(FR)AGILE?
RISK MANAGEMENT AND AGILE SOFTWARE DEVELOPMENT

John P. Beardwood and Michael Shour[1]

TABLE OF CONTENTS

---

[1] John P. Beardwood and Michael Shour are of the law firm Fasken Martineau DuMoulin LLP, Toronto. John is a partner, Co-Chair of the firm's National Technology and Intellectual Property Practice Group, and Co-Chair of the Outsourcing Practice Group. Michael is an associate in the Business Law Group. This is a general overview of the subject matter and should not be relied on as legal advice. For specific legal advice on the topic and related matters, please contact the authors.

1. INTRODUCTION

A recent report from Forester Research, in a Q3 2009 survey, asked a cross-selection of 1298 IT professionals to identify which methodology most closely reflected the development process that such professional was currently using. At 35%, the most popular response was Agile, in comparison to a mere 13% who responded that Waterfall was their current methodology of choice.[2] Put another way, three times as many IT professionals used Agile methodologies as did those who used Waterfall.

Over the last ten years, Agile software development methodologies—those which take an iterative and incremental approach, which aim to reduce unnecessary documentation and formality, and which seek to promote teamwork and experimentation—have increasingly been adopted by the software development community. Agile's advocates argue that by liberating programmers from the shackles of traditional rigid, formalized development methodologies, Agile has no equal for speed of development, for efficiency, and for fostering creativity. Agile's critics counter that Agile is another name for "cowboy coding", or undisciplined "hacking", which produces less robust and buggier software. From a legal perspective, however, the question is not whether Agile is a superior development methodology. Rather, the question is whether a contract for software development using Agile needs to be differently structured, and include different content, than where the development is using Waterfall. In this paper, we argue that the answer is an emphatic yes.

We begin by examining the differences between, and the advantages and disadvantages of each of, the Waterfall and Agile development methodologies. We examine how Agile is not single, monolithic methodology, and that it in fact suffers from some definitional problems (e.g. when does Iterative programming transition into Agile?), by briefly reviewing some of the major Agile methodologies. We then summarize the developing consensus as to when Agile will, or will not, be an appropriate methodology for a developmet project.

Having provided a briefing on Agile as a methodology, we then highlight the risks of Agile, and potential contractual strategies for responding to and mitigating such risks.

2. UNDERSTANDING (FR)AGILE SOFTWARE

(a) Traditional Software Development

Agile is perhaps best understood by contrasting it to Waterfall methods of development. The traditional plan-driven software development model of the Waterfall process was initially conceived to manage the development of large software projects in the 1960s, such as IBM's System/360 operating system. Waterfall arranges software development into a sequential series of compartmentalized phases, each with its own defined deliverables. A typical Waterfall

[2] Forester Research, Dr. Dobb's Global Developer Technographics Survey, Q3 2009. The other result of note was that the second most popular response, at 30.6%, was "Do not use a formal process methodology". As a result, the second most popular response which *actually identified a formal methodology* was Iterative Development, at 21%. Given that the iterative process is a key element of Agile, one might argue that this result is also reflective of a trend towards adopting Agile-styles of programming.

process will be comprised of the following phases: requirements analysis; project planning and system design; coding; and integration and testing.[3]

The Waterfall process is characterized by planning, organization, and documentation. At the outset of a Waterfall process, there will be a great deal of emphasis on gathering and defining the business *requirements* – that is, the "raw data" as to the what the customer wants. Through formal meetings and analysis, a highly detailed requirements document is developed. Once the requirements are set out in the requirements document, a system design document is developed, with a set of *specifications*[4] - or design "roadmap" - as to how those requirements will be achieved through software design. From an operational point of view, the process of developing the specifications involves the customer, and thus can itself assist in clarifying the requirements of the customer. From a contractual point of view, however, if there are ever disputes, or any ambiguity, as to what was promised to the customer, this also can be clarified through an examination of these documents.

Once the design specifications have been developed, the project is divided into distinct modules with detailed deliverables. Each module can be assigned to a different programming team, with each team being responsible for programming, testing, and debugging their respective module. Since the deliverables for each module are clearly set out in the planning documents, the programmers need not be well-rounded, highly-skilled programmers, but merely need to be sufficiently skilled to understand the requirements and carry out their respective tasks. Test plans and reports are produced to ensure that each module meets the specifications. If any changes are requested or required, a formalized and well-documented change management process is followed and change management forms must be completed. After the coding stage, the modules are assembled, tested, and debugged, often with significant time spent rewriting code to ensure the modules integrate properly.[5]

The planning and documentation associated with the Waterfall process yields some obvious benefits:

- *Limited, targeted customer involvement:* The business people who are consulted in developing the specifications are required to invest time at the early stages. Once the specifications have been developed, however, their involvement is generally limited to the final integration and testing phase, allowing them to carry on with their primary business functions.

- *Clear budgeting*: Clear specifications and timelines allow developers to make accurate bids for the work, while allowing the procurer to effectively budget for the project.

---

[3] Michael A. Cusumano and Stanley Smith, *Beyond the Waterfall: Software Development at Microsoft* (Working Paper #3844-BPS-1995, MIT Sloan School of Management and International Business Machines). The Waterfall model was first described by Winston W. Royce, "Managing the Development of Large Software Systems," Proceedings of IEEE, August, 1970.

[4] The terms are not always utilized so clearly, in that a requirements document is sometimes called a "requirements specification" i.e. simply the requirements written down on paper.

[5] See generally, Dan Marks, "Development Methodologies Compares Why different projects require different development methodologies" N-Cycles Software Solutions, December, 2002 (online: http://www.ncycles.com/e_whi_methodologies.htm).

- *Single development roadmap:* For contracting purposes, the detailed specifications lead to a requirements document that is useful from the perspective of both the developer and the customer; once work begins, large teams—even those spread out over different geographic locations—can follow the detailed requirements over longer time horizons.

- *Divisible modules:* Once the project is divided into specific modules with clear deliverables, outsourcing or subcontracting one section of the project becomes possible.

- *Benchmark for change management:* If any changes are requested or required, an examination of the requirements and design documents allow the project manager to quickly determine the impact of the change on the timeline, cost, and operation of the system. Such changes are well documented pursuant to the change management process in the development contract to ensure that there are no project-paralyzing disputes down the road.

On the other hand, the very structural elements (i.e. planning, organization, and documentation) that produce the above benefits, are the same elements which can disadvantage the Waterfall method:[6]

- *Less flexible/responsive to change:* While knowing exactly what each stage of the project will produce at the beginning of a development project has obvious benefits, it is not always realistic to assume that the customer will be able to clearly articulate its requirements at the outset. Often, the need for certain different or additional deliverables only becomes apparent once the development project is underway. The rigid planning and detailed documentation of the Waterfall method can become particularly burdensome when developing software for a rapidly changing market.

- *More isolated from collegial/customer feedback*: Segmenting the project into modules, to be completed by different teams, discourages cohesion and creativity amongst the development team, reduces valuable customer/user input, and often results in design flaws not being discovered until well into the testing phases. As a consequence, integration and acceptance both carry a significant portion of the operational and legal risk involved in a Waterfall development project.

(b) **What is Agile Software?**

Although the traditional Waterfall method has worked well for many large projects, some argue that different programming methods are suitable for different projects, and that not all projects fit well into the compartmentalized and formalized traditional Waterfall process.[7] While there are a number of software development methodologies that are viewed as Agile, what those

---

[6] David Parnas and Pail Clements, "A Rational Design Process: How and Why to Fake it." *IEEE Trans. Software Eng.*, February 1986, pp. 251-257.

[7] See for example, R. McCauley, "Agile Development Methods Poised to Upset Status Quo" *SIGCSE Bulletin 33(4), 2001*; R.L. Glass, "Agile Versus Traditional: Make Love, Not War! *Cutter IT Journal* 13(12), 2001 as cited in Pekka Abrahamson, Outi Salo, Jussi Ronkainen, & Juhani Warsta, "Agile Software Development Methods: Review and Analysis" (University of Oulu, Finland: VTT Technical Research Centre of Finland, 2002), online: http://Agile.vtt.fi [Abrahamson et al., "Agile Software Development"]

methodologies have in common is that all generally seek to move away from the traditional approach—where there is a great deal of up-front planning, a master document, and one or few iterations—towards an iterative and incremental approach, with less documentation and formality, and more teamwork and experimentation.[8]

In 2001, a group of software developers and consultants, all champions of different Agile methods, came together to publish the Agile Software Development Manifesto.[9] While the principles of the Agile Manifesto are attached here as Appendix A, in summary the central values of all Agile process are: individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; and responding to change over following a plan.[10] As some of these central values, such as the deliberate lack of comprehensive documentation and contract negotiation, are antithetical to the precepts of commercial contracting, each counsel to a customer involved in an Agile development project must strike an effective balance between protecting their customer and ensuring that their customer does not lose the benefits associated with Agile.

In every aspect of development, Agile methods differ greatly from the traditional Waterfall process:

- *Ongoing and evolving specifications*: Rather than developing a detailed set of specifications and documentation at the outset of the project, the requirements in an Agile project are largely emergent and will change throughout the life of project as a result of ongoing discussions between the development team and the customer.

- *More extensive and deeper customer involvement:* While the traditional Waterfall method requires initial and targeted involvement from the customer's subject matter experts mainly during the requirements phase, in an Agile process the customer's personnel will be on site regularly, collaborating with the development team and modifying requirements throughout the life of the project. Although this allows for a greater degree of control, it also entails a greater investment of time on the part of the customer. The customers must fully understand the involved nature of the Agile process at the outset or the collaborative efforts will fail.

- *Smaller workteams:* Whereas traditional software development teams may involve hundreds of programmers working in various locations on disparate modules, an Agile team will usually consist of two to eight people working together in a highly collaborative, self-organizing manner, usually within the confines of one "war room."

- *Well rounded, experienced programmers able to multi-task*: The team members will all be highly skilled experienced programmers, able to take on multiple areas of the project

[8] See generally, Abrahamson et al., "Agile Software Development."

[9] www.Agilemanifesto.org and www.Agilealliance.org. See Martin Fowler, "The New Methodology" MartinFowler.com, December, 2005 for a history behind the development of the Agile manifesto, pages 13-14 [Martin Fowler, "The New Methodology"]. An abridged version entitled "Put Your Process on a Diet" was originally published in *Software Development*, December, 2000.

[10] From www.Agilemanifesto.org (accessed September 21, 2009).

at once so that no individual acts as a bottleneck. The drawback of this, however, is that each programmer is much less replaceable – effect, there are more eggs in less baskets.

- *Ongoing, periodic testing/quality control:* Instead of testing and integrating various modules after coding is complete, Agile methods involve short increments of one to three months to allow for quick testing and repairing throughout the project. Specific tools and techniques such as continuous integration[11] are often used to improve quality and enhance project agility. This also reduces integration risk.

- *Success measurement.* Finally, in a plan-driven traditional software development project, a successful project is one that has met the deliverables set out in the plan in a timely manner. A successful Agile project, however, "will build something different and better than the original plan foresaw."[12]

(c) Different Agile Software Development Methodologies[13]

There are numerous Agile methodologies, each with their proponents and detractors, and each with their own eccentricities. While all share the common characteristics discussed above, each arrives at the ultimate goal of attaining development agility in different ways. For customers considering embarking on an Agile development project, each must conduct careful due diligence into the development methodologies that might be employed in carrying out their software development project to determine (I) if Agile is suitable at all given the project in question, the culture of their organization, and the resources their organization has available, and (II) if it is suitable, which particular Agile methodology would be most effective. Below, some of the major Agile software development methodologies are briefly described:

(i) Extreme Programming (XP)

Probably the most well-known Agile development method is Extreme Programming. Kent Beck, the father of Extreme Programming and one of the founders of the Agile Alliance, explains that the term was intended to convey the intensity with which programming can be done: "Extreme Programming is an aware and focused activity—all dials turned to 10—attending to everything you need to attend to and wasting no energy on things that don't matter."[14] Extreme Programming was first conceived by Beck in his role as project leader in the development of Chrysler's Comprehensive Compensation (C3) System payroll project around 1996. Beck published the first edition of his book, *Extreme Programming*, soon afterwards in 1999.[15] Extreme Programming quickly gained popularity in the late '90s and early 2000s and is

---

[11] "Continuous integration" or "CI" implements continuous processes of applying quality control - small pieces of effort, applied frequently. Continuous integration aims to improve the quality of software, and to reduce the time taken to deliver it, by replacing the traditional practice of applying quality control *after* completing all development. See http://en.wikipedia.org/wiki/Continuous_integration.

[12] See Martin Fowler, "The New Methodology".

[13] See Abrahamson et al., "Agile Software Development", *supra* note 7, for an extensive overview of Agile software development methodologies.

[14] Steve Hayes, "Battling Extreme Programming's Misconceptions" (08 January 2004) Builder.com.

[15] The book is now in its second edition: K. Beck & C. Andres, *Extreme Programming Explained*, 2nd Edition (Upper Saddle River, NJ: Addison-Wesley, 2005).

one of the most widely used Agile methodologies. Like most Agile methodologies, the Extreme Programming process is characterized by short development cycles, incremental planning, continuous feedback, and reliance on communication, and iterative design.

Many of the concepts and practices associated with Extreme Programming are not particularly unique or innovative, but rather it is the combination and implementation of these practises ("all dials turned to 10") that makes this method effective. In Extreme Programming, there is a significant focus on communication between stakeholders and on face-to-face feedback. One such communication-oriented activity is the "Planning Game" at the outset of the project, where user stories are written on cards, estimated and prioritized. For each release, another Planning Game is performed. Another communication-oriented principle is the "Onsite Customer". Throughout the development process, it is standard practice to have the customer onsite, or at least readily available, to answer questions, set priorities and determine requirements of the project. Finally, another key feature of Extreme Programming, one which distinguishes it from other Agile methodologies, is its focus on pairs of programmers programming together; one programmer "drives" while the other "codes." Roles are frequently switched to keep the programmers fresh and teams generally do not exceed a forty-hour work week to ensure productivity can be sustained.[16] Pairs are not only involved in coding, but also in testing and integration, which immediately follows any coding.[17]

(ii) Scrum

The term "scrum," taken from rugby terminology,[18] was first used in a Harvard Business Review article discussing new commercial product development, not software.[19] In the early 1990s, both Ken Schwaber and Jeff Sutherland independently developed and employed the similar Scrum-style development practices at their respective companies. They jointly began presenting papers and writing on the subject.[20]

Unlike some of the other methodologies, Scrum does not prescribe detailed engineering practices, but rather focuses on the management aspects of software development. A product backlog is developed and it defines everything that is needed in the final product based on the current understanding. It is continuously updated through consultations between the customer and members of the development team. Through a collaborative dialogue, the customer and the development team jointly estimate the effort required to carry out various tasks, and development is divided into thirty day iterations, called "Sprints." A Sprint planning meeting is organized by

---

[16] Juha Koskela, Mauri Myllyaho, Jukka Kääriäinen, Dan Bendas, Jarkko Hyysalo, & Anne Virta, "Experiences of Using Extreme Programming to Support a Legacy Information System Migration Project" (University of Oulu: VTT Technical Research Centre of Finland, 2004).

[17] See generally www.extremeprogramming.org and www.xprogramming.com.

[18] In rugby, a scrum or scrummage is the method of beginning play in which the forwards of each team crouch side by side with locked arms. Play beings when the ball is thrown in and the two sides compete for possession.

[19] H. Takeuchi and I. Nonaka, "The New Product Development Game" (1986), Jan./Feb., *Harvard Business Review*, at 137-146. The same authors have also, more recently, released a book covering similar ground: H. Takeuchi and I. Nonaka, *Hitotsubashi on Knowledge Management* (Singapore: John Wiley & Sons (Asia), 2004).

[20] The first of which was "Scrum Development Process" OOPSLA'95 Workshop on Business Object Design and Implementation (Springer-Verlag, 1995). See generally, http://www.controlchaos.com/, Shwaber's website; Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum*, (Prentice Hall, 2001).

the Scrum Master at the outset of each Sprint, in which the development team develops the objectives along with the customer. Close monitoring and control is attained through fifteen-minute daily scrum meetings, in which progress is discussed and any problems or other variables are addressed.

(iii) Crystal

Alistair Cockburn, a well known proponent of Agile software, formulated the Crystal development methodologies.[21] Unlike Extreme Programming, Crystal does not advocate specific processes, but recommends different methodologies for different size teams and different types of projects. Crystal can be understood as a people-focussed, rather than process-focussed, development tool; teams are encouraged to choose aspects from various methodologies, including Extreme Programming, that best suit the task at hand. Cockburn does suggest, however, the following aspects be present in an Agile development process: regular incremental delivery; progress tracking by software delivery rather than written documentation; direct customer involvement, including two user viewings per iteration; interviews and team workshops to determine the appropriate methodology for each individual project; and pre- and post-increment reflection for the project members.

(iv) Dynamic Systems Development Method (DSDM)

DSDM, based on rapid application development methodology, is a framework tool developed in the United Kingdom in the early '90s by a non-profit group of software engineering experts.[22] Unlike other Agile methodologies, which adjust time and resources dedicated to the development project in order to meet certain functionality requirements, DSDM fixes the level of functionality based on the time and resources available. This focus on the availability of resources means DSDM is particularly effective for projects with aggressive deadlines and tight budgets. The project is split into iterations called "timeboxes," and each timebox has its own budget and delivery date. Requirements are prioritized on a must-, should-, could-, or would-have basis to ensure that resources are dedicated to items of the highest priority first. Early prototyping and user testing is prescribed by the DSDM methodology to ensure the project stays on track and to maintain a high level of customer involvement.

3. CHOOSING THE RIGHT METHODOLOGY

As Agile has become more prevalent in the development community, the analytical focus has shifted from assessing whether Agile is "good" or "bad", to determining for *what types of projects* Agile is good or bad – or more accurately, for what types of project/contexts Agile is the most or least optimal methodology[23]. While Agile methods are still relatively new[24], there

---

[21] See Alistair Cockburn's website at http://alistair.cockburn.us/; See Alistair Cockburn, *Crystal Clear: A Human-As an extension of rapid application development* (Addison-Wesley Professional, 2004).

[22] See Abrahamson et al., "Agile Software Development", *supra* note 7 at 61-68; see also www.dsdm.org for the DSDM Consortium's website.

[23] Barry Boehm and Richard Turner have characterized each of the optimal contexts for each of Waterfall or "planned" methodologies, and Agile methodologies, as being such methods "homeground". See (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley. pp. 55–57.

appears to be a developing consensus that Agile is *less appropriate* (and thus "planned" methodologies such as Waterfall are more appropriate) in development contexts which exhibit one or more of the following characteristics:

- Large-scale development efforts.
- Distributed development efforts (non-co-located teams).
- Forced adoption of Agile by the development team.
- Junior developers.
- High criticality projects - i.e. mission-critical systems where failure is not an option at any cost (software for surgical procedures).
- Relatively fixed requirements –i.e. the requirements do not change often.
- Order-based development culture.
- Only limited degree of customer participation available.

Thus Agile is *more appropriate* in the following contexts:

- Smaller-scale development efforts.
- Co-located development efforts.
- Adoption by an Agile-experienced development team.
- Senior developers.
- Low criticality projects - i.e. systems with a higher tolerance for failure..
- Frequently changing requirements.
- "Chaos-based" development culture.
- High degree of customer participation available.

4. AGILE AND RISK MITIGATION: BABIES AND BATHWATER

(a) Contract required?

On reviewing such minimal literature as exists which addresses the issue of Agile development contracts, it becomes evident that there is a strong theme of disdain for the contractual process and the role it might play in a Agile development project (recall: "Customer collaboration over contract negotiation")[25]. In some ways, it is reminiscent of the some of the original hype regarding open source, where for a period developers seemed to be wilfully blind to the fact that the "L" in GPL signified "Licence" – i.e. a right to use that was governed by particular terms and conditions.

However, while there are obvious benefits to the collaboration and teamwork between the customer and the development team which Agile fosters, just as with Waterfall projects the contract between a customer and a developer must set out the roles of each of the different parties, and must address circumstances where the parties dispute one or more issues. In other words, notwithstanding that one unifying theme between the different Agile schools is the

[24] See Answering the "Where is the Proof That Agile Methods Work" Question". Agilemodeling.com. 2007-01-19. http://www.agilemodeling.com/essays/proof.htm.

[25] *Supra* note 11.

premium each places on (i) trust and understanding between the parties, and (ii) an minimum amount of documentation and formality, at the most basic level the customer still has a budget, and still requires results.

Further, whether or not the Agile collaborative approach reduces the likelihood of such disputes occurring, when such a dispute does occur, Agile's characteristics can make the resolution of that dispute more difficult. For example, in an Agile project where little or no documentation has been produced, to what documents can the parties point as evidence of their respective expectations? What if certain key project staff for one or both parties leave their respective employers –is there sufficient documentary evidence available to assist their successors? For this non-module based methodology, how are payment milestones addressed? In addition, given that there are different forms of Agile, are there specific additional risks which may be posed by the specific form of Agile in question?

(b) A Process-Oriented Contractual Framework: Governance, Governance, Governance

Once the customer has determined (i) that Agile is the right methodology, and (ii) the optimal form of Agile, in each case for their particular project/development context, the customer will needs to confirm that the development contract has all of the elements required in order to effectively manage the risk of the Agile project.

"Customer collaboration over contract negotiation"

- Key principle from the Manifesto for Agile Software Development[26]

An Agile development contract requires a greater emphasis on process than does a Waterfall contract; for example, more focus on governance and reporting. Again, where traditional "plan" software development methodologies and their respective contracts identify the achievement of very specific requirements as the end goal, Agile or process-oriented methodologies and their contracts anticipate an evolving development process. The customer will come to the table with initial business requirements that it wishes to achieve through the software, but, rather than agreeing on what the software will look like upon completing the development process, the parties will create a contractual framework for a process that will allow them to both respond to changing requirements, and to reach the end result desired, without actually specifying exactly what that result will be beyond the basic functional requirements. As a result, and as outlined in more detail below, an Agile development contract will necessarily require a greater emphasis on governance.

(c) Is Agile right for you? Buying into the Agile Process

Before initiating an Agile software development project, it is of paramount importance that all parties agree that Agile best suits the particular project. It is in the interest of both the developer and the prospective customer to ensure that they are well-educated regarding Agile philosophy, the terminology, and each step of the Agile process. This is a critical first step in avoiding disputes at a later date.

[26] http://www.Agilemanifesto.org/

We have outlined some of the factors which a customer should consider in ascertaining whether Agile is truly the right approach for their project and their organization. In developing its goals, priorities, and budget for the software project, the customer will need to consider whether Agile is the best method, in the specific context, to translate the deliverables from the project into business value.

For example, just to focus on one element, the customer needs to prepared for a more intense level of participation than in Waterfall projects - as a result, one or several of its people will have to be dedicated to the project on an ongoing basis. Further, the individual(s) that the customer dedicates to the Agile project must be knowledgeable about all aspects of the business that will potentially be impacted by the development project, in order to ensure that they are able to effectively articulate user requirements as they emerge through the development cycle.

(d) Doing the due diligence: The First Date

"Working software over comprehensive documentation"

- Key principle from the Manifesto for Agile Software Development[27]

Due to the high degree of uncertainty as to whether or not Agile may be the appropriate method for any given development project, the first stage of the Agile development process will require that the parties conduct some due diligence in order to confirm if Agile is so appropriate in the context. From a contractual point of view, that means that the first step is the development and execution of an initial non-disclosure and/or consulting agreement. This will both allow the parties to test the waters and opt out of the process after the initial phase if they so desire, and avoid expending money and time on drafting and negotiating a specialized, process-oriented contract before the developer and customer are sure they would like to engage in a development project based on Agile. Only once the parties have concluded this initial consultative phase by affirming that Agile will work in the context, should the parties then begin developing the process-oriented contract.

During this first due diligence stage of the development project, the customer and developer should work together in order to, for example:

- understand the customer's business objectives and gain an appreciation for the scope of the project;
- specify the human and technical resources which both parties will be required to dedicate, which both will provide a degree of confidence to the customer and will assure the developer that the customer is committed to dedicating the personnel and the resources required to successfully complete an Agile project; and
- agree on functional and non functional requirements and prioritize these requirements as they see fit.

[27] http://www.Agilemanifesto.org/

This is not to say that these stages do not take place and are otherwise required for Waterfall projects. Rather, the difference is in emphasis.

It also would be prudent to allow for a non-liability opportunity to opt out from the Agile process after this first stage of the contract, in order to, for example, adopt a Waterfall method of development. Since many Agile development shops only provide Agile services, a new developer may be required if the customer decides to pursue a more traditional approach.

Finally, if the customer does agree to use the Agile development methodology, the developer and the customer, having devoted a good deal of time to discussing the various details and plans during this first date stage, will have an easier time negotiating the terms of the process oriented contract.

(e) The Relationship: Iterations

> *"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale"*
>
> - Key principle from the Manifesto for Agile Software Development[28]

Once the first date stage is over and the parties have agreed to a longer-term development relationship pursuant to the terms of a process-oriented contract, the Agile development project will begin in earnest by entering into the iteration phase. Iterations are a key —if not *the* key— element of any Agile process, and, in our view, also the most risky. The parties must therefore focus on implementing a governance structure for the iteration process which is both efficient (in order to maintain agility) and effective (in order to minimize the risk).

There are some critical differences in risk profile between Waterfall and Agile, as a result of the iteration process. The compartmentalized and non-iterative nature of a traditional Waterfall project means that a great deal of the risk will allocated to the end of the project – i.e. during the critical integration and acceptance testing phase. As a result, a Waterfall contract will address, in detail, the procedures for testing the software and the parameters for accepting such tests. In contrast, during the iteration phase the free-flowing nature of the Agile process, where the design "roadmap" is not set out in advance, can lead to a significant risk of scope creep.

The process-oriented contract, therefore, should provide a governance framework for the iteration process which specifies the frequency of iterations, how iteration meetings should be conducted, and the process for agreeing on timelines, priorities, and functionality. Often, an early iteration will be the development of a prototype with minimum functionality as it will provide a solid base from which the customer and development team can work. Generally, the contract should require that mandatory functionality should be listed, the development effort for the items required should be estimated, and consent from both sides regarding the objectives of the iteration should be obtained, for each iteration. The contract should also specify when and how the dispute resolution/arbitration provision can be activated with respect to a dispute over running a given iteration.

---

[28] http://www.Agilemanifesto.org/

(f) **Testing**

"Working software is the primary measure of progress"

- Key principle from the Manifesto for Agile Software Development[29]

While, as with traditional Waterfall development, acceptance testing is also an important aspect of Agile development, unlike Waterfall, testing in Agile occurs over a different time horizon. In Waterfall, the acceptance testing is often a major undertaking, which may require thirty days of testing or more, and the results of which may trigger or delay payment. In Agile, where the iterations are relatively short, the deliverables are smaller, and the testing occurs throughout the project, the testing will be much more informal and may take an hour or a day. The contract should provide testing criteria that clearly set out when an iteration is deemed complete, and should clearly provide the consequences of passing and failing an acceptance test. It is important to note that, as the project progresses, a test of a given iteration may show that the iteration works well independently, but does not integrate properly within the system. Different consequences may be prescribed for such a situation. Final test results should also be documented and signed off on by representatives of both parties. The contract should also specify when and how the dispute resolution/arbitration provision can be activated with respect to a difference of opinion regarding an acceptance test.

(g) **Who is calling the shots?**

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation"'

- Key principle from the Manifesto for Agile Software Development[30]

Since Agile software development involves intense collaboration between members of the development team and members of the customer team, it is important that the contract set out a governance framework which specifies which individuals or classes of individuals working on the project may make key decisions. While the highly formalized and compartmentalized Waterfall process is hierarchical, the Agile process creates a flatter organizational dynamic. By definition the scope of an Agile project will evolve, and, at certain points, changes to the scope will be made and activities outside the scope will be carried out. Which representatives will have the authority to make changes—only the project leaders or any member of the team? Allowing all members of the team to effect such change may result in a more rapid and creative development process, but may also entail greater risk of error or disagreement. Reserving such decision-making power to the most senior member(s) of the team reduces agility but also reduces risk. The customer and the developer must discuss and agree upon the level of risk they are comfortable with in this respect. Generally, once the scope of the project is well defined, items within the scope can be dealt with verbally, and any disputes with regard to changing the scope should be dealt with through the agreed-upon change management process, or, if need be, the dispute management process.

---

[29] http://www.Agilemanifesto.org/

[30] http://www.Agilemanifesto.org/

(h) Documentation and Change Management

"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage"

- Key principle from the Manifesto for Agile Software Development[31]

As the process-oriented contract will focus on procedure rather than specifications, many of the specifications, and decisions regarding same, will arise through daily meetings between members of the development team. Due to the informal nature of the Agile decision-making process, it is important to consider what constitutes a valid agreement in the course of the development project. While Agile contracting processes advocate a low level of documentation and formality, a complete dearth of formality entails legal risk.

Simple documentation can go a long way, not only in providing some legal certainty, but also in solidifying priorities. We, therefore, advocate adopting, and incorporating into the Agile development contract, procedures that facilitate the inclusion of the meeting minutes into the agreement between the customer and the developer. Although it might be overly burdensome to draft a mini-contract for each iteration, taking clear minutes at each iteration planning meeting and having each party sign off on and file these minutes will create additional certainty with the imposition of a minimal additional burden. These minutes will assist in solidifying priorities and objectives, and in minimizing the opportunity for unwanted scope creep and future disagreement. As discussed above, the team members with the authority to change the scope must be specified in the contract, and it is those individuals who should be required to sign off on the minutes where the effect of the minutes is to alter the scope of the project.

The developer and the customer may agree on a set protocol detailing specific requirements for the minutes. Though the minutes should be as brief as possible, variation to scope should be clearly stated, changes required in the resources (human resources and otherwise) should be clearly described, and consequential changes to the rest of the system should be clearly set out – in effect, the minutes will act as a change order.

(i) Mini Arbitration

"Business people and developers must work together daily throughout the project"

- Key principle from the Manifesto for Agile Software Development[32]

While it is hoped that the risk mitigation strategies discussed above—i.e. the implementation of (a) the first date/due diligence process; (b) frameworks for managing iterations and testing; and (c) a clear allocation of roles and responsibilities such that having individuals with an agreed-upon level of authority sign off on documentation and changes—will reduce occasion for dispute, the parties still must be prepared to deal with conflict should it arise.

[31] http://www.Agilemanifesto.org/

[32] http://www.Agilemanifesto.org/

A mid-project "mini arbitration" clause should be included to deal with any deadlocks in the decision-making process. The sort of agreement that triggers such arbitration should be specified along with notice requirements. A list of technically qualified arbitrators can be agreed upon, and money can be allocated or even paid in advance, to keep one or several arbitrators on retainer to allow for speedy resolution. If the parties agree throughout, all or a portion of such retainer money as is never used can be returned to the general budget, and distributed between the parties upon completion of the project. Indeed, merely having an arbitration clause in place can encourage the parties to come to an agreement.

(j) Sharing the Risk: Fee Structures for Agile Projects

(i) *Alternative Fee Structures*

It is up to the parties to decide upon a fair compensation system that both motivates the developer and ensures that the developer has reasonable cash flow during the development project. Ideally, the risk should be shared in some fashion between the parties. In a Waterfall project, it is conventional for the project plan to include a series of payment milestones: for example, an initial payment to initiate the project; a mid-term payment; a payment upon receipt of the product for a acceptance, and a final payment upon the successful completion of the acceptance testing. In terms of proportionality, a customer will seek to have as much as possible of the payment allocated to the successful completion of the acceptance testing – for example, in the form of a payment worth 33% or 50% of the total.

Similar motivations exist for an Agile project, but the difference is that, without a preset set of specifications or "design roadmap", it is difficult to agree in advance on a series of payment milestones. Beyond that, many Agile commentators argue that Agile development is simply incompatible with fixed price projects, because successful fixed price engagements are dependent on fixed scope, and Agile projects by definition do not have a fixed scope. That being said, customers need to budget software projects, and therefore need some certainty built into the payment structure.

A number of solutions have been suggested to address this concern, including:

- *Estimate and fund by iteration only:* Given that each iteration is intended to produce some form of deliverable (recall: delivery of working software over documentation), this is a good way to manage costs while allowing for more flexible time and materials payments for each iteration. Disciplined agile methods such as Open Unified Process have built in "stage-gate" decision points which to facilitate this approach.[33]

- *Cost Plus*: The developer is paid for their costs only, but delivery bonuses are paid for working software, such that the developer will not make a margin unless they consistently deliver working software.

---

[33] "Agility@Scale: Strategies for Scaling Agile Software Development: Funding Agile Projects" by ScottAmbler July 30 2009), at https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/funding_agile_projects?lang=en

- *Cost-Sharing:* If there is a budget target, any amount by which the actual cost exceeds or is below the target (or past a specified acceptable percentage above or below) is be shared by both the customer and the developer, such that both cost savings and cost overruns are shared.

For Agile projects, Cost Plus and Cost-Sharing would best be applied by iteration in order to be successful fee structures.

(ii) *EVM Fee Structures*

Earned Value Management ("EVM")—a common project management technique for measuring planned expenditures, actual expenditures, and cost against planned performance—contains elements of the fee alternatives set out above, and can be employed to track the progress of an Agile development project.[34] Ultimately, EVM provides metrics for evaluating work actually accomplished and fee payments can therefore be tied to meeting certain thresholds on these metrics. Generally, EVM considers the following three dimensions:[35]

1. **Planned Value** (or "Budgeted Cost of Work Scheduled"; "BCWS"): a valuation of planned work, based on the cost budgeted for the work to be completed in a given time period;

2. **Actual Cost**: the total cost that was actually incurred in accomplishing the work during the given time period; and

3. **Earned Value** (or "Budgeted Cost of Work Performed"; "BCWP"): uses pre-defined "earning rules" (also called metrics) to quantify the accomplishment of work during a given time period.

EVM is perhaps best explained as follows:

- Whether or not the fact that Actual Cost is less than Planned Value (i.e. under-budget) or greater than Planned Value (i.e. over budget) at any one point in time is meaningless without assessing the extent to which the work was actually accomplished during that time period.

- More specifically, the fact that Actual Cost exceeds Planned Value at Week 10 may not be problematic if in fact the project has been completed at Week 10, when originally it was scheduled to be completed at Week 15 – i.e. if it has been completed five weeks ahead of time.

---

[34] See generally, David S. Christiansen, & Daniel V. Ferens, "Using Earned Value for Performance Measurement on Software Development Projects" (Spring, 1995) *Acquisition Review Quarterly*, DAU Press, at pg. 115-170; Glen B. Alleman, "Project Management = = Herding Cats: A Field Report, Agile Project Management, *PMForum*, online: www.pmforum.org/viewpoints/2003/0203Agilepm.htm; Tamara Sulaiman, "AgileEVM – Earned Value Management the Agile Way" (January 18, 2007), *Agilejournal.com*; Tamara Sulaiman, Brant Barton, & Thomas Blackburn, "AgileEVM – Earned Value Management in Scrum Project" *solutionsIQ.com*. Note a template Excel spreadsheet is available at http://www.solutionsiq.com/Agile_index.html.

[35] See Anthony Cabri, and Mike Griffiths, "Earned Value and Agile Reporting," Quadrus Development Inc., online: http://leadinganswers.typepad.com/leading_answers/files/Agile_and_earned_value_reporting.pdf.

- The third variable of "Earned Value" is intended to capture this third dimension of *technical progress/achievement*, and allows the parties to derive more useful conclusions from the available data.

- First, the **Schedule Variance** (Earned Value minus Planned Value), shows whether technical progress is ahead or behind schedule.

- Second, the **Cost Variance** (Earned Value minus Actual Cost), by ignoring the planned budget in favour of actual technical progress produced against actual cost, will indicate whether the the project was actually under or over budget, *relative to the amount of work accomplished* (rather than to the planned budget) since the start of the project.

- Third, the **Cost Performance Index** (Earned Value divided by Actual Cost) gives the parties the ratio by which the project is exceeding or is under budget, on an ongoing basis – i.e. if the resulting value is less than 1.0, it is apparent that project is proceeding over budget, as for every cent spent, less than a cent in value is being produced. For example, if the project is proceeding over budget and the Earned Value is $60,000 and the Actual Cost is $100,000, the Cost Performance Index will be 0.6.

- Fourth, multiplying the total Planned Value budget for the project by the Cost Performance Index then allows the parties to estimate whether, if the cost performance index was assumed to be a constant throughout the term of the project, the project will or will not be over or under budget. Thus, using the example above, if the total budget for the project is $1,000,000, the estimated total cost at completion is $1,000,000/0.6 = $1,666,667, or $666,667 over budget.

- Fifth, the **Schedule Performance Index** (Earned Value divided by Planned Value) provides the parties with the rate of technical progress of the project – i.e. if the resulting value is less than 1.0, then there is an overrun and the project is behind schedule. For example, if the project is proceeding ahead of schedule and the Earned Value is $100,000 and the Planned Value is $75,000, the Schedule Performance Index will be 1.33.

4. Sixth, dividing the total budgeted schedule by the Schedule Performance Index provides the estimated remaining time to completion. Thus, using the example above, if the total planned time remaining to completion is 4 (months or iterations), the revised estimated remaining time required to completion is 4/1.33 = 3 (months or iterations).

In summary, adding this third variable of "Earned Value" to capture the third dimension of *technical progress/achievement*, allows for the calculation of very helpful performance metrics which can then provide warnings of performance, scheduling, and budget issues.

How can EVM be applied to an Agile project? As we earlier referenced, in a Waterfall project the scope is defined at the outset, project milestones are set out, and a master schedule and budget are prepared. Scope changes are intended to be exceptions to the rule, and if there is a change, the high level of documentation makes it simpler to estimate the effect this change. With Agile development, however, the scope can change throughout and, therefore, calculating Planned Value can become particularly difficult.

To avoid this concern, rather than focusing on the Planned Value budget for the entire project, Cost and Schedule Performance Indexes can be calculated for each iteration. Instead of budgeting dollars and time, an Agile project can be evaluated based on planned features and budgeted iterations. This way, compensation can be tied to each iteration (if the customer is paying separately per iteration) or the ability to meet targets averaged over the course of the entire project. To ensure features are properly prioritized, the developer and customer can also agree on different weighting for different features in the iteration. Finally, to the extent that there are scope changes, the focus on doing the calculations on an iteration basis allows for more flexibility in taking account of such scope changes for each iteration.

To encourage agreement and open discussion between the developer and the customer, the rewards and penalties for meeting or failing to meet performance and scheduling targets could also be shared by both the customer and the developer. In addition to assisting in determining compensation, these metrics can allow the development team and the customer to more effectively track progress.

(k) Changing Horses: Exiting Agile Projects

As a final point, customers should recognize that Agile projects can very difficult to exit, whether through terminating or outsourcing[36] all or part of the project(s), for reasons which include the following:

(i) *Escrow arrangements have limited utility:* Without written specifications to deposit, the value of any source code escrow arrangements will be limited. As a result, the nature of the deposited materials will require greater scrutiny by the customer. The customer should ask for the escrow deposit materials to include, for example, working notes, and contact information for each of the developers.

(ii) *Non-solicitation covenants in the master agreement:* Again, given the lack of documentation, it is the personnel rather than the code which are the most significant assets, such that in the event of a termination customer would need to ensure that the developer waived any non-solicitation covenants in the master agreement, so that customer could have ready access to such developers, even if on a seconded basis

(iii) *Isolation from overall development process:* For high-risk projects, embed one or more customer "shadow" developers in the development

[36] Once the customer has the contract in place, customer may be tempted to outsource a section of the project to another developer who, for example, may be able to produce a given iteration for 20% less than your development team. Outsourcing is not optimal for customers involved in an Agile process. Given that one of the major benefits of the Agile process is having a small, highly skilled development team that knows the issues, each other, and the project very well, outsourcing a piece of the project runs the risk of demoralizing the team, disrupting the trust between the parties, and reducing the benefits of accumulated knowledge. In short, outsourcing within an Agile development process is not recommended.

team, in order to ensure that such developers have a general understanding of how the code is being developed. Also, ensure that customer personnel benefit from the high degree of interaction involved in an Agile project by taking appropriate notes.

5. CONCLUSION

One of our chapters herein is entitled "Agile and Risk Mitigation: Babies and Bathwater" in order to emphasize that it is facile to argue that the parties can "throw out" the concept of a contract as a natural corollary to "throwing out" the more disciplined and planned elements of Waterfall methodologies. Rather, it is important to note that, first, a robust contract remains an essential element of each an Agile project, and second, in order to ensure that the contract is indeed "robust" in addressing the risks of an Agile project, counsel needs to understand what are those risks, and their potential solutions.

We opened this paper with the results of the Forester study as to the prevalence of different software methodologies in the software development community, in order to highlight to increasing predominance of Agile. However, in some respects another result was even more significant: the second most popular response as to the software methodology of choice, at 30.6%, was "Do not use a formal process methodology"[37]. How to contract with developers using *no* formal methodology at all, will have to be subject of another paper.

[37] *Supra* note 2.

APPENDIX A: PRINCIPLES BEHIND THE AGILE MANIFESTO

From: http://www.Agilemanifesto.org/principles.html

We follow these principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity--the art of maximizing the amount of work not done--is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.