is a lack of defining exactly *what does Agile mean* in your organization. Agile is an umbrella term since it may include more than one methodology (as described above), and more than one technique or practice (e.g., continuous integration, pair programming or test driven development). Without a clear understanding of what Agile means, each Project Team may be doing something very different even though they all call it an Agile development.

A white paper by (Stuart, 2012) referenced a 2008 survey reporting that over 71% of teams using Agile were using some variation of Scrum, about 8% were using XP, and none of the other methods accounted for more than 5% each. As a result, this discussion of Agile will focus on Agile/Scrum**.** In order for Agile/Scrum to work effectively, there are some important environmental factors that must be in place. They include:

- Self-managing teams that are mature—meaning the team must work cohesively, be accountable to commitments and able to maintain respectful and productive relationships
- A software product where your customer has agreed to an incremental delivery schedule—the key here is to determine how the software product can be broken down into small segments that can be designed, built and tested in about two weeks each
- Stakeholders and a parent organization that are committed to the Agile philosophy

4.3.2 Core Principles of the Agile/Scrum Methodology

Scrum is an Agile management *framework* for software development. The term "agile" implies flexibility as well as rapidity and the core principles exemplify those goals. In summary, they are:

- *Self-Organizing Teams*: A powerful aspect of Agile is the self-organization of its multidisciplinary teams where the team members continually interact while working on specific short-term goals. *Note:* The multidisciplinary teams are similar to the Software Integrated Product Teams (SwIPT) described in this Guidebook, but the SwIPTs are not self-organizing.
- *Self-Directed Teams*: Agile teams interact like self-correcting organisms based on a constant feedback mechanism. (Beware of an adage: If everyone is in charge, no one is in charge!)
- *Daily Stand-Up Meetings*: Short daily early morning meetings provide self-directed teams with timely feedback information the team uses to monitor and adjust their daily tasks.
- *Information Sharing*: Information is shared by *all* team members; communication is facilitated by co-location of the team members sometimes augmented with "team rooms" containing informative charts such as burn-down charts showing the work remaining.
- *Minimal Processes and Principles*: The Agile approach is to down-size processes to the bare minimum set needed to address only the current work. Processes, procedures, documentation, checklists, etc. are replaced in Agile by *guiding principles*. The Scrum Master, or the Project Leader, decides which principles to employ and how they are implemented.
- *Frequent Releases*: Most Agile approaches release software more frequently than the traditional methods. By delivering code early and often, Agile Teams get frequent checkpoints to validate their understanding of the requirements. The basic unit of development in Agile/Scrum is called a *Sprint* (an iteration of a specific duration fixed in advance for each Sprint). The Sprint duration is usually between one week and one month, with two weeks being common.

  Sprints normally start with a planning meeting where the Product Owner and the Development Team agree upon exactly what work will be accomplished during the Sprint. The Development Team decides how much work can realistically be accomplished during the Sprint. The Product Owner determines the approval and acceptance criteria. Acceptance normally requires the software comprising the Sprint to be fully integrated, fully tested, documented as required, and potentially shippable to the customer.
- *Continuous Testing*: Agile Teams test at the end of each Sprint or iteration to ensure that all of the planned features are performed. This frequent testing helps to reduce risk because more issues are identified early in the life cycle.
- *Supporting Infrastructure*: In order to produce working software, especially for large implementations, that is sufficiently stable, and at the prescribed level of quality, a *supporting infrastructure* is needed that may include:
  - Continuous builds that are validated by practices such as automated smoke tests.
  - An automated unit-level testing framework.
  - Automated system-level Regression Testing.
  - An automated software release and status process.
- *Customer Collaboration*: Communication is a hallmark of the Agile/Scrum methodology, and face-to-face communication maximizes its effectiveness. Co-location of the customer, or the customer's representative, is necessary in order to interface with the Agile Team daily. However, as the complexity and size of the system grows, it can be very difficult to find a single person who is a good communicator, available full-time and understands the entire system.

4.3.3 Is Agile a Silver Bullet?

Lessons Learned. I believe there are never two sides to every story—there are always *three* sides! You can call the three sides *the positive*, *the negative* and *the neutral* or call them *the optimistic*, *the pessimistic* and *the realistic*. Whatever you call them, this Guidebook takes the third side: *neutral and realistic* and that is how Agile is discussed.

There are people who believe that the Agile methodology is the *only* way to go. Treatment of the Agile methodology in this section is neutral and realistic because both the pros and the cons are discussed, and other options are described, so it is left up to the Project Manager to decide what is the best fit for his or her project. Hopefully, as the Software Project Manager you will have the opportunity to decide what methodology is best for your project rather than being forced to adopt a mandatory Agile methodology by your home organization regardless of its fit.

Most Agile methodologies are *team-level* workflow approaches. These approaches can be highly effective at a local small team level, but they do not easily scale up to geographically dispersed large systems, and they fail to address the entire system architecture, system requirements, system planning needs, and effective management of large systems. Successful implementations show that very large geographically dispersed software system developments require extensive planning, vigorous control, formalized communication abilities, considerable (applicable) documentation and comprehensive integrated testing procedures; all of that is essentially inconsistent with the Agile core principles.

Since there are no silver bullets, I do not believe that Agile is the greatest thing since sliced bread—but many advocates believe that to be the case. Regardless, if applied properly in the right environment, even on large systems, Agile can be an effective methodology (as described in Subsection 4.3.5).

If a self-directed team means no one is in charge, it can have inherent risks. Large complex software-intensive systems are typically one-of-a-kind, first time, expensive and risky developments; you need to embark on such a journey with as much *clarity and direction* as you can get. There may be a software development standard process produced by your organization that can be followed, but someone must decide if the standard process can be followed for your project, if it needs to be tailored, or if you need to develop a hybrid process for your project. Also, self-directed teams often cannot make the quick decisions needed during development because of differences of opinion. It may be possible for the process to be managed by a self-directed team, but a large project must be managed by someone who has clear *responsibility and authority* to make those decisions: the Software Project Manager.

4.3.4 Agile Operational Considerations

Working Software. The goal of every software project, regardless of size, is to produce *working software*, but what does that mean? Some strong proponents of the Agile methodology believe that *delivery of working software code is the only goal*. Even if the software does not interact with hardware, that opinion is not valid because you must successfully *test and verify* that the software is responsive to your customer's requirements, it must include the documented information needed to run and maintain the system, it may involve user training, and it usually involves much more. If you are developing a complex *mega system*, the goals are much more than just "code that works."

Definition of Done. Defining "done" is important for every software development, but for Agile developments it is of paramount importance to the team. The definition of done is specific to each project. For Agile developments the fundamental definition of "done" at a minimum should include:

- The code for each Sprint or story is complete, and it compiles without error
- Required tests have been written, executed and passed
- Functionality is fully compliant with the user story and test results have been reviewed and approved by the customer
- Applicable documentation was produced, reviewed, approved and checked in

Control of Agile Projects. In Chapter 2, Figure 2.2 displayed the amount of project control that is needed compared to the size of the project. That figure clearly shows that large and complex projects require more management control than small, simple projects. Figure 4.5 is a modification of that figure with the additional mapping of the principal area of the Agile focus versus the focus of this Guidebook. Agile is best suited for and most effective on small systems. The recommended processes and procedures in this Guidebook

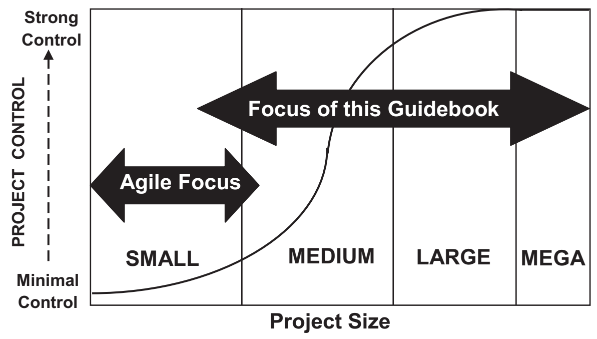

Figure 4.5 Project control mapped to project size and focus areas.

are best suited for and highly effective on medium through large and mega systems.

It is almost certain you have never seen a diagram of an Agile group with a Software Project Manager box at the top. The very nature of the Agile/Scrum methodology, and its fundamental building blocks, seem to be the antithesis of the systematic structure and defined processes needed to successfully produce, control and deploy large complex software-intensive systems as described in this Guidebook. Furthermore, if Agile is misapplied, and force-fitted into situations that it is not suited for, it would be a good example of trying to put square pegs in round holes.

4.3.5 Application of Agile to Medium and Large Systems

Despite the Agile focus issue discussed in Subsection 4.3.4, Agile is important because, if properly applied in the correct environment, it can be and has been an effective approach. When properly applied, Agile can be used in a large project by integrating the development activities of multiple Scrum Teams. There have been many attempts to scale up the Agile philosophy to large projects with varying levels of success. Some of the Agile scaling technique in current use include:

- *Scrum of Scrums (SoS)*: The Scrum of Scrums technique is the oldest and likely the most commonly used. It is used for integrating the work of multiple Scrum Teams (normally 5–9 members each) working on the same project. The SoS approach is described below when used in combination with a traditional approach on large systems.
- *Large-Scale Scrum (LeSS)*: An expanded version of a single Scrum Team focused on producing the entire product with one Product Owner and a single Product Backlog.
- *Scaled Agile Framework (SAFe)*: A knowledge-based technique used to scale lean-Agile developments in large software enterprises.
- *Disciplined Agile Delivery (DAD)*: The DAD approach is to build and test the most risky or difficult parts first to confirm feasibility of meeting the project goals.
- Nexus Scrum of Scrums (Nexus SoS): Best used for 3–9 Scrum Development Teams with a common Product Backlog.

Since Agile is ideal for small applications, it is reasonable to apply the Agile methodology to *small pieces of a large system*. As displayed in Figure 4.6, Agile can be performed in *combination* with one of the non-Agile approaches described in this chapter. Figure 4.6 also identifies the equivalent Agile terminology. This combination of both methodologies will work if you can identify Software Units (SUs) that are small enough and consistent with the Agile criteria. Then the Agile developed SUs can be combined to produce higher-level software components (also called higher-level SUs). From that point on, one of the traditional methods takes over as the full system is integrated and incrementally built up.

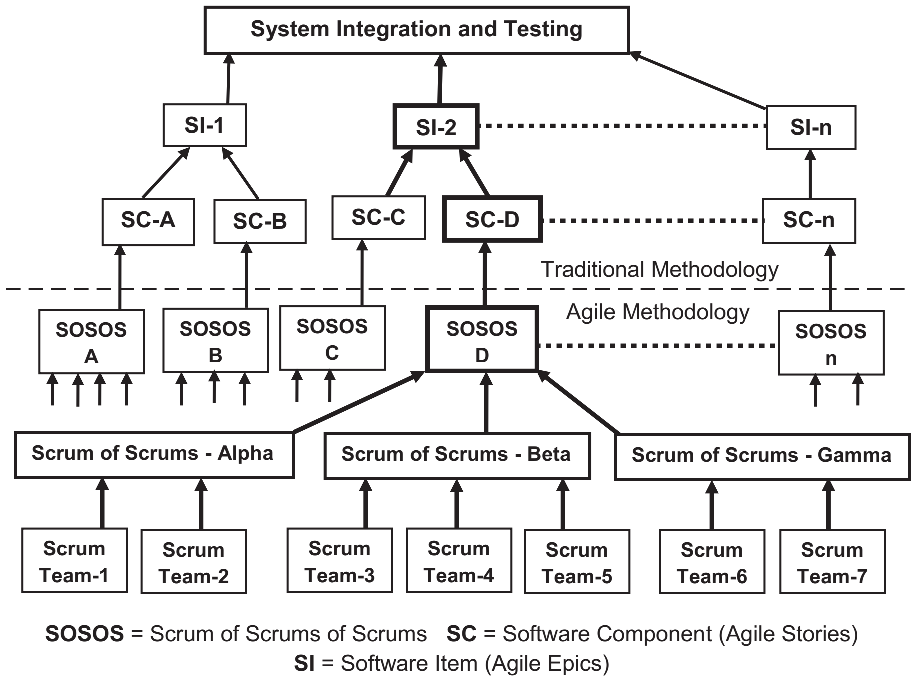

Figure 4.6 Combination of Agile and traditional methodologies in large systems.

When using the Agile methodology combined with a traditional approach, the work products produced must be reviewed and approved, usually by the Chief Software Engineer (CSwE), to make certain the developed products are compliant with system requirements and to confirm that a *self-directed Agile Team* did not decide to go in another direction.

Attempting to apply a *fully Agile approach* to the entirety of a very large software-intensive system is extremely difficult if not impossible. Successful developments of large complex systems require processes that are generally inconsistent with the core principles of the Agile methodology. If you plan to follow some type of a scaled Agile framework, consider hiring an Agile consultant who has experience with Agile scaling. You can then decide what scaling option is best for your project or, even better, customize an option and then have a proactive plan to improve it incrementally during the developmental process.

4.3.6 Agile/Scrum Roles

The principal roles performed may be called by different names depending on the specific Agile methodology being used but the basic concepts are similar. There is no formal role for a Software Project Manager or Chief Software Architect as these roles are incompatible with the Agile concept. *If* a Project Manager is involved with an Agile project, they are typically focused only on resolving issues that block progress originating outside the team's boundaries (i.e., hands off the team!).

Scrum Master. In the Agile parlance the *Scrum Master*, also called the *Team Leader* or *Coach*, *essentially* performs some functions similar to the Software Project Manager including acting as the interface between the Agile Team and the outside world. The Scrum Master is the *keeper of the Agile process*, responsible for making the process run smoothly, removing obstacles that impact productivity, facilitating the critical meetings, understanding Scrum well enough to train and mentor the other roles, and educating and assisting other stakeholders. The Scrum Master maintains a constant awareness of the status of the project but does *not manage the team*, and does *not assign tasks* to team members since task assignments are the internal responsibility of the team. An important function of the Scrum Master is to prevent new requirement changes from being introduced during a Sprint. This allows the team to focus, without distraction, on performing their tasks and benefit from the resulting stabilizing environment.

The Scrum Team. The Agile/Scrum Project Team is a *self-organizing cross-functional group* of people who perform the hands-on work of developing and testing the product. Since the team is responsible for producing the product, they also have the authority to make decisions about how to perform the work. The team is therefore *self- directing*: Team members decide how to break work into tasks and how to allocate tasks to individuals for each Sprint. The team size is normally kept in the range of 5–9 people. A *Scrum Team* refers to the team plus the Scrum Master and Product Owner.

Product Owner. The Product Owner is the keeper of the customer requirements and the *single point of contact regarding all requirements* and their planned order of implementation. The Product Owner is the interface between the organization, the customer, their product related needs, and the team. He or she buffers the team from feature and bug-fix requests that come from multiple sources. Each project has only one Product Owner. In other Agile frameworks (such as Extreme Programming) the Product Owner is usually the customer or the customer's representative. Communication is a core responsibility of the Product Owner who also serves as the team's representative to the overall stakeholder community.

When several teams work on one product, they should generally use a single Product Owner (who has the authority to make business decisions) and a single Product Backlog with customer-centric requirements. Each Scrum Team should strive to become a *feature team*, able to build a complete slice of a deliverable product.

The Customer. Your project's customer, or the customer's representative, plays a substantial role in an Agile project. During the Agile planning process, the customer is the leader as they specify what they want and the order they want to receive it. The ideal Agile concept has the customer, or a proxy, *co-located* with the Agile Team in order to maximize the benefits of daily interaction during the sprints. When co-location is not practical or possible, planned daily telephone or video conferences can be held. The modern office can be viewed as a "virtual office" since team members, and the customer can communicate all day by teleconferencing with VoIP headsets.

For larger software-intensive systems, the customer typically prepares detailed specifications such as the *Request for Proposal* (RFP), *Statement of Work* (SOW), a *Technical Requirements Document* (TRD) and sometimes the *Work Breakdown Structure* (WBS). All the specifications are expected to be updated as specific requirements change during the development process. In many (or most) Agile developments the customer is not involved with (or cares) *how* the software is built; the customer is deeply concerned with *what* the software produces. Consequently, it would appear that involving the customer in meetings about the detailed structure of the software may be a waste of their time and yours and may slow you down.

4.3.7 Agile/Scrum Terminology

Many new terms have been "invented" for the Agile methodology, and many of them are metaphors to help the team develop a *vision* of what they are trying to accomplish. It does appear that some Scrum names are *new names for existing terms*. My attempt to define "essential equivalents" of the Agile terminology is shown in the example Rosetta Stone in Table 4.3 listed alphabetically.

4.3.8 Adopting the Agile Philosophy

The Agile approach could be considered more of a modern *philosophy* than a development methodology. This philosophy is focused on value to the customer, effective collaboration, rapid responsiveness to changes, and efficiency in the approach to product delivery. As described by (Kennaley, 2010), "The goal is not to just "do agile" but to "be agile." Simply utilizing an agile process will yield some benefit, however, if being agile is the goal a *culture of agility* needs to be created."

Some companies who decide to introduce Agile practices do so without thoroughly thinking through *why* they are doing it and without confidence that switching to Agile will bring long-term benefits. Often, executive management decides to make the switch because their peers are going the Agile route. For such companies, Agile adoption is like a fashion statement. Such companies care more about keeping up with the mainstream than making a well thought-through and carefully planned step forward.

Table 4.3 Agile/Scrum Nomenclature Rosetta Stone

Scrum Name	Essential Equivalents(*)
Backlog or Product Backlog Item	Requirements not yet implemented
Cadence	Time between sprints or releases
Feature Team	Systems Engineering Integration and Test (SEIT) Team
Information Radiator	Status Chart
Poker Planning	Estimating
Product Level Testing	System Testing or Release Testing
Product Owner or Chief Product Owner	Responsible Product Engineer or Chief Software Engineer or Software Project Manager
Product Owner Team (or Council)	Integrated Product Team
Product Roadmap	Software Development Plan
Product Vision	Concept of Operations(**)
Release	Release or Version
Retrospectives	Project Review (Lessons Learned) Meetings
Scrum Master (or Coach)	Software Project Manager/Team Leader
Scrum of Scrums	Meeting of two or more Project Teams
Spikes	Milestones
Sprint	Build (or Software Unit) Iteration
Sprint Backlog	SU Requirements Prioritized
Story Points	Requirements Estimation
Team-Level Testing	SI Qualification Testing (SIQT)
Time Boxes	Time Durations/Sprint Iterations
User Stories	Requirements or features needed
Velocity	Throughput

(*) There are subtle differences for some of these equivalents to the Scrum name.
(**) Could also be the Capabilities Development Document (CDD), the Technical Requirement Document, or the Project Charter.