Chapter 3
Assessing Product Development Agility

Daniel X. Houston and Stephen W. Rosemergy

Abstract Agile software development grew out of a variety of alternative software development methods that shared a common set of values and principles. After two decades with these alternative methods, agile software development remains loosely defined, but has been widely accepted. This acceptance has gained the attention of other fields with discussions of applying agile to their work, for example agile systems engineering and agile program management. However, within the larger field of product development, agility was defined in terms of software development, both in practice and in principle. This chapter focuses on a set of general agile characteristics derived from the agile values and principles embraced by many software developers. This set of characteristics provides a basis for (a) assessing difficulties in software development projects employing agile practices, (b) applying concepts of agility to other disciplines beyond software development, and (c) measuring agility. In addition to deriving general agile characteristics, this chapter relates two stories of agile methods adoption that illustrate both the need for and the utility of general agile characteristics.

3.1 Introduction

According to the American Society for Quality, the quality movement can be traced to the trade groups of medieval Europe in which craftsman organized into guilds that used strict rules for applications of their crafts. During the industrial revolution, factories produced more specialized work and, in the late nineteenth century, factory planning became a discipline for increasing productivity. In the early twentieth century, process improvement was formalized with time and motion studies and statistical quality control. In the mid-twentieth century, emphasis shifted to the quality of both production and produced items in the Toyota Production System, Total Qual-

D.X. Houston (✉) · S.W. Rosemergy
The Aerospace Corporation, Los Angeles, CA 90009-2957, USA
e-mail: dan.houston@aero.org

S.W. Rosemergy
e-mail: steven.rosemergy@aero.org

M. Kuhrmann et al. (eds.), *Managing Software Process Evolution*,
DOI 10.1007/978-3-319-31545-4_3

ity Management, ISO 9000 series of quality management standards, Six Sigma, and Lean Product Development [1].

Viewed in the broadest context of development and production processes, contemporary system and software development share the ancestry of the quality movement. In balancing concerns for product quality, technical features, cost and timely completion, and productivity, emphasis has varied over the centuries, especially in the last century. The nascent agile movement that was underway in the early 1990s in business and manufacturing [13] had a coincident expression in alternative software development methods that later came to be grouped under the label "agile." This movement exhibited another emphasis in product development, one focused on flexibility and leanness [8]. Conboy [8] systematically developed a definition of agility based on these two concepts of flexibility and leanness. In this chapter, we take an alternative approach and develop general characteristics of agility based on experiences with the alternative software development methods that were distilled in the Agile Manifesto of 2001 [3].

3.2 Background and Context

The agile movement in product development has been fueled particularly by the field of software development. Software development was dubbed "software engineering" in 1968 and major advances in the ways of producing software took on the character of large engineering programs with the specification of requirements, design of architectures and details, and implementation followed by stages of integration and testing. The ability for an organization to develop software according to engineering methods was canonized in standards and in levels of capability maturity. However, the poorly understood dynamics of product development that challenges most engineering endeavors were especially troublesome in software projects, which—due to software's less tangible nature—seem to amplify the effects of "inadequate" prescriptive planning.

During the 1990s, some software developers reacted against the generally accepted engineering approach and tried various alternative practices and techniques. Methods such as eXtreme Programming, Scrum, Feature Driven Development, and Crystal Clear arose in this period, each with its own discipline for developing software. As these methods and their practices were published, software development groups began to embrace them. The authors of the various alternative methods convened in 2001 to produce the well-known Agile Manifesto, with a set of values and principles that called for a re-evaluation of software development processes. A later entry was Lean Software Development, which abstracted principles from the Toyota Production Systems and applied them to software development.

After 2001, the various alternative methods began to be referred to as agile methods with development groups referring to themselves as "agile." As the agile movement gained prominence, less and less attention was given to the disciplines underlying each of the methods. Thus the agile software development movement has exhibited a tendency toward homogenization of the different methods that gave rise to it. Today, agile software development is a mindset with a set of values, principles, and practices, but does not prescribe a particular process or set of processes.

With increasing acceptance of agile values, principles, and practices, several phenomena have occurred.

- Concept adaptation. In recent years, the idea of agile development has been applied widely, both within and outside the field of product development. Within product development, agile concepts have been applied to software requirements, systems engineering, product architecture, project management, and process improvement. Outside the field of product development, agile concepts have been applied to enterprises, business intelligence, supply chains, defense acquisitions, research methodology, and so forth.
- Agile precedents. Students of agile methods have found software development programs that preceded the current agile movement, but can now be described as agile. Duvall [11] provides eight examples of DoD programs that exhibited agile characteristics well before the agile movement in software development. Reagan and Rico [23] provide a similar list.
- Research growth. Because agile software development was largely a practitioner-led movement, it received almost no attention from academic researchers prior to 2001. Between 2001 and 2005, 36 empirical studies were found [12]. A 2012 study of agile methods research demonstrates growing research attention [10].

These phenomena all affect the meaning of "agile." Broad application of agile concepts has resulted in semantic inflation: agile development no longer refers clearly to the software development methods from which it arose. Similarly, searches for precedents have found agile characteristics in development programs of previous decades. On the other hand, research counters semantically inflationary effects by requiring clear definitions for the sake of answering questions such as "What constitutes agility?" "Under what circumstances is agility beneficial?," and "How does one become agile?" This chapter is motivated by the first question and seeks to address that question with derivation of a set of agile characteristics and a proposal for using the characteristics to answer other research questions.

3.3 Software Development Dynamics and the Need for Agility

The agile movement in software development arose out of need to harness the dynamics of software development beyond what software engineering had accomplished. The dynamics that drive product development projects out of control are amplified in software development because software is less tangible and unconstrained by physics. Therefore, functional specifications are more likely to over-reach what can be accomplished realistically with available resources while underestimation is more likely. Furthermore, functional changes are expected to be easier in software than in hardware. This section offers a brief explanation of software development dynamics

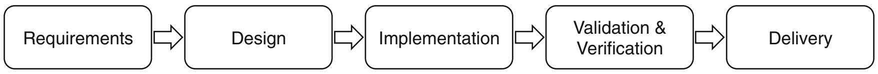


Fig. 3.1 A plan-driven process

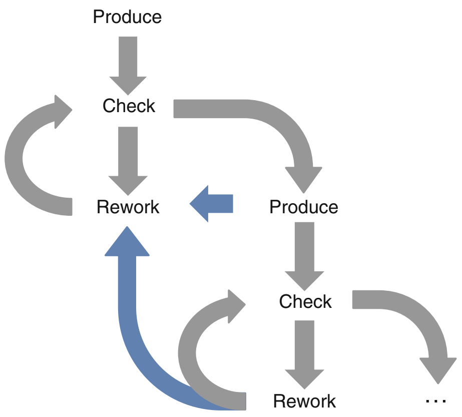
that is intended to demonstrate the need for the agile movement and provide some hints as to what allows agility to work.

In the engineering or plan-driven approach to developing software-intensive systems, plans are made for producing a system with specified functional capabilities and a sequence of steps is followed, at least at a high level (Fig. 3.1). However, Fig. 3.1 does not show the rework cycles, both within and between development phases.

Rework cycles in product development have been studied extensively in the System Dynamics Modeling community. The underlying problem with most product development plans is that they measure progress based on the plan and are unable to account for product quality and undiscovered rework in their progress measures. Consequently, quality shortcomings accrue until the need for rework delays progress. The product development rework cycle (Fig. 3.2) has come to be recognized as the central structure for modeling development projects because it is the most important feature for explaining project behavior.

Consider the rework cycle in a plan-driven process. For rework that is found in-phase, for example design errors found in the design phase, delays are incurred, but the cost of rework can be relatively low. For problems found in later phases (highlighted arrows in Fig. 3.2), the delays are much longer and the rework costs much higher. For example, a misinterpretation of a requirement that is not discovered until V&V testing means reworking artifacts in all phases from requirements through V&V. Changes to requirements during development produce rework that propagates through the development process. To complicate the project, delays put the project under schedule pressure. Developers working under schedule pressure are more likely

Fig. 3.2 The product development rework cycle



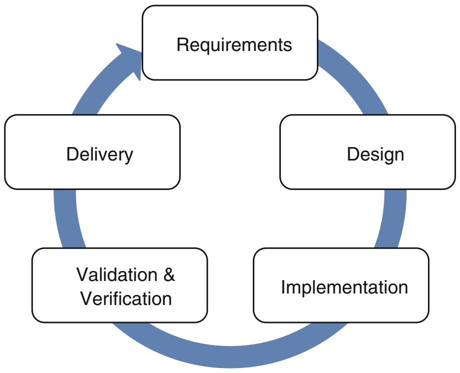


Fig. 3.3 An agile process

to make errors and to skip quality-inducing steps, such as peer reviews of work, thereby increasing the cycling of rework.

Agile software development, first and foremost, accepts changes as a fact of life and seeks to incorporate them in an ordinary workflow. Thus, the moniker, "agile." To accomplish the goal of an agile development process, the development cycle is scaled down to produce a smaller working product in a shorter amount of time. The sequence of phases is visited in every delivery cycle (Fig. 3.3). By approaching development incrementally and delivering an increment of system capability, say every few months, the rework cycles are dramatically shortened. Rather than building up schedule pressure over many months and incurring all its corrosive effects, delays are absorbed prior to each release by delivering only as much working functionality as possible. Thus, agile trades off a commitment to a delivery date against a plan-driven commitment to required functionality.

3.4 Development Challenges and Agile Methods

Since the Agile Manifesto, its proponents argue that the key to building better software is to view it not as a destination, but as a journey supported by underlying values and principles that deemphasize (but do not eliminate) practices and work products associated with project management best practices [7]. After more than two decades of discussion, debate, and informative evidence, we continue to debate the merits of agile software development. Both proponents and opponents agree that agile methods provide benefits in the forms of improved communication, team coordination, increased customer focus, less process overhead, and improved predictability [18].

Nonetheless, practitioners report issues (Table 3.1) that may be perceived as insurmountable challenges to agile software development teams, most notably project scaling, use of geographically distributed teams, cultural barriers to successful adoption, and applicability of agile principles to other technical domains [4, 8, 18]. Even though these challenges are reported often, examples of overcoming them success-

Table 3.1 Perceived issues that plague agile teams [17]

Issue	Description
Project scaling	Increasing team size also increases the amount of interaction required to coordinate, allocate, and integrate work between team members. Coordinating change across a large team is difficult [18]
Distributed teams	Frequent team interactions are not always possible with geographically distributed teams; remote teams lack the necessary accessibility to the product owner and are unable to develop and maintain the level of contextual expertise required to support the project [8, 18]
Culture change	Adoption of agile methods decentralizes day-to-day decision-making. Decentralized decision-making breaks down functional/hierarchical silos; organizational hierarchies are large impediments to decentralized decisions [19]
Technical domain	Agile methods are not applicable to non-software development or multidiscipline projects [8]

fully are available [6]. The following story illustrates the use of agile software development principles in addressing one of the most common challenges, geographically distributed teams.

3.4.1 Project Scaling and Geographic Distribution

Company A was a mid-size (8000 employees) software company that developed small-business software products. Based out of San Jose, California, they also employed an offshore team located in Hong Kong. This team provided specialized expertise in support of product internationalization. The remote team used Scrum very successfully for integrating application content and layout to support foreign language usage in U.S. markets. The San Jose team was happy with both the responsiveness and quality of the work delivered by the remote team. With the expanded language support of their products, demand for products tailored to locales outside the United States increased.

3.4.1.1 Transitioning to Global Product Development

Because Company A architected their system as a product line, whereby core assets could be quickly applied for new variant products [15], they were confident that their product was well positioned to address global markets. Having proven their ability to support Internationalization, Company A expanded the scope of the team in Hong Kong, to address International markets, starting with Asia (Fig. 3.4).

The two teams met in San Jose, agreed to continue using Scrum for their development method, e-mail and Skype for collaboration, and a common infrastructure for

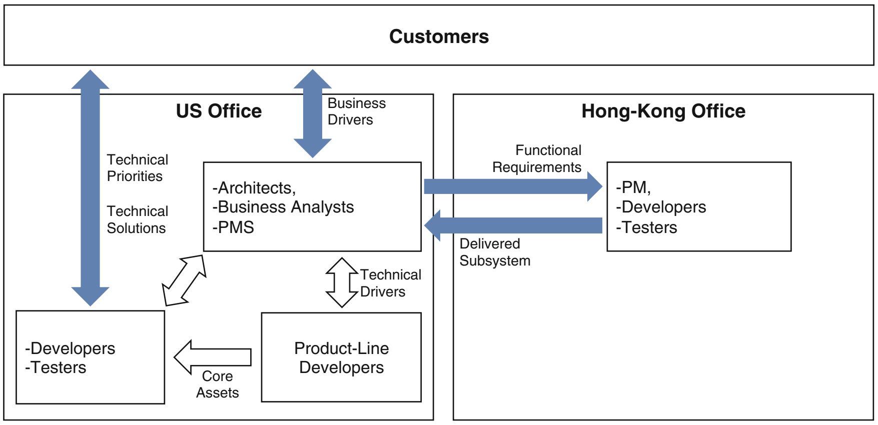


Fig. 3.4 Company A globalization

storing source code and documentation, instead of transferring files between sites. The San Jose team would host daily meetings, and because of time-zone differences, the Hong Kong team would shift their workweek by one day (Tuesday through Saturday).

3.4.1.2 Global Software Development Challenges and Agile

Although work progressed on schedule for each of the teams, trouble began as the San Jose team integrated and tested the software developed by the remote team. As both the frequency and severity of problems rose, tension between the two teams mounted. Daily standup meetings increased in length from 20 min each to 1.5 h, with most of the time devoted to reporting on status.

In trying to understand their problems, the company initially assumed that the scope of their endeavor (large-scale software development) and geographically distributed development were a mismatch with their agile method. They called for a face-to-face meeting of key contributors in San Jose where the two teams gathered at a local hotel to share their concerns. Their findings, which were consistent with other software development companies managing geographically distributed projects [19], did not point to agile development practices as the source of their problems. Rather, they indicated a failure to adhere to the principles of agile software development as they expanded their efforts. Specifically, the San Jose team had ignored the impacts of their locale-specific organizational constructs and delegated responsibility in what seemed the most expedient manner, not cognizant of the effects on the remote team. Changes were neither well-coordinated nor welcomed across teams, and interactions between teams became increasingly transactional, with emphasis placed on status rather than cooperation and collaboration.

3.4.1.3 Addressing the Challenges

Company A soon realized that in order to be responsive to customer needs, both locally and internationally, they needed to realign their efforts. In doing so, they evaluated their organizational structure, project responsibility partitioning, and project infrastructure in view of the agile software development principles and values.

They found that by scaling the teams in the most expedient manner, that is dividing responsibility functionally, instead of organizing around motivated individuals, they had inadvertently formed organizational barriers to communication and collaboration. Furthermore, collaboration and tacit knowledge transfer between the remote team and the customer was no longer practical because they had placed an intermediary between the remote team and the customer. So while both the local and remote teams embraced the principles of frequent delivery, face-to-face communications, and measured and constant progress, the remote team became information-constrained and were trusted only to organize themselves around the functions they were to deliver to the project. Both teams depended on each other to deliver, but neither team, particularly the remote team, had the authority or access to evolve the requirements, architecture, and designs.

After realizing their mistake, Company A revised its organizational structure (Fig. 3.5). The new organizational model established three distinctly separate development teams, one at each locale, and a product-line team, managed by a single team distributed across the two locales. In addition, Company A co-located architects, business analysts, and project manager in each locale—with responsibility partitioned by customer product, rather than by functional responsibility [20].

To anchor implementation efforts across teams, the Product-Line Development team established a continuous integration environment and collaboration environment to link communication, configuration management, and testing, and deployment of software releases [20]. Scrum Master roles were tailored to facilitate cross-team

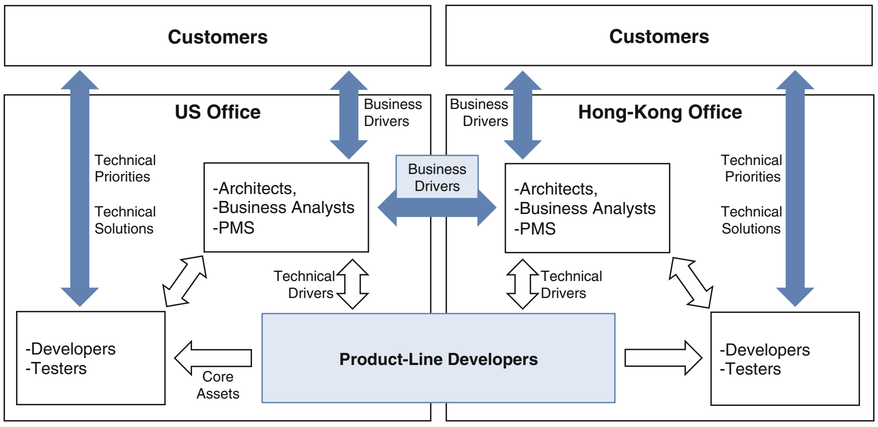


Fig. 3.5 Company A's revised organizational model

coordination and collaboration, surface problems during coordination meetings, and to remove barriers [2].

The preceding story demonstrates that impediments to agility can be difficult to identify, much less, solve. Had this organization not revisited the principles and values of agile software development, they might have concluded incorrectly that agile methods could not work for their business or that globalization was incompatible with their business objectives. Not all software development is well suited to agile practices and not all software organizations are disposed to employing agile methods. Nonetheless, the story suggests that periodically revisiting principles of agile software development can help a software business using agile practices recognize ways in which development problems can be addressed. Taking this a step further, we propose that the values and principles of agile software development can be generalized so that they can be applied to other disciplines in product development. The following section pursues the question of the nature of product development agility in an effort to identify a set of general agile characteristics derived from software development experience.

3.5 The Nature of Product Development Agility

The task of characterizing product development agility can be pursued in different ways. Conboy [8] takes a conceptual approach to developing a definition and taxonomy of agility by starting with its conceptual underpinnings and progressing through 16 steps. This chapter takes another approach that builds on the distilled experience of agile software development. The agile software development values and principles are distillations of the experiences of the practitioners of agile methods. Although the methods, and the practices that comprise them, are the building blocks of the agile software development movement, the values and principles have provided the movement a unifying identity.

3.5.1 Agile Values, Principles, and Practices

The self-dubbed agile alliance defined itself through values and principles. These were published as fixed lists [3], in contrast with practices often embraced by agile methods. Although lists of agile practices are available, the lists are not definitive because agile software development is not limited to any particular practices. In fact, whatever practices promote agility in a given circumstance may be regarded as agile practices. Furthermore, some agile practices originated decades earlier in the history of software development. Therefore, any published list of agile software development practices remains open-ended, guided by the values and principles as well as empirical success. For this reason, we focus on the values and principles for deriving general characteristics of agility.

Table 3.2 Agile software development principles

Issue	Description
Continuous value delivery	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
Welcome change	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
Frequent delivery	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
Business-developer collaboration	Business people and developers must work together daily throughout the project
Motivation centricity	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
Face-to-face conversation	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
Progress measure	Working software is the primary measure of progress
Constant pace indefinitely	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
Technical excellence	Continuous attention to technical excellence and good design enhances agility
Simplicity	Simplicity—the art of maximizing the amount of work not done—is essential
Self-organizing teams	The best architectures, requirements, and designs emerge from self-organizing teams
Reflect and adjust	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Table 3.2 lists the principles of agile software development, preceded by phrases used in Table 3.3. Table 3.3 relates the values and principles, indicating the degree to which the principles address or explicitly support the values. For example, "continuous value delivery" strongly supports "working software." Séguin et al. [25] performed a similar assessment of correspondence between the principles and values. In terms of correspondence, Table 3.3 agrees with their results in 96 % of the cells.

Table 3.3 indicates that the values are not supported equally by the principles. Not only does each value statement represent a prioritization, but the set of principles represents a prioritization of the four values: "individuals and interactions" and "working software" are more supported by the principles than "responding to change" and "customer collaboration."

Agile software development started with practices that sought to improve the production of software. Some of the practices, such as pair programming and story point estimation, were created to satisfy a specific objective. Others, such as iterations and test-driven development, were refined from ideas used in early computer program-

Table 3.3 Agile values and principles matrix (●: major, ◐: moderate, and ○: minor support)

	Agile software development values			
Agile software development principles	Individuals and interactions over processes and tools	Working software over comprehensive documentation	Customer collaboration over contract negotiation	Responding to change over following a plan
Continuous value delivery		●	◐	◐
Welcome change			●	●
Frequent delivery		●		◐
Business-developer collaboration	●		◐	
Motivation centricity	●			
Face-to-face conversation	●		○	
Progress measure		●		
Constant pace indefinitely	◐		◐	
Technical excellence		●		
Simplicity		●		◐
Self-organizing teams	●	◐		
Reflect and adjust	●			◐

ming. Still others, such as coding standards and software configuration management, were simply included from accepted software engineering practice. As practices were created or appropriated, and refined, they were collected and their use integrated into methods or processes. Agile software development practices continue to evolve, guided by the values and principles.

With alliance of the “agilists” and identification of the various alternative software development approaches as “agile,” the practices formerly identified with each method have become pooled as agile software development practices. Consequently, when members of a software development group describe themselves as “agile,” they must further explain the practices they employ. Referencing a specific agile method may be helpful also.

3.5.2 Deriving General Agile Characteristics

The matrix of Table 3.3 indicates intersections that can be aggregated and abstracted to produce general characteristics of agility beyond software development. Abstracting these characteristics should also remove overlaps in the values and principles. The following list of general agile characteristics (GAC) was abstracted from the agile values and principles.

- Interpersonal interaction
- Working product or service
- Customer/user collaboration
- Responsiveness to change
- Continual delivery of customer value
- Self-organizing, multifunctional collaboration
- Leadership by the motivated
- Technical excellence and simplicity

Table 3.4 uses a checkmark (✓) to relate these characteristics to the agile software development values and principles (a) to demonstrate that the characteristics cover the values and principles, and (b) to define the meaning of each characteristic in terms of the values and principles.

3.5.3 Comparison of General Agile Characteristics with Other Sources

Turner [31] has also produced a list of key characteristics of agile software development though he does not provide a derivation for his list.

- Learning attitude
- Focus on customer value
- Short iterations delivering value
- Neutrality to change (design processes and system for change)
- Continuous integration
- Test-driven (demonstrable progress)
- Lean attitude (remove no-value-added activities)
- Team ownership

This list compares well with the preceding list, though the two lists have a few differences. Turner's list does not explicitly include product characteristics of technical excellence and simplicity, but it does include "learning attitude," which may refer to learning about both the product under development and the development processes employed. Also, Turner's list does not mention leadership motivation. His list does introduce lean attitude as a willingness to remove non-value-added activities.

Table 3.4 General agile characteristics defined from agile software development values and principles

	General agile characteristics						
Agile software development values and principles	Interpersonal interaction	Customer/user collaboration	Responsiveness to change	Continual delivery of customer value	Self-organizing, multifunctional team	Leadership by the motivated	Technical excellence and simplicity
Individuals and interactions	✓	✓			✓	✓	
Working software				✓			✓
Customer collaboration	✓	✓	✓	✓			
Responding to change		✓	✓		✓		
Continuous value delivery				✓			
Welcome change		✓	✓				
Frequent delivery				✓			
Business-developer collaboration	✓				✓	✓	
Motivation centricity					✓	✓	
Face-to-face conversation	✓						
Progress measure				✓			
Constant pace indefinitely	✓	✓		✓			
Technical excellence				✓			✓
Simplicity	✓						✓
Self-organizing teams	✓				✓	✓	
Reflect and adjust	✓		✓		✓	✓	✓

Diebold and Zehler, Chap. 2, also produced a list of characteristics that they claim describe all known agile methods. They say that these characteristics are based on principles defined in the Agile Manifesto, but do not offer a derivation.

- Self-organizing teams
- Evolutionary development with short iterations and release cycles
- Active involvement of the customer with feedback
- Simple reactions and quick changes without formal change requests
- Simple design
- Test as central point in the development

This set of characteristics compares well with the derived set of GAC. With the exception of Interpersonal Interaction and Leadership by the Motivated, a one-to-one correspondence can be drawn between the derived general characteristics and Diebold and Zehler's set.

Conboy's [8] Taxonomy of Information Systems Development (ISD) agility provides another example of a set of characteristics of agility.

1. To be agile, an ISD method component must contribute to one or more of the following:
   a. Creation of change
   b. Proaction in advance of change
   c. Reaction to change
   d. Learning from change
2. To be agile, an ISD method component must contribute to one or more of the following, and must not detract from any:
   a. Perceived economy
   b. Perceived quality
   c. Perceived simplicity
3. To be agile, an ISD method component must be continually ready, i.e., minimal time and cost to prepare the component for use.

Conboy derived his taxonomy rigorously from definitions of "leanness" and "flexibility" rather than from agile values and principles. Consequently, it has a different structure than the previous sets of characteristics of agility. Nonetheless, it provides a useful set of characteristics for comparison.

Comparing the four sets of agility characteristics, several observations can be made.

- Only the GAC explicitly lists "interpersonal interaction," a strong motivator in the agile software development movement for increasing agility by reducing documentation. Table 3.3 illustrates that this value underlies six of the principles, so the other sets of characteristics likely treat this implicitly as an enabler of other characteristics.

- Turner [31] and Conboy [8] each include similar characteristics that the other two sets do not include. Turner includes "lean attitude (remove no-value-added activities)" and Conboy includes "perceived economy." These do not trace directly to agile values and principles, but do trace to an agile method, Lean Software Development, and to one of Conboy's starting concepts, "leanness."
- Another characteristic that Turner and Conboy include, but is not included in the other two sets, is learning: "learning attitude" (Turner) and "learning from change" (Conboy). In addition to Conboy's derivation, this characteristic is traced to the agile principle of "reflect and adjust."

3.6 Agility and Other Endeavors

A general set of characteristics provides a basis for discriminating between conformance and nonconformance to an ideal: a product development program can be described as agile to the extent to which it exhibits the characteristics. Thus, one can use such a set of characteristics to assess, at least qualitatively, whether a development program is behaving as an agile program is expected to behave. Because the set of characteristics is generalized, this included not only software development, but also other types of development programs. Furthermore, the set of characteristics could serve as a basis for developing a quantitative measure of agility.

To illustrate how we can use these characteristics to evaluate the degree of agility, we will examine the use of agile methods in the context of another domain: highly regulated systems development.

3.6.1 Development Process Agility in Highly Regulated Environments

Company B was a small (200 employees) bio-tech software start-up based in North America. Their primary product offering was an enterprise medical informatics diagnosis and digital record keeping software system. In spite of its relative size, Company B dominated the clinical informatics industry by virtue of its patented high-performance image streaming and business process automation technology. Company C was an established international medical device company, with over 100,000 employees worldwide. In addition to medical imaging devices, Company C sold enterprise business process automation tools, similar to that of Company B. However, Company C's product offering was not competitive due to Company B's patented streaming technology, which gave Company B a market share advantage of more than 40 % over Company C.

In order to gain market share in the clinical informatics industry, Company C acquired Company B. Company C had a reputation for delivering high-quality, innovative products to the marketplace. Moreover, because the European medical industry is highly regulated, Company C had established corporate-wide engineering policies, documented practices, and work product standards that ensured both transparency and compliance with IEC 62304 standards for medical device software development [14]. In contrast, Company B had used test-driven development (TTD) methods, with an emphasis on test automation. Having already built a testing infrastructure to accommodate both regression testing and just-in-time product enhancements, Company B delivered new capability for customer evaluation every four weeks. Company C, concerned for the success of their acquisition and the perceived risk of delivering new capability compliant with IEC and ISO standards, (a) investigated the possibility of imposing any mandates on the TDD team and (b) sought to learn from the software development successes of Company B.

Through interviews with Company B's software development team, and assessments of their product related work products, Company C found the following strengths:

1. While company B placed less emphasis on developing detailed documented requirements, they were able to trace driving requirements from user stories, to test cases, to documented design decisions (through both their feature tracking tools and the source code), and finally to test results.
2. Architectural decisions and constraints, while discussed only in face-to-face forums, were well understood by all internal stakeholders (well beyond the software engineers).
3. Product implementation followed establishment of requirements mandates and constraints through the creation and execution of tests, each of which served as a mechanism for demonstrating technical progress and achievement of both quality attributes and functional requirements alike.
4. Customers drove feature innovation, based on real-world use and evaluation of prototypes. Company B's mechanism for evaluating product features with customers, early and often, pruned unimportant features from the product line.

On the other hand, Company C found that although Company B products were not subject to medical device regulatory standards (ANSI, AAMI, IEC, and ISO) [24], they were not compliant with corporate IT safety standards (IEC 60950-1). Also, they found no mechanisms for demonstrating compliance with regulatory requirements if they chose to integrate their medical imaging products directly with Company B products.

3.6.2 Addressing Regulatory Concerns with an Agile Process

Company C's evaluation of Company B's practices and work products found that B's practices served as a motivating force for innovation, collaboration, and the delivery of both high-quality and marketplace-relevant products. They also determined that dismantling B's approaches could put the company acquisition at risk. The biggest

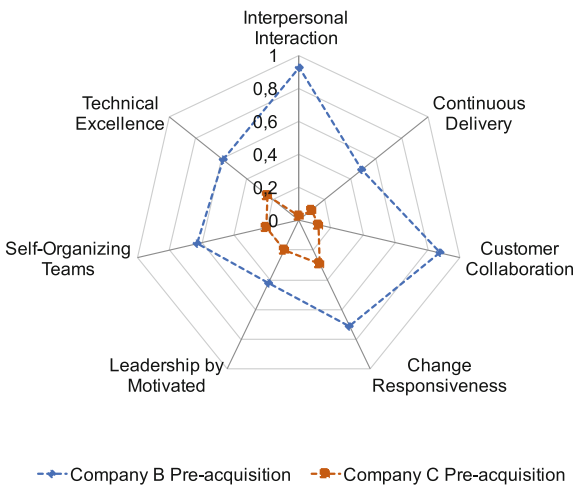


Fig. 3.6 Pre-acquisition company product development comparison using general agile characteristics

challenge they faced was incorporating ISO and IEC medical device safety compliance standards, so they approached B's development team and asked them how they could demonstrate compliance with these regulatory standards while maintaining the general characteristics of agility.

Company B's development team reviewed both the regulatory standards and Company C's compliant practices against the general agile characteristics. Next, the team engaged Company C's compliance experts and developers to assess all development practices (of both companies), using the agile general agile characteristics (Fig. 3.6). Understanding that relative agility was not reflective of product quality, together they found significant differences with respect to development practices, each of which either promoted or inhibited the general agile characteristics. Next, the team evaluated the regulatory constraints and requirements to determine the extent to which they might inhibit (or possibly promote) development agility. Together they determined the following points.

1. They could, with careful attention, maintain performance that exhibited all the general agile characteristics.
2. TDD provided the infrastructure for demonstrating compliance with regulatory requirements but required some enhancements to be fully compliant.
3. With the help of good coaching and an embedded subject matter expert, they could demonstrate traceability to regulatory and safety standards at each delivery, and the acquiring organization could adopt agile practices.
4. Integration with external safety critical software/hardware would need to be decoupled architecturally to ensure that their certification would not impede the deployment of new products.

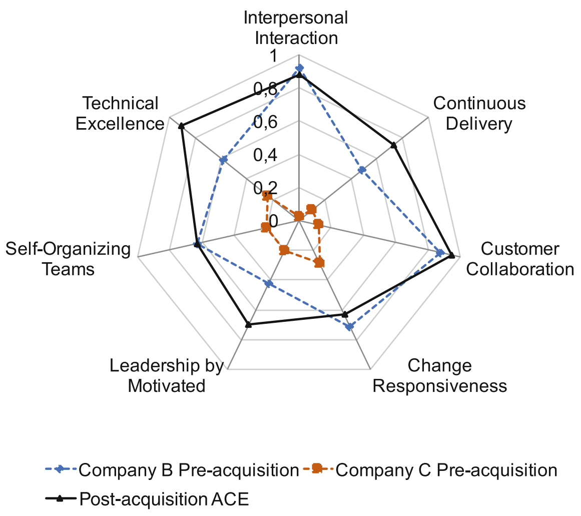


Fig. 3.7 Pre-acquisition and post-acquisition company product development comparison using general agile characteristics

3.6.3 Epilogue: Further Adoption of Agile Approaches

After finalizing the acquisition of Company B, Company C created a new business unit to house its medical informatics product suite, and then incrementally migrated its existing customer base to the acquired product suite. Within two years, the financial performance of the new business unit eclipsed the combined performance of all Company C's other medical business units.

Under stakeholder pressure to improve the financial performance of the other business units, Company C embarked on a two-year plan to adopt agile methods on all software-intensive systems. To this end, they created an Agile Center of Excellence (ACE) led by a long-time Company C leader. Because the company was more than 100 years old, this action met with initial resistance and distrust. After four years of coaching and mentoring both leaders and individual contributors (more than 5000 employees), the company has strongly embraced agile approaches to product development, as shown in the results of an assessment against general agile characteristics (Fig. 3.7). In terms of business value, the company attributed its profits to their "Agile Renewal."

3.7 Measurement of Agility

The preceding story demonstrates the value of using GAC as a basis for measuring agility. Measurement of agility has been a topic of discussion in product development [29] and production research [30] for over a decade. In software development circles, a number of agility measures have been discussed in various forms and for various purposes.

- Datta [9] has proposed an Agile Measurement Index based on five software project dimensions, for use in selecting a development methodology.
- Bock [5] suggests Agility Index Measurements for comparing capabilities of software development teams using seven scales.
- Seuffert [26] uses an 11-item questionnaire to measure degree of agile adoption.
- Kurian [16] produced a fuzzy model for measuring agility based on seven project characteristics.
- Lappo and Andrew [17] categorize agile goals and offer an example of collecting data for assessing a goal.
- Shawky and Ali [27] produced a measure of change rate, or entropy as an indicator of agility.
- Qumer and Henderson-Sellers [22] developed a four-dimensional framework (4-DAT) for evaluating the degree of agility in agile methods.

Of these six proposals, the first, the fourth, and the seventh hold the most promise. They are based on project characteristics, recognize degrees of agility, can produce leading indicators, and can be extended beyond software development. However, the first of these three have two shortcomings: (1) neither use characteristics that have been verified as dominant variables for measuring product development agility; and (2) the scales and mathematical models employed by each require validation for their ability to produce meaningful measures. Using the Agile Manifesto as a starting point, Tables 3.3 and 3.4 have sought to address the first shortcoming with a set of characteristics that are clearly traceable to a widely accepted set of values and principles that define agile software development. The second aforementioned shortcoming remains to be addressed.

The seven proposals indicate needs for measuring agility and hint at some of the potential benefits. One of the benefits would be overcoming the popular misconception of a binary approach to agile development: either a development organization is agile or it is not. Measureable definitions of agility would recognize that organizations demonstrate degrees of agility and would facilitate discussion of those degrees.

Another benefit of measuring agility is technical definition. “Agile” is a word so broadly used that its meaning has been overly inflated. Ironically, it fails to carry substantial meaning for people who must manage technical development processes. Measurement of agility would provide a technical basis for the term and support clear communication about the merits, shortcomings, and suitability of development processes. Measurement of agility would lend objectivity to a number of practical concerns, from guiding and supporting process improvement decisions, to choosing a development method for a specific project, and to choosing the best group for a development project.

Each agile software development method is usually recognized by its practices, but practices may be modified to fit a particular circumstance (a combination of development organization, customer, software type, product domain, contract, regulations, and so forth). In a multi-case study of Scrum projects, Diebold and Zehler, Chap. 2, found deviations, variations, and adaptations of Scrum. When such variations are undertaken, the question may arise as to the ability to perform agilely. As the preceding story illustrates, measuring agility from a set of characteristics can produce valuable results.

3.8 Conclusion

Product development always requires balancing concerns for cost, duration, features, and product quality. Although business and manufacturing had begun developing agile production concepts, the authors of the agile manifesto took a step forward by producing a set of values and principles based on a decade of experience using various alternative software development practices and methods. From those values and principles we have distilled a set of general agile characteristics and demonstrated the usefulness of these characteristics in facilitating software-intensive systems success. As general characteristics, they can be applied to other product development domains. More importantly, they provide a basis for judging the agility of a particular development process. The stories in this chapter suggest that qualitative assessments are the usual means of judging process agility, but some work has pursued quantification. More work is necessary to develop good measurement scales based on general agile characteristics.

3.9 Further Reading

Balancing Agility and Discipline: A Guide for the Perplexed, by Barry Boehm and Richard Turner, shows that agile and disciplined methods lie on a continuum. They have worked out guidelines for determining where on the continuum a project lies and how agile or disciplined a method must be.

For readers interested in degrees of agility, we recommend the following works cited in the References section: Conboy [8], Chow and Cao [6], and Qumer and Henderson-Sellers [22]. Qumer and Henderson-Sellers [21] provide more background on the 4-DAT analytical framework for evaluating methods from the perspective of agility in their article. Sheffield and Lemétayer [28] discuss factors that indicate software project agility and project success.

“The Right Degree of Agility in Rich Processes,” by Diebold and Zehler, is the Chap. 2 in this volume. It discusses two approaches, evolutionary and revolutionary, to integrating of agile software development practices into a structured process.

References

1. American Society for Quality: ASQ history of quality. Available from http://asq.org/learn-about-quality/history-of-quality/overview/overview.html
2. Bass, J.: Scrum master activities: process tailoring in large enterprise projects. In: Proceedings of the International Conference on Global Software Engineering, pp. 6–15. IEEE, Washington, DC, USA (2014)
3. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development. http://agilemanifesto.org (2001)
4. Begel, A., Nagappan, N.: Usage and perceptions of agile software development in an industrial context: an exploratory study. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement, pp. 255–264. IEEE Computer Society, Washington, DC, USA (2007)
5. Book, D.: Improving your processes? Aim high. http://jroller.com/bokmann/entry/improving_your_processes_aim_high
6. Chow, T., Cao, D.B.: A survey study of critical success factors in agile software projects. J. Syst. Softw. **81**(6), 961–971 (2008)
7. Chrissis, M., Konrad, M., Shurm, S.: CMMI. Guidelines for Process Integration and Product Improvement, 2nd edn. Addison Wesley, Boston, MA (2007)
8. Conboy, K.: Agility from first principles: reconstructing the concept of agility in information systems development. Inf. Syst. Res. **20**(3), 329–354 (2009)
9. Datta, S.: Agility measurement index: a metric for the crossroads of software development methodologies. In: Proceedings of the Southeast Regional Conference, pp. 271–273. ACM, New York, NY, USA (2006)
10. Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N.B.: A decade of agile methodologies: towards explaining agile software development. J. Syst. Softw. **85**(6), 1213–1221 (2012)
11. Duvall, L.: Be quick, be useable, be on time: lessons in agile delivery of defense analytic tools. 21st Century Defense Initiative Policy Paper (2012)
12. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: a systematic review. Inf. Softw. Technol. **50**(9–10), 833–859 (2008)
13. Goldman, S., Nagel, R., Preiss, K., Dove, R.: Iacocca Institute: 21st Century Manufacturing Enterprise Strategy: An Industry Led View. Iacocca Institute, Bethlehem (1991)
14. ISO/TC 210: Medical device software – software lifecycle processes. International Standard IEC 62304:2006, International Standards Organization (2006)
15. Krueger, C.: Software product line reuse in practice. In: Proceddings of the IEEE Symposium on Application-Specific Systems and Software Engineering Technology, pp. 117–118. IEEE, Washington, DC, USA (2000)
16. Kurian, T.: A fuzzy based approach for estimating agility of an embedded software process. http://www.siliconindia.com/events/siliconindia_events/Global_Embedded_conf/Globa_Embedded_Conf_PPT_final_tisni.pdf (2011)
17. Lappo, P., Andrew, H.: Assessing agility. Extreme Programming and Agile Processes in Software Engineering. Lecture Notes in Computer Science, vol. 3092, pp. 331–338. Springer, Berlin (2004)
18. Murphy, B., Bird, C., Zimmermann, T., Williams, L., Nagappan, N., Begel, A.: Have agile techniques been the silver bullet for software development at Microsoft? In: Proceeding of the International Symposium on Empirical Software Engineering and Measurement, pp. 75–84. IEEE, Washington, DC, USA (2013)
19. Niazi, M., Mahmood, S., Alshayeb, M., Rehan Riaz, M., Faisal, K., Cerpa, N.: Challenges of project management in global software development: initial results. In: Proceedings of the Science and Information Conference, pp. 202–206. IEEE, Washington, DC, USA (2013)

20. Phalnikar, R., Deshpande, V., Joshi, S.: Applying agile principles for distributed software development. In: Proceedings of the International Conference on Advanced Computer Control, pp. 535–539. IEEE, Washington, DC, USA (2009)
21. Qumer, A., Henderson-Sellers, B.: An evaluation of the degree of agility in six agile methods and its applicability for method engineering. Inf. Softw. Technol. **50**(4), 280–295 (2008)
22. Qumer, A., Henderson-Sellers, B.: A framework to support the evaluation, adoption and improvement of agile methods in practice. J. Syst. Softw. **81**(11), 1899–1919 (2008)
23. Reagan, R., Rico, D.: Lean and agile acquisition and systems engineering, a paradigm whose time has come. Defense Acquisition University, Defense AT&L (2010)
24. Rottier, P., Rodrigues, V.: Agile development in a medical device company. In: Proceedings of the Agile Conference, pp. 218–223. IEEE, Washington, DC, USA (2008)
25. Séguin, N., Tremblay, G., Bagane, H.: Agile principles as software engineering principles: an analysis. Agile Processes in Software Engineering and Extreme Programming. Lecture Notes in Business Information Processing, vol. 111, pp. 1–15. Springer, Berlin (2012)
26. Seuffert, M.: Agile Karlskrona Test. http://mayberg.se/archive/Agile_Karlskrona_Test.pdf (2009)
27. Shawky, D., Ali, A.: A practical measure for the agility of software development processes. In: Proceedings of the International Conference on Computer Technology and Development, pp. 230–234. IEEE, Washington, DC, USA (2010)
28. Sheffield, J., Lemétayer, J.: Factors associated with the software development agility of successful projects. Int. J. Proj. Manag. **31**(3), 459–472 (2013)
29. Sieger, D.B., Badiru, A.B., Milatovic, M.: A metric for agility measurement in product development. IIE Trans. **32**(7), 637–645 (2000)
30. Somanath, N., Sabu, K., Krishnanakutty, K.V.: Measuring agility of organizations - a comprehensive agility measurement tool (camt). Int. J. Innov. Res. Sci. Eng. Technol. **2**(1), 666–670 (2013)
31. Turner, R.: Toward agile systems engineering processes. CROSSTALK the Journal of Defense Software Engineering, pp. 11–15 (2007)