-point of an SA&D course that incorporates some project management skill sets. We have also utilized our own SA&D textbook (Spurrier & Topi, 2021), which includes a chapter specifically focused on explaining the model, including a series of concrete examples illustrating use of the model. For instructors not utilizing our textbook, this paper may be used to support the coverage of the key aspects of the model. It may also be useful for instructors to note the impact of Team Characteristics on the modeling approach being taught, including case studies from Section 4.3. These Team Characteristics may be especially germane to project management courses. Again, for many students, IT Team Skills will be high in the sense that students are still learning modeling techniques and other technical skills. This may motivate a greater use of comprehensive BRUF modeling techniques in an SA&D classroom for a given set of Functional Requirements Characteristics than those same students might utilize later in their careers as experienced professionals.

Finally, it is our experience that the model is useful in helping students understand why learning the hybrid approach, including comprehensive requirements and associated up-front project planning techniques, still provides value, even in an era when agile techniques are especially popular. This recalls the McKendrick (2020) blog entry noted in Section 1.1 – in an era where uncritical belief in the near-universal superiority of agile methods often holds sway, and many students enter the SA&D course with a view that older, plan-driven methods inherently lack value and, therefore, are hopelessly out of date. To counteract this naive and incorrect viewpoint, instructors may, for example, use the model to analyze realistic situations where functional requirements are clear, stable, and highly interdependent. This can help students grasp that, in these kinds of situations, BRUF employed in the context of the hybrid approach does provide significant value and, in fact, may be necessary for project success. Using the model and the cases presented in this paper has also helped students understand the differences in the impact of BRUF and BDUF factors. This, in turn, gives the students more clarity regarding the difference between functional and non-functional requirements.

7. SUMMARY

A key message of these case studies is that there are many best ways to develop software. Furthermore, the optimal choice between agile, plan-driven, and hybrid approaches should be driven by a project's alignment with the extended home grounds model. In contrast, while culture sometimes influences how teams pursue software projects, it is not the main barrier to the widespread adoption of agile in enterprise-class projects. Rather, it is the recognition by IT development leaders that a wider range of techniques is appropriate and necessary in promoting the success of ESD projects. We hope that these case studies and the radar chart tool for analyzing the selection factors will provide instructors of IS courses focused on systems development relevant material that will help students understand the complex, heavily intertwined factors that should affect development approach choice.

More specifically, we believe the case studies support the view that enterprise software development is seldom executed in a purely agile fashion in the real world. This is not a failure, but, rather, happens because IT development leaders are

pragmatic and innovative in optimizing their software development approaches for any given project. Rather than strictly adopting any specific published agile or plan-driven approach, organizations actively tune their approaches based on a wide range of project and organizational characteristics, including functional requirements, non-functional requirements, and team characteristics. As such, organizations need meta-agility, meaning agility in selecting the optimal systems development approach depending on each project's characteristics. It is essential that the students graduating from computing programs understand from the beginning of their careers that one systems development approach does not fit all projects. We hope the material covered in this paper will help instructors convey this important message.

Furthermore, the extended home grounds model shows that, while software projects are among the most complex of human activities, they are not incomprehensibly so. Rather, by systematically applying the extended home grounds model using a radar chart, we can readily make sense of key project characteristics to choose the optimal project approach. This makes the extended home grounds model, together with the radar chart technique, a valuable teaching tool with which instructors can address one of the most fundamental decisions in an enterprise software development process: selecting the right development project approach.

In conclusion, over 20 years after introducing and popularizing agile software development methods, we see the value of those methods and their limitations. Using this model and these illustrative cases, we can teach students to focus on the reality and the principles behind optimally combining agile and plan-driven techniques in enterprise software development. Furthermore, we can equip those students to understand and succeed as they move into that reality as IT professionals.

8. REFERENCES

- Adkins, J. K., & Tu, C. (2019). Applying an Agile Approach in an Information Systems Capstone Course. *Information Systems Education Journal*, 17(3), 41-49.
- Agile Manifesto. (2001). Manifesto for Agile Software Development. <http://agilemanifesto.org>
- Ali, M., & Miller, L. (2017). ERP System Implementation in Large Enterprises – A Systematic Literature Review. *Journal of Enterprise Information Management*, 30(4), 1-30.
- Almudarra, F., & Qureshi, B. (2015). Issues in Adopting Agile Development Principles for Mobile Cloud Computing Applications. In *Procedia Computer Science*, 52, 1133-1140. <https://doi.org/10.1016/j.procs.2015.05.131>.
- Ambler, S., & Lines, M. (2012). *Disciplined Agile Delivery*. IBM Press.
- Ambler, S. (2014). *Examining the 'Big Requirements Up Front (BRUF) Approach.'* <http://agilemodeling.com/essays/examiningBRUF.htm>
- Augustine, S., Payne, B., Sencindiver, F., & Woodcock, S. (2005). Agile Project Management: Steering from the Edges. *Communications of the ACM*, 48(12), 85-89.
- Baird, A., & Riggins, F. J. (2012). Planning and Sprinting: Use of a Hybrid Project Management Methodology within a CIS Capstone Course. *Journal of Information Systems Education*, 23(3), 243-257.
- Baskerville, R., Pries-Heje, J., & Madsen, S. (2011). Post-Agility: What Follows a Decade of Agility? *Information and Software Technology*, 53(5), 543-555.
- Batra, D., Xia, W., VanderMeer, D., & Dutta, K. (2010). Balancing Agile and Structured Development Approaches to Successfully Manage Large Distributed Software Projects: A Case Study From the Cruise Line Industry. *Communications of the Association for Information Systems*, 27(1), article 21, 379-394.
- Beck, K. (1999). Embracing Change with eXtreme Programming. *IEEE Computer*, 32(10), 70-77.
- Beck, K. (2000). *Extreme Programming: Embrace Change (1st ed.)*. Indianapolis: Addison-Wesley.
- Boehm, B., & R. Turner. (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston: Addison-Wesley Professional.
- Cockburn, A. (2001). *Writing Effective Use Cases*. Boston: Addison-Wesley.
- Cockburn, A., & Highsmith, J. (2001). Agile Software Development, the People Factor. *Computer*, 34(11), 131-133.
- Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Boston; Addison-Wesley.
- Cram, W. A., & Brohman, M. K. (2013). Controlling Information Systems Development: A New Typology for an Evolving Field. *Information Systems Journal*, 23(2), 137-154.
- Dhir, S., Kumar, D., & Singh, V.B. (2019). Success and Failure Factors That Impact on Project Implementation Using Agile Software Development Methodology. In: Hoda, M., Chauhan, N., Quadri, S., & Srivastava, P. (eds) *Software Engineering. Advances in Intelligent Systems and Computing*, 731, 647-654. Singapore: Springer. https://doi.org/10.1007/978-981-10-8848-3_62
- Digital.ai (2021). *15th State of Agile Report: Agile Adoption Accelerates Across the Enterprise*. <https://stateofagile.com/#ufh-i-661275008-15th-state-of-agile-report/7027494>
- Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and Success Factors for Large-Scale Agile Transformations: A Systematic Literature Review. *Journal of Systems and Software*, 119, 87-108. <http://doi.org/https://doi.org/10.1016/j.jss.2016.06.013>
- Dingsøyr, T., & Moe, N. B. (2014). Towards Principles of Large-Scale Agile Development (pp. 1-8). Presented at the *International Conference on Agile Software Development*.
- Feng, S., & Salmela, H. (2020). Mapping IS Curriculum Research Areas: A Systematic Literature Review from 2010 to 2019. Included in the *Proceedings of the AIS SIG-ED 2020*.
- Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising Agile Methods to Software Practices at Intel Shannon. *European Journal of Information Systems*, 15(2), 200-213.
- Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley.
- Gemino, A., Horner Reich, B., & Serrador, P. M. (2021). Agile, Traditional, and Hybrid Approaches to Project Success: Is Hybrid a Poor Second Choice? *Project Management Journal*, 52(2), 161-175.
- Gerster, R., Dremel, C., Brenner, W., & Kelker, P. (2020). How Enterprises Adopt Agile Forms of Organizational Design:

- A Multiple-Case Study. *The Data Base for Advances in Information Systems*, 51(1), 84-103.
- Grady, R. (1992). *Practical Software Metrics for Project Management and Process Improvement*. Upper Saddle River, NJ: Prentice-Hall.
- Harb, Y. A., Noteboom, C., & Sarnikar, S. (2015). Evaluating project characteristics for selecting the best-fit agile software development methodology: a teaching case. *Journal of Midwest Association of Information Systems*, 1(1), 33-51.
- Hastie, S., & S. Wojewoda. (2015). *Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch*. <https://www.infoq.com/articles/standish-chaos-2015>
- Highsmith, J. (2010). *Agile Project Management: Creating Innovative Products*. Pearson Education.
- Jacobson, I., Spence, I., & Kerr, B. (2016). Use-case 2.0. *Communications of the ACM*, 59(5), 61-69.
- Knaster, R. & Leffingwell, D. (2017). *SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering*. Boston: Addison-Wesley.
- Landry, J., & McDaniel, R. (2016). Agile Preparation within a Traditional Project Management Course. *Information Systems Education Journal*, 14(6), 27-33.
- Larman, C., & B. Vodde. (2017). *Large-Scale Scrum: More with LeSS*. Boston: Addison-Wesley.
- Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises*. Boston: Pearson Education.
- Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Boston: Pearson Education.
- Leffingwell, D., Knaster, R., Oren, I., & Jemilo, D. (2018). *SAFe Reference Guide*. Scaled Agile, Inc.
- Leidig, P., & Salmela, H. (2021). *IS2020 A Competency Model for Undergraduate Programs in Information Systems*, AIS and ACM. <https://dl.acm.org/citation.cfm?id=3460863>
- Marek, K., Wińska, E., & Dąbrowski, W. (2021). The State of Agile Software Development Teams During the COVID-19 Pandemic. In: Przybyłek, A., Miler, J., Poth, A., Riel, A. (eds) *Lean and Agile Software Development*. LASD 2021. Lecture Notes in Business Information Processing, 408. Springer, Cham. https://doi.org/10.1007/978-3-030-67084-9_2
- McAvoy, J., & Sammon, D. (2005). Agile Methodology Adoption Decisions: An Innovative Approach to Teaching and Learning. *Journal of Information Systems Education*, 16(4), 409-420.
- McKendrick, J. (2020). Culture Still Keeps Eating Agile Software Strategies for Breakfast, *ZDNet* (blog), June 6, 2020, <https://www.zdnet.com/article/culture-still-keeps-eating-agile-software-strategies-for-breakfast/>
- Nelson, R. R., & Morris, M. G. (2014). IT Project Estimation: Contemporary Practices and Management Guidelines. *MIS Quarterly Executive*, 13(1), 15-30.
- Niederman, F., Lechler, T., & Petit, Y. (2018). A Research Agenda for Extending Agile Practices in Software Development and Additional Task Domains. *Project Management Journal*, 49(6), 3-17.
- Rubin, K. S. (2013). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Upper Saddle River, NJ: Addison-Wesley.
- Rush, D. E., & Connolly, A. J. (2020). An Agile Framework for Teaching With Scrum in the It Project Management Classroom. *Journal of Information Systems Education*, 31(3), 196-207.
- Sarker, S., & Sarker, S. (2009). Exploring Agility in Distributed Information Systems Development Teams: An Interpretive Study in an Offshoring Context. *Information Systems Research*, 20(3), 440-461.
- Schwaber, K. (1995). SCRUM Development Process. *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*.
- Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum*. Upper Saddle River, NJ: Prentice-Hall.
- Sharp, J. H., & Lang, G. (2018). Agile in Teaching and Learning: Conceptual Framework and Research Agenda. *Journal of Information Systems Education*, 29(2), 45-52.
- Sharp, J., & Lang, G. (2021). IS 2020 and a Snapshot of the Current State of Systems Analysis and Design. Presented in the *Proceedings of AIS SIGED 2020*.
- Sharp, J. H., Mitchell, A., & Lang, G. (2020). Agile Teaching and Learning in Information Systems Education: An Analysis and Categorization of Literature. *Journal of Information Systems Education*, 31(4), 269-281.
- Spurrier, G., & Topi, H. (2017). When Is Agile Appropriate for Enterprise Software Development? *Proceedings of the 25th European Conference on Information Systems, ECIS 2017* (pp. 2536-2545), Guimarães, Portugal: AIS.
- Spurrier, G., & Topi, H. (2020a). A Holistic SA&D Approach to Agile, Plan-Driven and Hybrid Systems Project Options. *Proceedings of the 26th Americas Conference on Information Systems, AMCIS 2020*, Salt Lake City, Utah: AIS.
- Spurrier, G., & Topi, H. (2020b). Selecting the Optimal Systems Project Approach. *Proceedings of the 21st ACM Annual Conference on Information Technology Education, SIGITE 2020*, Omaha, Nebraska: ACM.
- Spurrier, G., & Topi, H. (2021). *Systems Analysis & Design in an Age of Options*. Burlington, VT: Prospect Press.
- The Standish Group. (2015). "Chaos Report 2015." <https://www.standishgroup.com/store/services/chaos-report-2015-blue-pm2go-membership.html>
- 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.
- Wake, B. (2003). INVEST in Good Stories, and SMART Tasks. *XPI23*. <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>
- West, D., Gilpin, M., Grant, T., & Anderson, A. (2011). *Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today*. Forrester.
- Williams, L., & Cockburn, A. (2003). Agile Software Development: It's About Feedback and Change. *Computer*, 36(6), 39-43.

AUTHOR BIOGRAPHIES

Gary Spurrier is a visiting scholar at Bentley University. He



has been an Assistant Professor of Practice of Management Information Systems at the University of Alabama. Further, he has held many industry positions in information technology, including CIO, COO, project leader for enterprise-level software projects, commercial off the shelf (COTS) software product

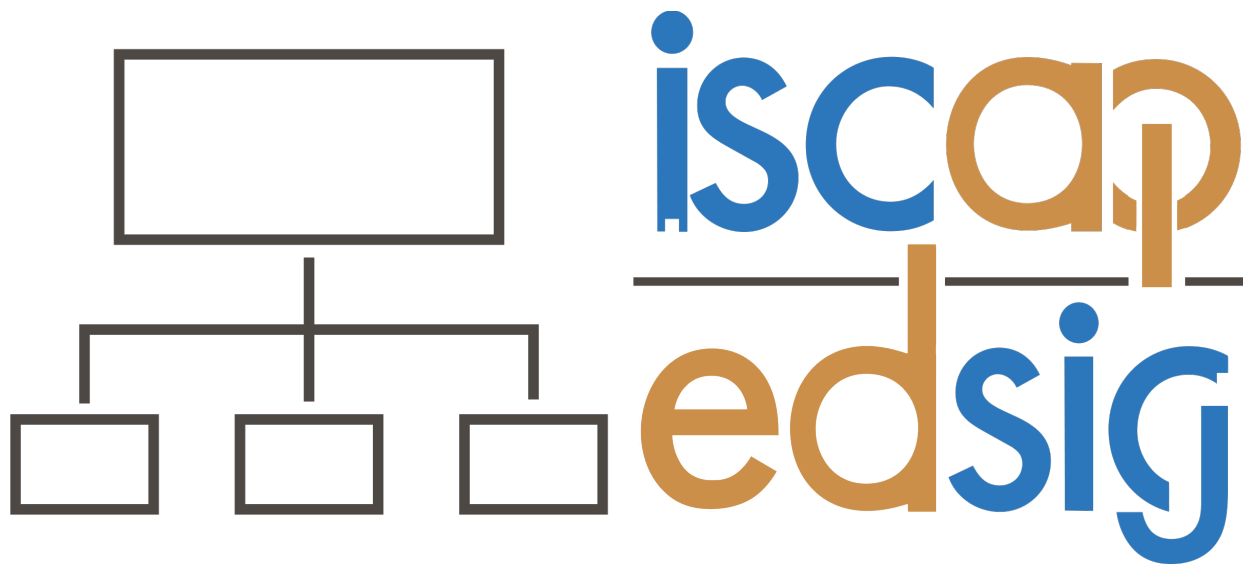
manager, and IT and operations consultant. He earned his Ph.D. in MIS at Indiana University-Bloomington. His research focuses on enterprise software development and systems analysis and design.

Heikki Topi is professor of computer information systems at



Bentley University. His Ph.D. in Management Information Systems is from Indiana University. His research focuses on systems development methodologies, information systems education, and human factors and usability in the context of enterprise systems. His scholarly output includes journal

articles, conference papers, large-scale edited volumes, textbooks, and curriculum recommendations (including *IS2010* and *MSIS2016* as task force co-chair). He has served the IS education community in multiple leadership positions, including AIS Vice President of Education.



**Information Systems & Computing Academic Professionals
Education Special Interest Group**

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the *Journal of Information Systems Education* have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

Copyright ©2023 by the Information Systems & Computing Academic Professionals, Inc. (ISCAP). Permission to make digital or hard copies of all or part of this journal for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial use. All copies must bear this notice and full citation. Permission from the Editor is required to post to servers, redistribute to lists, or utilize in a for-profit or commercial use. Permission requests should be sent to the Editor-in-Chief, *Journal of Information Systems Education*, editor@jise.org.

ISSN: 2574-3872 (Online) 1055-3096 (Print)