AGILE CONTRACTS PRIMER

Derived from the book...
Practices for Scaling Lean & Agile Development:
Large, Multisite, & Offshore Product Development with Large-Scale Scrum

by Tom Arbogast, Craig Larman, and Bas Vodde

Version 5

Please send us comments for future versions, at www.agilecontracts.org.
Note: Check website for latest version; share the URL (rather than file) to keep up-to-date.

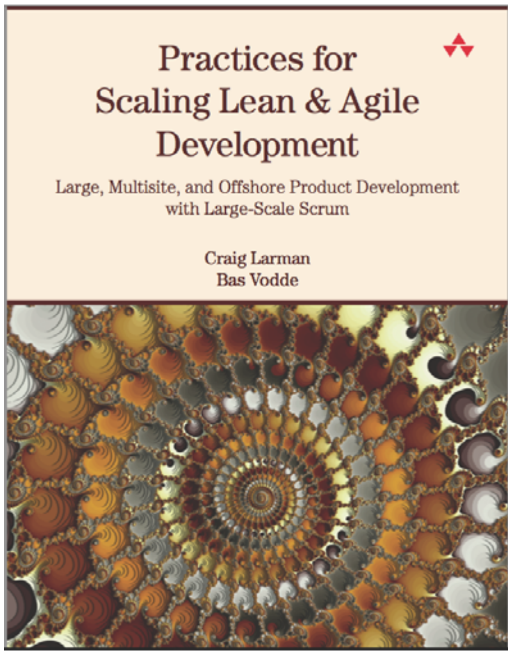

Practices for Scaling Lean & Agile Development
Large, Multisite, and Offshore Product Development with Large-Scale Scrum
Craig Larman
Bas Vodde

Table of Contents

Note: The “Try...” and “Avoid...” structure in this chapter suggests *experiments*, not formulas.

About the Authors

Tom Arbogast is a lawyer with years of experience in IT projects and contracts, combined with knowledge of agile principles, systems thinking, and lean thinking. He has worked *three* sides of the fence: (1) as a contract lawyer for *customers* of outsourced IT services, (2) as a legal professional for IT outsourcers (‘suppliers’), and (3) in business development for suppliers (sales agreements influence contract content). Tom leads the Professional Services and Legal Practice teams for ClearEdge Partners Inc., specializing in IT, M&A, Negotiation, Delivery Management and Operations. He served as a VP at British Telecom and in executive and legal positions in telecommunications and technology. He has worked as a lawyer in private practice and argued a number of cases before the Supreme Court of Canada. He is a graduate of the University of Colorado and University of British Columbia Law School.

Craig Larman and **Bas Vodde** are the authors of (1) *Scaling Lean & Agile Development: Thinking & Organizational Tools for Large-Scale Scrum* and (2) *Practices for Scaling Lean & Agile Development: Large, Multisite, & Offshore Product Development*.

They work as organizational-design consultants, and as management and engineering coaches in groups adopting lean thinking and agile development for very large systems product development, and in ‘offshore’ and multisite development.

For consulting or more information, please see *craiglarman.com* and *odd-e.com.*

Introduction

History will be kind to me, for I intend to write it.
—Winston Churchill

Companies have been successfully writing and using "agile contracts" or "evolutionary contracts" for many years. At Valtech (where Craig worked), they apply Scrum in the outsourced projects they take on—both in their Bangalore development center and elsewhere—and write contracts that support this. Other agile outsourcers, such as ThoughtWorks, have done likewise.

This introduction is written with two audiences in mind: non-lawyers and (contract) lawyers. We encourage sharing it with legal professionals since some of the material is written for them[1]—because most of the work in creating contracts that support agile values and practices is not in the language of the contract, but in *educating legal professionals* about these values. This involves understanding and appreciating traditional legal concerns, addressing those, and helping lawyers grasp the implications of agility and systems thinking. So the early suggestions focus on *understanding*. Later topics focus on a few concrete agile-contract suggestions.

Caution…

For every complex problem, there is a solution that is simple, neat and wrong.—H.L. Mencken

Do not assume that contract negotiations are much less complex or vigorous for legal professionals who grasp the implications of agile principles. It is important to recognize that contracting is an inherently complicated process, even more so in a domain of high complexity and uncertainty such as software development. And lawyers, by training and duty, must continue to pay close attention to the frameworks necessary to deal with a breakdown of trust and collaboration between parties.

1. This chapter summarizes core agile concepts already familiar to the expert agile reader, assuming legal professionals new to the subject are an important audience.

PART 1: THINKING ABOUT CONTRACTS

Try…Share these key insights with contract lawyers

The following points are central; they need to be clearly explored with legal professionals:

- The structural and legal aspects of agile-project contracts are the same as for contracts of more traditional development styles. The key difference is the approach to and understanding of *operational process* and *delivery* and how this is captured in or intersects with contracts.
- An understanding of *agile and lean principles* and *systems thinking* is necessary for contract lawyers. Why? Because applying these thinking tools leads to less risk and exposure, and that needs to be expressed in the contract. An agile approach enables rapid incrementally deployable deliverables and collaborative decision-making between the parties, and so relieves pressure on liability, warranty, and similar issues.
- Contracts reflect people's hopes and, especially, fears. Successful projects are not ultimately born from contracts, but from relationships based on collaboration, transparency, and trust. 'Successful' contracts contain mechanisms that support the building of collaboration, transparency, and trust. As trust builds between a customer and supplier, the commercial and contract model should 'relax' to support increasing "customer collaboration over contract negotiation."

Overriding fundamental insight

Everyone's number one priority is to deliver a successful project (in other words, to realize the business case), and each member of the organization, including legal professionals, must strive to reduce local optimizations, silo mentality, and wastes.[2] Other (legal) concerns are important, but subordinate to the goal of project success. This is a shift in mindset because many lawyers see their discrete *function* as the priority—that is, to deliver a 'successful' *contract*.

Try…Lawyers study agile, iterative, & systems-thinking concepts

A lawyer writing a contract for an agile project (most commonly, done with Scrum) needs to grasp the key ideas before she can articulate an agile contract. We suggest that legal professionals study introductory material in these subjects. For example:

2. Wastes: 1. Overproduction of features; 2. Waiting and delay; 3. Handoff; 4. Relearning; 5. Partially done work; 6. Task switching; 7. Defects (and related testing, inspection, and correction); 8. Underutilizing people; 9. Knowledge loss and scatter; 10. Wishful thinking.

- in the book *Agile & Iterative Development* [Larman03], chapter two, *Iterative & Evolutionary*, and chapter three, *Agile*
- *The Scrum Primer* (www.scrumprimer.com)
- the section on *Continuous Improvement* in *The Lean Primer* (www.leanprimer.com)
- articles on systems thinking; www.thinking.net has both articles and many links
- in the book *Scaling Lean & Agile Development* [LV08] chapter two, *Systems Thinking*
- this introduction

Try…Appreciate a traditional lawyer's point of view

Legal professionals are wired differently. This rewiring starts from the moment the student enters law school. The concepts of *Professional Responsibility* and *Advocacy* become ingrained into a lawyer's way of thinking. Legal professionals are trained to act, under legal ***duty***, to advance their client's interests and protect them against all pitfalls, seen or unseen. How do you define a client's interests? A *client* would probably say simply the successful delivery of the project. A *legal professional* will say she is successful if she protects her client to the greatest degree possible against exposure and risk, while at the same time advancing the end goal of the contract/project.

One has only to look so far as statutory definitions of a lawyer's duty to see how a lawyer perceives her role:

> *(5) A lawyer should endeavour by all fair and honourable means to obtain for a client the benefit of any and every remedy and defence which is authorized by law. The lawyer must, however, steadfastly bear in mind that this great trust is to be performed within and not without the bounds of the law.*[3]

So lawyers view their role as being there to *protect* clients from things they may not even know about. A lawyer is ostensibly trained to be *distrustful*—not necessarily of other people—but of unrealistic expectations and outcomes (the waste of wishful thinking), particularly at the start of a project.

It is important to appreciate this dynamic in the context of a contract negotiation. When a lawyer states that part of her role is to address—contractually—the point where trust deteriorates, it does not imply that the lawyer does not trust the other party. Rather, it means that she does not

3. The Law Society of British Columbia; Rules of Professional Conduct.

necessarily trust the *expectations* of the anticipated outcome and is mandated to deal with most anticipated outcomes—good and bad.

The third value of the Agile Manifesto is *customer collaboration over contract negotiation*. Naturally, when first reading this, a contract lawyer will take note, react, and perhaps think, "*That's nice, but I am here to ensure that my client is properly protected. She can think anything she wants, but I bet she wouldn't say she values collaboration over contract negotiation when everything goes south and a lawsuit is filed*." It is the lawyer's *duty* to consider the 'unthinkable' in contractual relationships and provide a framework—expressed in the language of the contract—for dealing with unpleasant outcomes. Lawyers are educated in, and all-too-experienced in, dealing with what happens when relationships deteriorate and trust fractures.

Stare Decisis[4]

Lawyers are creatures of habit. This comes from how the law has developed.

> *The life of the law has not been logic; it has been experience.—Oliver Wendell Holmes*

It is often said that law is behind the curve and not ahead of it. This is because of the very nature of how law develops. Cases are brought before courts and analyzed in the context of prevailing legal principles. This idea applies to all areas of law, including accepted contracting principles.

Once an issue has been reviewed and analyzed *ad nauseam*, including in legal academic circles, it will then be accepted into common practice. This process could take a decade or more.

Lawyers therefore look to past models that are tried and true, dusting off old precedents and looking to accepted law as a guide. Anything that is perceived as new or a sea change (for example, agile methods) is seen with skepticism and distrust. And this dusting off applies to contract models—it is easier to reuse an existing model than to create something new.

Traditional project assumptions: Impact on contracts

What do lawyers assume is the nature of software projects? First, it is common that they view it as similar to a *construction* project—relatively predictable—rather than the highly uncertain and variable research and development that it usually is. Second, that in the project (1) there is a long delay before something can be delivered that is well done, with (2) late and weak feedback, (3) long pay-

4. *Stare decisis* implies that precedent rules and will not be altered until an alternative is accepted by the courts; it applies principally to countries with a *common law* system.

ment cycles, and (4) great problems for the customer if the project is stopped at any arbitrary point in time. *These assumptions are invalidated in agile development*.

Naturally, these assumptions are expressed in the language of the contract, and in the time and attention lawyers give to concepts such as risk and liability for delay, termination, indemnification, acceptance testing, payment criteria, and warranty, amongst others.

Try…Debug common misunderstandings when lawyers are introduced to the third agile value

As mentioned, legal professionals new to agile values will react to first reading *customer collaboration over contract negotiation*. It is useful for non-lawyers to be aware of likely reactions and help address misunderstandings through discussion. And lawyers can correct these misunderstandings by studying these:

False dichotomies—The first and perhaps most common misunderstanding is to misinterpret the agile values in terms of a false dichotomy; that is, "customer collaboration is good and contract negotiation is bad" rather than, to quote the Agile Manifesto, *…while there is value in the items on the right, we value the items on the left more*. Legal professionals need to appreciate that this value does not mean that the contract is subrogated to the collaborative effort, but rather that collaboration is dominant for successful delivery of a project.

Not only should this collaboration be expressed in the behavior of the parties during project development, it can and should be expressed in the contract language—and behavior of lawyers. The contract can define a framework to encourage collaborative practices, and in this way the legal professionals can support their clients' goals of agility, and, most importantly, enhance project success.

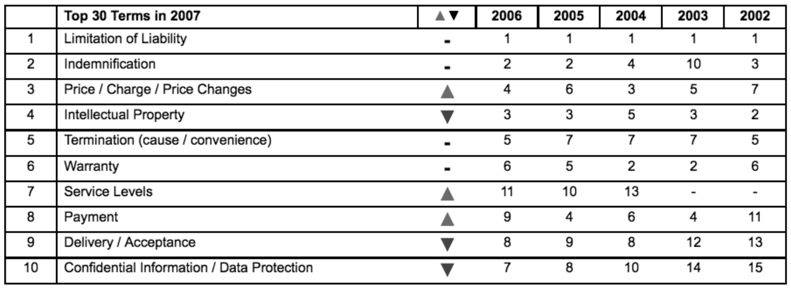
Non-legal 'contracts'—Another common misunderstanding is assuming that the third value is solely for *legal* contracts. But "contract negotiation" does not exclusively refer to business or legal contracts. It is meant to include the broader notion of agreements or specifications between parties in product development, and whether the emphasis is on *nailing down* these agreements or on ongoing collaboration, learning, and evolution. For instance, a traditional approach includes an early detailed specification of requirements and then "signing off" on these, which are then passed on to development teams for realization—a 'contract' of requirements.

Legal professionals may exacerbate or ameliorate, by the language of the legal contract, an unhealthy focus on these non-legal 'contracts' during project execution. For example, if they draft a contract that contains a clause requiring the definition of and the signing-off on the specification of all or most requirements before starting implementation, there is a lack of agility in the project and an undesirable emphasis on (non-legal) 'contract' negotiation.

Try…Lawyers study problems arising from silo mentality and lack of systems thinking

Figure 1.1 (from the International Association for Contract and Commercial Management, IACCM) depicts the top ten (of thirty) contractual terms corporate lawyers were concerned with from 2002 to 2007. It is difficult to imagine that delivery personnel are concerned on a day-to-day basis with most of the issues listed. And it is striking that *a description of the object of the contract (the project goal) is not mentioned*.[5] That is an astounding observation. The very thing the contract is ultimately about, the expectation of a deliverable (for example, software that will accelerate bills to be processed), is not in the top ten issues.

Figure 1.1 top ten (of thirty) contractual concerns of corporate lawyers



	Top 30 Terms in 2007	▲▼	2006	2005	2004	2003	2002
1	Limitation of Liability	-	1	1	1	1	1
2	Indemnification	-	2	2	4	10	3
3	Price / Charge / Price Changes	▲	4	6	3	5	7
4	Intellectual Property	▼	3	3	5	3	2
5	Termination (cause / convenience)	-	5	7	7	7	5
6	Warranty	-	6	5	2	2	6
7	Service Levels	▲	11	10	13	-	-
8	Payment	▲	9	4	6	4	11
9	Delivery / Acceptance	▼	8	9	8	12	13
10	Confidential Information / Data Protection	▼	7	8	10	14	15

Consider this scenario: A lawyer at a large company is asked to "measure the success" of contracts the legal department has entered into. The lawyer answers, "We entered into over six billion dollars worth of obligations over the past year encompassing over 400 different contracts, and we have only been sued, or had to sue, on two of those contracts. This is consistent with our year-to-year performance, amounting in my estimation to a 99%+ success rate."

In the traditional lawyer's world this is their definition of success, a 'best' or 'optimal' situation. But of course it is only *locally optimal*. The lawyer did not address if the business case behind the project was realized, if the consumers of the new software were delighted, if the project was delivered, or if too much had been paid over the life of any particular contract.

How do lawyers measure success with respect to a contract negotiation? There is a traditional saying regarding contract hardball that, "you know you have a good contract when both parties are

5. *Delivery/Acceptance* is referenced in item-9. However, this references the concept of delivery meeting a specified acceptance regime, and is not concerned with the underlying object of the delivery.

unhappy" because neither party got what it wanted. An agile mindset argues the opposite for both parties, and that a "win-win" approach is really what is mutually optimal.

But regardless of how one measures the 'goodness' of a contract, one thing will be constant, and it goes to the heart of a lawyer's fear in drafting a contract: If something goes wrong, the client will look to the contract (and therefore the lawyer) to ensure that the issue is covered in the client's favor. This fear, as well as expectation from the client, leads a lawyer to locally optimize strictly from the client's point of view with respect to *legal problem scenarios*.

This then comes down to the concept of local optimization, or the tendency of actors within a complex system to do the 'best' thing in the confines of their own duties and roles, without understanding the larger impact of their choices and actions or ignoring higher-level goals of the system.

The lawyer's response to the query about contract success was cogent in a local context but does not appreciate the larger systems issues. And why is this? There is…

- a wide gulf between the legal and delivery groups
- endemic silo mentality among contract lawyers
- a lack of systems thinking and resulting local optimizations
- measurement and incentives based on legal concerns

On this last point: Measurement and incentives not only inject dysfunction and locally optimizing behavior into project delivery, they do likewise in contract writing. If professionals in a legal department are rewarded on the basis of legal outcomes, there may be fewer legal issues—but not greater project success.

Form versus function

We have all been in buildings that, whilst beautiful and aesthetically pleasing from a distance, are dysfunctional and confusing internally. This is the difference between form and function. Any legal professional will tell you that, when a contract is finished and dusted, with the ink just drying in the signature boxes, the gleaming end product—which could be a meter-thick stack of document and appendices—is a beauty to behold.

But of course the real test of a contract is in the execution stage of the project, when the people on the ground are working together. During this stage, any need to refer to the contract is arguably a sign of failure—not only of collaboration but also of the legal professionals' ability to foster a framework for collaboration and success.

That said, if reference to the contract is needed, this is where it takes on a life of its own, much like a building. *It is thus critical to envision what the contract will be like in everyday use*. This view goes hand-in-hand with a systems-thinking approach.

All this then begs the question: How much time is spent negotiating different areas of the contract? Are legal professionals locally optimizing the concerns and language of the contract (reflecting their silo mentality) on necessary but secondary issues, and as a consequence actually increasing the risk project failure?

Many lawyers spend an inordinate amount of time and concern on ‘legalistic’ areas of a contract (for example, spending bone-numbing hours on areas such as force majeure and liability). These areas are certainly important to consider, but how important are they in the larger picture of ensuring the success of the underlying focus of the contract—the project?

There is an amusing story [Parkinson57] told by the British civil servant, C. Northcote Parkinson, illustrating his *Law of Triviality*: Time spent on any item of an agenda is inversely proportional to the cost of the item. He shares the story of a government steering committee with two items on the agenda: 1) the choice of technology for a nuclear power plant, and 2) the choice of coffee for the meetings. The government mandarins, overwhelmed by the technical complexities and science, quickly pass the technology recommendation of the advising engineer, but *everybody* has an opinion on the coffee—and wants to discuss it at length.

A similar dynamic plays out amongst lawyers writing project contracts: There is an inverse relationship between time spent on the terms that are being negotiated and what is being dealt with on a day-to-day level during execution of the project.

But there is good news with respect to negotiating issues: An agile and iterative approach can—by design—decrease risk. Therefore, pressure on negotiating “big issue” terms (such as liability) is alleviated because agile methods imply early and frequent incremental delivery of *done* slices of the system. The early feedback and delivery of a working system every two weeks (for example) fundamentally changes the dynamics behind negotiating some terms, whose excruciating negotiation in traditional ‘waterfall’ projects is driven by the assumption (and fear) of a long delay before delivery.

One can understand how extreme pressure comes to bear on articulating terms, when viewed in the light of a big “all or nothing” delivery model. Because of the small, iterative nature of deliverables in an agile approach and the ability to stop the project at any two-week boundary (since each incrementally small slice of the system is done and potentially deployable or ‘shippable’), there should be less pressure on concepts such as liability multiples and indemnity.

In *The Fifth Discipline*, Peter Senge states that *systems thinking* and a *learning organization* are ultimately aimed at building “*…organizations where people continually expand their capacity to create results they truly desire, where new and expansive patterns of thinking are nurtured, where collec-*

tive aspiration is set free, and where people are continually learning how to learn together." In this context, it is critical for legal professionals to acknowledge that the project contract is secondary, though admittedly necessary, to expanding that capacity. So it is critical for the Legal department to acknowledge that the contracts they craft can all too often degrade project success and degrade organizational learning because of a lack of systems thinking, a silo mentality, and local optimization on secondary issues—and this point holds true also for Finance and Human Resources, amongst other departments.

Try…Lawyers study the impact of potentially deployable two-week increments on assumptions and contracts

The Lexus LS versus the Lexus IS

Traditional non-agile projects envision an end product that is akin to buying a top-end Lexus LS. The final delivered solution has all the fine features, nicely polished. And—consistent with the car metaphor—lawyers probably envision an implementation approach in which one first builds the chassis, then drops in the engine, then the body and electronics, then the interior and paint. So you do not get to see how the final product all fits together until the very end, when all the components are assembled.

The corresponding pressure that this puts on contractual mechanisms designed to protect exposure is enormous. For a customer, it means that there is a delayed, complex, end-user acceptance regime that must occur after *final* delivery. And it means the customer cannot ascertain the quality of the 'car' until it is *finally* delivered. In software projects, this means that a customer will want to have maximum protection for the overall scope of the project—usually a liability multiple of the overall cost of the project. This means that a supplier cannot fully be comfortable with the deliverable until the end of the project, and may not therefore be able to recognize total order value until the final deliverable.

An agile project addresses both sets of concerns. It aims to build *not* partial components of a project iteratively, but rather to build a deployable working model of value to the customer that can be accepted and used at each two-week iteration. This is a critical point that legal professionals new to agile concepts do not always grasp; they may misinterpret agile development as incrementally delivering *undeployable* components rather than the agile model of delivering a *useful deployable* system after each short iteration, with gradually more functionality.

After the first iteration, the deployable solution or model may be characterized as a Lexus IS—a simpler entry-class vehicle. As each iterative solution is delivered, the level of the working model goes up in functionality and stature. In a sense, it is like a trade-in of the previous model every two weeks.[6]

The implications for this approach are critical for concepts such as liability and exposure. The customer has something of value that she has paid for and accepted. The supplier can be confident that it has delivered something from which it can recognize revenue and value. If there were, *heaven forbid*, a breakdown in the relationship and the project went *to hell*, each party will be nearly whole in terms of its relative exposure.[7] The customer will not be left having paid for a partial project that is now nothing more than shelfware, and the supplier will not be left with having expended effort on something that it will not get paid for. Granted, the ultimate expectations for either party may not have been met, and the partial system may not have enough functionality to usefully deploy,[8] but from a pure business and exposure perspective, the relative concerns of the parties are not nearly as extenuated as they may be in a traditional ‘waterfall’ project scenario. *It is vital for contract lawyers to appreciate the implications of this point in how they contemplate, negotiate, and draft project contracts!*

Try…Lawyers study how agility reduces risk and exposure

There are three general areas to be concerned with when drafting a contract:[9]

- risk and exposure (liability)
- flexibility to allow for change
- clarity regarding obligations, deliverables, and expectations

An agile-project contract may articulate the same limitations of liability (and related terms) as a traditional-project contract, but the agile contract will better support avoiding the very problems that a lawyer is worried about. That is, a contractual approach that embraces agile methods will actually *decrease* risk and *advance* a client’s relative interests. A contract that does not address agile meth-

6. This analogy is imperfect: Unlike trading in cars, software—and contracts—can *evolve* into something grander each refinement cycle.
7. This simplified analogy does not address the issue of expectation costs, consequential damages, lost profits, and other damages.
8. The well-done quality of the partial system makes it easier for another development group to pick it up and continue.
9. There are clearly many different aspects of a contract, but they can generally be subsumed into these three categories.

ods may actually do the opposite of what is intended and *increase* risk and *inhibit* a client's interests.

An example of this is in the area of requirements gathering and testing of the software that is developed to meet those requirements. In a sequential-development project, the lawyer will enforce (via the contract language) the client's wishes to articulate every possible case and attendant testing to meet the anticipated requirement.

An agile approach contemplates that requirements will be articulated in an iterative and evolutionary manner so that time and money is not wasted in developing software for requirements that are not ultimately needed. It also recognizes that money may be better spent for requirements that were *not* recognized at the beginning. Requirements identified and developed in a sequential-development project may never be used, because they were ill-conceived or lacked effective engagement with real users. And after delivery of a system that "conforms to the contract," requirements still need to be added to meet the true needs. From a contractual perspective, this means that a contract based on a sequential approach will actually increase the risk that the client pays more and gets less than she expects, and that the reverse will occur when the agile approach is understood and addressed in a contract.

This point cannot be overstated, both from a legal and financial perspective. In a sequential-development project, a client could pay much beyond the anticipated cost to get what she initially expected. The attendant contract will not protect against this scenario but will actually promote it by incorrectly assuming that it is quite possible to define and deliver a large set of requirements without ongoing feedback and evolution of understanding.

Contracts that promote or mandate
sequential life cycle development *increase* project risk.

For a legal practitioner, the implication is that agile principles can protect a client from things they may not know. This dovetails with the earlier recitation of the perceived duty of a lawyer to her client. Hence, once a lawyer knows about agile principles, she will be neglectful if she does *not* protect her client's interests by continuing to allow (by continuing to write traditional contracts) that client to pay for what she doesn't need and then allowing that client to pay extra to realize what she truly needed.

This means that an agile approach

- reduces risk because it limits both the scope of the deliverable and extent of the payment
- allows for inevitable change

- focuses negotiations on the neglected area of delivery

As an initial imperative, hands-on people from the business area (for the new system) and supplier development-team members must be closely involved and collaborating throughout the life of a project. Legal professionals are encouraged to look for signs that the parties have this intention in mind and to encourage it during contract negotiation and drafting.

This ongoing collaboration of customer and supplier does not mean that a lawyer has vast opportunity for further billables (or if one is internal, that a boatload of more work is now necessary) due to the increase in interaction and *joint discovery*. Rather, it means that contractual constructs must be created to allow for continual customer participation, assessment, and evolution. If the right model is created, a lawyer may have minimal further involvement—at least, unless conflict arises—because the right framework is in place to facilitate the cooperation inherent in an agile approach.

Try…Heighten lawyer sensitivity to software project complexity by analogies to legal work

If you are a coach or manager interested in increasing the appreciation among legal professionals as to the inherently complex, variable, discovery-oriented nature of software projects, try sharing the following thought-experiment with them:

“I want a fully complete project *contract* for my new project: A new enterprise-wide financial management system that will probably involve around 200 development people in six countries involving four outsourcing service providers never used before, and that takes between two and four years to complete. To the exact hour, how long will it take you to negotiate and write the contract with the four providers? To the exact word count, how many words will be in the contract? What will be the exact cost?”

Discuss the parallels between that scenario and software development, and what are realistic versus unrealistic, and effective versus ineffective ways to deal with uncertainty, discovery, and variability.

A lawyer will most certainly say that in this case it is impossible to ascertain, to any degree of certainty, what the end contract will look like, because of the evolutionary nature of contract drafting and negotiation. The lawyer may be able to give a ballpark figure in round numbers as to how much time, generally, it would take to negotiate a complete the contract, say, 200 hours, but would never commit to anything concrete in terms of actual total hours. Yet, ironically, lawyers and business leaders (in a waterfall mindset) expect that IT people will be able to ascertain, via requirements analysis and articulation, what a software project—something of far greater complexity, size, and variability than a “contract project”—will look like and how much it will cost, to a high degree of certainty.

Avoid…Incentives and penalties

It is common for those involved in contracts (legal professionals, sales people, procurement agents, …) to spend considerable time inventing, negotiating, and writing incentive and penalty clauses in contracts. There is an unquestioned assumption and belief that incentives (related to performance or target dates) or bonuses are beneficial. However, this is inconsistent with evidence-based management research [PS06, Austin96, Kohn93, Herzberg87], and there is ample evidence incentives lead to increased gaming, a reduction in transparency and quality, and other dysfunctions. Research was summarized in the *Organization* chapter of our book *Scaling Lean & Agile Development*.

Penalties (negative incentives) lead to the same problems.[10]

Incentives and penalties also foster a competitive us-them relationship between customers and suppliers, rather than cooperation.

Alternatives?

- simplicity—no performance-based incentives or penalties
- if the customer is extremely dissatisfied with performance, terminate the engagement at the end of iteration
- shared pain/gain models

Try…Share the pain/gain

Some upcoming sections, such as the "Try…Target-cost contracts" section on page 36, present models that share the pain or gain. This can foster collaboration and improving together. For example, in a target-cost model, if the actual cost is lower than the target, the customer pays less and the supplier profit margin is higher.

Avoid…"Quality Management Plan" and "Deliverables List"

Traditional *contracted* outsourced work involves low levels of transparency and trust, and a long delay until some software is *done*. One (of many) classic contract-responses to this is to mandate a conventional "quality management plan" or "deliverables list" that defines a long checklist of

10. We are not referring to major penalties for gross negligence, but to penalties connected to performance variation.

documentation to provide, rather than a focus on delivering real value: *working software*. One negotiator shared with us the following story:

> *An agilist involved in contract negotiation needs to be skeptical about extra documentation, and argue for measuring the cost of producing it—but of course be willing to discuss why an organization may require something. My experience [with deliverables lists] has varied, including in the worst case a very long negotiation with a company that had a huge internal chasm between the agile-friendly business people and traditional internal IT. The internal IT group had been "thrown a bone" by the business person who was the primary negotiator with us, by IT gaining agreement that a "Quality Management Plan" would be agreed to. The IT representative tried to reinstitute waterfall thinking and documentation, and traditional command-and-control, through various drafts of the 'quality' plan. Fortunately, we finally succeeded in effectively eliminating both the quality plan and the authoritarian non-value-producing "quality manager" role that the IT representative was trying to build for himself.*

What obviates the (assumed) need for a deliverables list, if anything? In Scrum, it is the *Definition of Done* that the supplier teams and client-side business-area Product Owner—rather than an IT manager or legal professional—define and evolve each iteration. Contract lawyers need to understand the Definition of Done because it changes how agile contacts are framed, and how projects are done. In short, the Scrum Definition of Done defines the "doneness" of the product increment each iteration in terms of activities and artifacts, and should be such that the product is potentially usable or shippable each iteration. For example, a Definition of Done for a particular product for a particular iteration includes "coded, integrated, functional/performance/usability tested, documented." (An actual definition is longer, with more detail and less ambiguity.) To reiterate, "done" is redefined by the supplier and customer at the start of each iteration, and it will adaptively evolve as both parties learn what is truly valuable.

Try…Collaborate early and often with lawyers

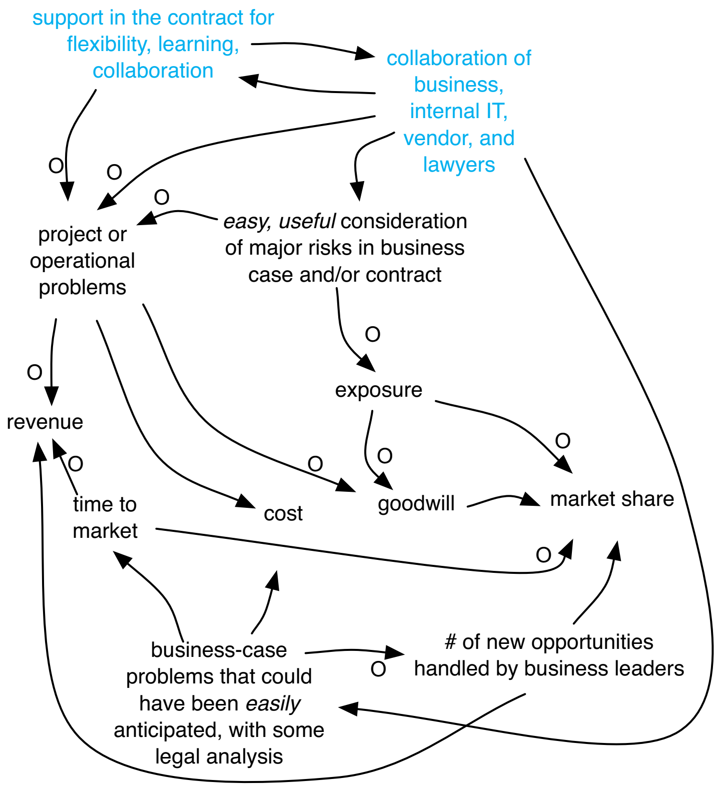
Collaboration over negotiation and *more and earlier feedback loops* apply not only to the customer and supplier in an outsourced contracted agile development project—they also apply to engagement with legal professionals.

The system dynamics model in Figure 1.2 illustrates, in broad terms, possible outcomes of increased support for flexibility and collaboration in contracts. But especially relevant to this section, Figure 1.2 also illustrates the impact of more and earlier collaboration of lawyers in business ventures and projects. This closer engagement is part of a larger theme explored in the *Teams* chapter of the book *Scaling Lean & Agile Development*: *cross-functional teams*. People often assume—wrongly—that the boundary of a cross-functional team is the people within the R&D or IT department. Not so. For example:

Cross-functional means that team membership includes all the key functions involved in the project, usually Engineering, Marketing, and Manufacturing, at a minimum. [Smith07]

Beyond "at a minimum" is the inclusion of Legal.

Figure 1.2 system dynamics of degree of contract flexibility and early lawyer collaboration


support in the contract for flexibility, learning, collaboration
collaboration of business, internal IT, vendor, and lawyers
O
O
O
project or operational problems
easy, useful consideration of major risks in business case and/or contract
O
O
revenue
exposure
O
O
O
O
time to market
cost
goodwill
market share
O
business-case problems that could have been *easily* anticipated, with some legal analysis
O
# of new opportunities handled by business leaders

The following is a case I (Tom here) saw where the impact of delayed lawyer engagement—and silo mindset—was costly: Business leaders identified marketing and cost-saving opportunities by creating a new web-based billing system. The business case hinged on developing it cheaply—which they *believed* could be accomplished by offshore outsourcing. Eventually, after a proposal and bidding process, a finalist was chosen and the parties started negotiations. This was and is usually the stage where legal professionals get involved to craft the contract terms and conditions.

Unbeknownst to the business leaders, many jurisdictions have recently tightened the rules regarding export of *personal data* across national boundaries. This only became apparent with the late engagement of some lawyers. Offshoring was infeasible due to these rules. The business case thesis—and project—was invalidated.

Fortunately, the mistake was caught before it was too late. But it would have been easy—given the limited and siloed engagement of the lawyers—to miss this issue completely, leading (as shown in Figure 1.2) to increased company exposure. And even though it was caught before project commencement, the initial work consumed significant business resources. In addition to the waste of this abandoned work, the subsequent reworking of a new business case and a new cycle of proposals and bidding reduced the time and energy that business people had to explore other business opportunities.

In addition to the value of cross-functional teams that include legal professionals, this case illustrates a point that IT people do not necessarily appreciate: A contract lawyer has a duty to consider *two* kinds of risks:

- internal project risks
  - these are reduced with agile development, and so it behooves legal professionals to support this contractually
- risks from knock-on effects (such as data export violation)
  - these are reduced by cross-functional teams that include legal professionals, and early, regular collaboration

PART 2: COMMON TOPICS OF AGILE CONTRACTS

Why No Specific Contract-Language Examples?

When drafting this introduction, we first considered including example clauses from agile-project contracts that have been created at Valtech, ThoughtWorks, and other parties. There are many corporate examples, in addition to variants such as the DSDM and Norwegian PS-2000 contract templates.

However, the feedback from lawyers who reviewed drafts of this introduction, and the opinion from our co-author, Tom, were consistent: *Copy-paste* is a real and present danger among lawyers and sales people, who—instead of grasping the underlying domain-specific principles (such as agile or lean principles) embodied in contract language—simply copy-paste clauses to draft new contracts. The legal professionals involved in this introduction had a clear message: Focus on principles that help educate both IT people and lawyers about the intersection of agile and lean development and contracts; sample clauses obscure what is important and invite avoiding the deeper understanding and systems thinking that contract lawyers will need to develop.

Topics Overview

The major *topics* of agile-project contracts (such as acceptance and termination) are the same as for traditional-project contracts. However, the *content* of these topics in the contract—and legal professional's mindset behind it—contains elements that support collaboration, learning, and evolution.[11]

Agility implies "responding to change over following a plan" and "customer collaboration over contract negotiation," how does this impact the following contract topics?

- delivery cycle	- project scope
- change control	- termination
- acceptance	- deliverables
- timing of payment	- pricing
- warranty	- limitations of liability

Delivery Cycle

For legal professionals new to agile development, it is imperative to understand the new delivery cycle. The cycle, from the start, is simply this:

- At the end of each two-week (or up to four-week) timeboxed iteration, deliver a deployable system with useful features.
  - it may have insufficient functionality to be of interest to deploy, but each cycle it is closer to *interesting deployment*

Incremental delivery is not a novel concept in contracts; many identify intermediate milestones, either fixed by date or by goals with associated acceptance criteria or a statement of work. The noteworthy differences for legal professionals to grasp regarding delivery cycle and milestones in agile development include

11. The special case of fixed-price, fixed-scope contracts is covered later.

- *doneness and deployability*—each iteration delivery is *done*, programmed, tested, and so on, and is in theory deployable
- *duration*—smaller, usually two weeks
- *timeboxing*—fixed time but variable scope

Project Scope

Agile contracts do not define an exact and unchanging project scope, although there are variations[12] in the degree of scope specificity and change, ranging from low to high. These variations are usually related to the pricing scheme, as will be seen.

Near one end of the spectrum are **target-cost contracts**, in which the overall project scope and details are identified at the start as best as possible (in order to establish the original target cost), but with mechanisms for change throughout. At the other end are **progressive contracts**, in which no (necessary) scope is defined beyond one iteration.

Summary of vision—In the contract

There are contract examples in which the *scope, vision, and business motivation* of the project or product is utterly inscrutable. Avoid that because it suggests that the contract framers are not involved in the project. Rather, they may be demonstrating legalistic, locally optimizing silo mentality—a weakness discussed earlier. Plus, a contract without a project overview is less comprehensible.

Therefore, invite the legal professionals to creatively write from their own understanding—not to copy-paste—a Moore-style vision statement [Moore91]. To achieve this, they will need to participate in project visioning (for example, during a workshop) and other project-engaged activities. Also, include a summary of the general contract, price, and payment model. Place both of these in the contract preamble. For example:

> ***For*** *Accounting and Marketing—****Who*** *want to consolidate bills, reduce billing costs, and do targeted marketing with bills—****Our new product*** *KillBill is a new billing system—****That*** *provides web-based billing presentation and payment, and customized marketing.* ***Unlike*** *our existing billing system—****Our new product*** *is web-based and has 80% lower operating costs.*
>
> *Contract Model: This is a target-cost model. The basis is an expected delivery price of $YYY. Supplier will deliver and be paid for, on an incremental basis of two-week iterations.*

12. Specific contract models, including target-cost contracts, are discussed in a later section.

Change Management

The issue of change is largely *inherently* addressed within the overall philosophy of an agile approach because of a re-prioritizable backlog and adaptive iterative planning; no special (traditional) change-management process, board, or request mechanism is needed. Indeed, it is critical for legal professionals to expunge old change-management language from contracts because such language may violate the essence of agility.

This does not mean that all kinds of change management are dispensed with in contractual form. Pertinent concepts in agile-project contracts fall under two categories:

- change in relationships between parties
    - For example, when a party is being acquired by another entity, a fundamental change in corporate direction may occur. Then, existing change-management language commonly used in contracts is likely still applicable. However, keep in mind that, because of the nature of iterative deliverables and concurrent payment inherent in an agile approach, there will be less pressure on relative expectations and the impact a major change will have on them.
- change in project scope
    - This area requires the most care in contracting, to prevent subverting the point of agile development: to make change easy and frequent in the collaboration between customer and vendor. Avoid mandating change-management boards, change requests, or special change processes.
    - But, as with project scope, there are variations in change-management flexibility, ranging from high flexibility without penalty when using flexible-scope progressive contracts, to medium flexibility with shared gain/pain when using target-cost models.

Also, see *Termination*…

Termination

The concept of termination is linked with change control in that an agile project should be amenable to changing course, to the point of actually *stopping further effort at the end of any iteration*. In contrast to conventional project thinking, legal professionals need to understand that *early termination should be viewed as a positive, desirable event in an agile project*, because early termination need not mean failure—it can mean that success was achieved quickly.

Arguably the ideal termination model in an agile contract is to allow the customer to stop, without penalty, at the end of any iteration.

Naturally, if the vendor has dedicated 100 people for an anticipated two years, and termination is unexpectedly much earlier, they likely have an expensive problem on their hands. Thus, agile-termination-clause variations include a sliding scale of penalty to the customer that reduce over time (and iterations).

Termination can be one of the most difficult areas to negotiate in any contract. The key mitigating differences in an agile approach is that (1) the customer has a working system each iteration, and (2) both parties will have clear and up-to-date views on the state of the deliverable. These are crucial points for legal professionals to grasp.

Acceptance

"Is it done?"—"What to do if not done?"—"We have now decided to change our minds and reject the iteration delivery from three iterations ago. Do you mind?"

These are vital questions in outsourced project work, and ambiguity around such issues is a possible source of conflict—and of litigation. Clarity (in so far as practically feasible) regarding *doneness*, *acceptance*, and *correction* both in the minds of the parties and the contract language should be a leading concern for legal professionals. They can help considerably in defusing the explosives in this minefield with careful attention to *negotiating* a contractual framework for acceptance that encourages *collaboration*.

Acceptance still exists, but is much simplified because of iterative delivery and acceptance, and because acceptance is incremental and adaptively agreed upon for each iteration. Further, agile practices usually include highly automated acceptance testing so that little or no manual (human) effort is required for validation.

Acceptance builds upon itself such that the final acceptance is the culmination of a number of acceptances that have occurred throughout the life cycle of the project, ideally most being repeatedly verified with automated acceptance tests.

In terms of contract work, this means that acceptance definition and negotiation does not have to be a massive comprehensive exercise; only the *framework for acceptance* must be contractually clear.

Broadly, for each iteration, acceptance is based on conformance to a prior agreed-on acceptance-test set, and in the case of Scrum, in conformance with the "definition of done."

Another element of acceptance in agile development—worth considering in the contract framework—is to include candidate users of the new system in the definition of acceptance and acceptance testing. Legal professionals concerned with a successful project should ask, "Are the right people—the hands-on users—involved in acceptance, and at each iteration are they collaborating with the supplier?"

Sample clauses

In contrast to this introduction's general avoidance of sample clauses, we decided an example in this case helps clarify the suggestion:

> *a) Customer and Supplier define acceptance of the Deliverable as follows:*
>
> *i. Deliverable passes all new automated and manual acceptance tests that were defined before the most recent iteration.*
> *ii. Deliverable passes all prior automated and manual acceptance tests, verifying that no regression has occurred.*
> *iii. Deliverable conforms to the "definition of done" that was defined before the iteration.*
>
> *b) Acceptance tests are incrementally defined together by Customer and Supplier members ("Acceptance Group"), including candidate users of the Deliverable, each iteration. The Acceptance Group reviews acceptance at the end of each iteration, starting at Sprint Review.*
>
> *c) Customer will have a period of half the business days of one iteration ("Evaluation Period", "Half Iteration") after provision to it of the final Deliverable to verify that the Deliverable or part thereof is not deficient.*
>
> *d) If Customer notifies Supplier in writing prior to the expiration of the relevant Evaluation Period that the Deliverable or part thereof is deficient in any material respect (a "Non-conformity"), Supplier will correct such Non-conformity as soon as reasonably practical but no longer than the length of one iteration, whereupon Customer will receive an additional Half Iteration period ("Verification Period") commencing upon its receipt of the corrected Deliverables or part thereof to verify that the specific Non-conformity has been corrected.*
>
> *e) Customer will provide Supplier with such assistance as may reasonably be required to verify the existence of and correct a reported Non-conformity.*

Limitation of Liability

Negotiation of liability clauses is perhaps the most difficult area in any contract negotiation, and an agile approach does not change that. However, it may help. For instance, it can attenuate liability because there is a usable deliverable at the end of each iteration.

For example, a defect in an iterative deliverable may have a lesser impact in operation because the negative consequence is discovered sooner. This does not mean there are no knock-on effects that never have to be addressed through the liability paradigm, but the consequences could be less.

Consider a case that I (Tom here) came across: In a traditional sequential life cycle project for a new billing system, it was discovered, after the “one big delivery at the end,” that duplicate and erroneous charges were sent to *many* customers. The fallout and extra costs were considerable: the company had to cut new bills, offer rebates, and repair its image with customers—plus paying to correct the underlying problems. There was then an ensuing fight with the external supplier as to who should pay for the damages—liability.

In an agile approach, the same problematic bills could be sent. But it is also possible that those bills would be sent early to a much smaller subset of customers, using an early release of the system with just-sufficient functionality to field-test this critical feature.

This would reduce cost, exposure, and damage to goodwill. It might also be cheaper to fix because the system would be smaller with fewer entanglements between its software components.

Hence, liability may be attenuated with agile development.

Warranty

Similar to liability, the concerns related to warranty are attenuated in an incremental approach; the risk profile associated with the final warranty is considerably less because of the confidence and acceptance in the deliverable itself, due to incremental acceptance. This is especially enhanced if automated acceptance testing is employed.

As with liability, warranty should be tied to each incremental working deliverable (at the end of each iteration), though there is still an overall warranty to the final product.

Deliverables

Traditional project contracts often include a detailed, prescriptive list of what should be delivered (many documents, …), and how acceptance of these artifacts is accomplished. These details are sometimes embodied in a statement-of-work or “quality plan” appendix. Avoid such specificity and rigidity—avoid including a detailed deliverables list in the contract. Why?

- It leads to an increase in waste activities rather than a focus on working software, and there is a presumption—possibly untrue—of knowing what artifacts are valuable.
- There is a focus on negotiating and conforming to “quality plans” rather than cooperating to create useful software.

- It reinforces (the illusory) command-control predictive-planning mindset rather than learning and responding to change.
- It reinforces the (untrue) belief that a fully defined system can be predictably ordered and delivered as though it were a meal in a restaurant rather than creative discovery work.

All that said, we have seen custom software for which the *source code* was never provided by the supplier—somebody forgot. So, there are cases in which the customer does not at first understand what is critical. But in such cases, discovery of valuable deliverables can be more simply achieved through frequent incremental delivery and deployment, rather than through a contract deliverables list.

On occasion, technical documentation to support maintenance is valuable—usually as a learning aid for people new to the system—and its delivery is often specified in a traditional project contract. Yet, maintenance of a recently deployed system is frequently outsourced to the same people that created the system and so have less need for such documentation. Therefore, it could be wasteful to require it as an early deliverable. If, at some future time, there is a *demonstrated* need for documentation for the customer (for instance, if the customer takes over the maintenance work), then it can be created by the supplier, perhaps in a joint agile-documentation workshop with the customer *after the system is finished*.

Timing of Payment

Perhaps the most popular system is to pay each iteration, once there is final acceptance of the deliverable for that iteration. In the simple case, such as with basic progressive contracts, payment is 100% of the agreed iteration price. More complex payment schemes are usually tied to more complex overall project pricing schemes. For example, in the various “shared pain/gain” systems such as target-cost contracts, in addition to iteration payments there will be a final deferred payment at project end. Or, at each iteration there may be an X% holdback that accumulates and may be paid at various intermediate milestones.

Pricing

Time and materials

Variations of time and materials (T&M) make for good agile-project pricing models: simple, straightforward. Recommended. Note that T&M applies to both fixed- and flexible-scope contracts.

One traditional concern with T&M, common on sequential development projects, is that customers are locked into a seemingly endless cycle of payments before they see useful results. Another classic concern is whether customers are getting good value for their money. These concerns are ameliorated in an agile approach with a usable system each iteration—progress measured in terms of usable software features, high transparency, and termination that can occur at the end of any iteration.

T&M requires trust and transparency between the parties. That takes sincere effort and time to develop. On several projects, Valtech India has started with variations of fixed-price contracts, and after building trust, has been able to move to variations of T&M models with their clients.

Several variations of T&M limit the customer's exposure and/or balance the pain/gain. For example:

- capped ("not to exceed") T&M per iteration
- capped T&M per project or release
- capped T&M per iteration, with adjustment—For the next iteration, the price is capped, but if all original iteration goals are delivered and accepted, at a T&M cost below the cap, there is an adjustment payment to the supplier, such as one half of the savings below the cap. A similar shared pain/gain pricing scheme is used in the project-level *target-cost* model.

Fixed price per iteration (per unit of time)

This model has the virtue of simplicity and predictability, and is not uncommon among agile outsourcers. There are two key cases:

- requirements defined and agreed-on before the iteration
- highly flexible or no predefined requirements

In the first case, the issues are identical to fixed-price per large project…The supplier has to clarify the work and have sufficient confidence in the estimates in order not to lose money. The small scope of an iteration makes this much more likely than for a large project. The key issue (or cost) for customers is that the supplier adds a contingency fee to the rate because of the risk associated with variability in research and development work.

In the second case, the key issue is customer trust in the supplier. Transparency, frequent delivery, and easy termination help.

Fixed price per unit of work

Several agile outsourcers offer a fixed price per unit of work (UoW) model. In contrast to traditional development, where a UoW might mean a document or other incomplete solution, an agile UoW reflects the seventh agile principle: *Working software is the primary measure of progress*. That is, the UoW is related to running, tested software features.

These models go by various names and with various systems to estimate a UoW: “price per story point,” “price per function point,” “price per feature point,” and so on.

Points of size versus value—In the schemes we have seen, the ‘point’ is always related to an estimate of size or effort—and so, related to *cost*. Although vendors may claim the price model is *value*-related by using modern agile-sounding terminology such as “price per point,” it is not accurate to say that a “story point” or “feature point” is a value measure—in the idiom of “bang for buck” a *point* reflects *buck*, not *bang*. However, there does exists *in theory* the ability to define points in terms of *business value impact*—where this is measured using a system such as Tom Gilb’s *impact estimation tables* (and he has proposed such a value-impact price model [Gilb05])—but we do not know any application of this approach.

We have seen agile outsourcers use one of two schemes to determine the fixed price per point: (1) an average based on several previous projects, and (2) a customized amount. In the latter case, the customer pays the supplier-average point value for a few iterations (or pays T&M, or …) during which time detailed costs are tracked. Then the supplier and customer agree to a custom fixed price per point, based on this cost plus some profit margin.

This model is congruent with agile and lean themes of being delivery- and value-oriented: Assuming that an agile UoW is loosely related to value to customers (which is not always true), they pay proportional to value received. However, since in the schemes we have seen that a point is related to size or effort rather than true value impact for the customer, this *point* is somewhat lost.

A key issue to attend to in this model—and one that needs consideration in the contract—is a clear and common (for customer and supplier) framework for defining a *point*. For example, only function points are relatively unambiguously defined—and can be identified and verified by certified function-point analysts. In contrast, story points (also known as relative effort points) have no *independent* meaning.

Pay-per-use models

XPLabs (an agile-development company in Italy) promotes pay-per-use contracts with their customers. Every ‘use’ (usually, a transaction) of a custom-built or pre-built system that is deployed for a client is automatically tracked by XPLabs. The customer is regularly invoiced based on frequency of use—a simple payment model. The approach tends to align the interests of the customer and supplier, and both parties win if the system is increasingly used.

If it is a pre-built solution, the model is especially attractive to customers: they have no maintenance or update costs, and only pay extra for custom enhancements. Each new customer deployment is based on the same, simple contract model.

If it is a custom-built solution for only one customer, the contract model for the development work may be any of the other approaches discussed here, such as T&M, perhaps at a below-average rate to adjust for the future anticipated pay-per-use revenue.

Hybrid shared pain/gain models

There are numerous hybrid pricing schemes in business, well known and not repeated here (see the recommended readings). One hybrid shared pain/gain model applicable to agile development was proposed by Bob Martin [Martin04]:

Discounted fixed price per unit of work, plus discounted T&M—For example, assume the following project scenario:

project estimate	average velocity (140 people, 2-week iterations)	original person-day estimate	payment if $500 per person-day
100,000 points	4,000 points	35,000	$17,500,000

In this model, a lower person-day rate is offered, with a complementary per-unit-of-work rate. For instance, assume a standard person-day rate of $500. The supplier offers a discounted price per person-day of $150, and a discounted *price per point* of $122.50.[13] Then:

actual person-days	actual customer payment	change in estimate-to-actual effort	change in estimate-to-actual payment	*effective* person-day rate
30,000	$16,750,000	-14%	-4%	$558
35,000	$17,500,000	0	0	$500
40,000	$18,250,000	+14%	+4%	$456

Observations:

- If actual effort equals original estimate (35,000 person-days, 100,000 points), the customer payment is equal to a simple T&M scheme at $500 per person-day.
- If actual effort varies, customer payment varies less severely.
- As with target-cost and some other adjustment schemes, the customer and supplier are sharing the pain or gain if the project takes more or less effort than original estimate.

Fixed price per project and target-cost pricing

Both these pricing schemes are covered in the next section. Their impact extends beyond pricing, to overall contract or project model.

PART 3: CONTRACT MODELS

Humans have been writing contracts since the dawn of time, encapsulating their hopes and fears. There are myriad models and variations on those—see the recommended readings for a broader treatment. This section focuses on common models and their variations that customers and suppliers in agile projects will frequently see or consider.

Avoid…Fixed-price, fixed-scope (FPFS) contracts

Fixed-price, fixed-scope—and worse, with fixed duration—contracts and projects tend toward lose-lose situations for both the customer and supplier; customers often do not get what they really need, and suppliers can easily lose money. And in an effort to deliver something within the constraints of price and scope, suppliers will often degrade the quality of their work—reduced code quality, less testing, and so forth. All this leads to an increase in future costs for customers, who will eventually have to pay for the sins of the past, as follow-on change requests[14] to get what they truly need and as increased maintenance costs for software of low quality and high "technical debt."

Fixed-price bids have added a large risk contingency (a percentage of estimated cost as high as 50%) to the overall price—this premium is usually hidden in the effort estimate.[15] This leads to

13. The price $122.50 is derived: Given a person-day rate of $150, 35,000 person days, and 100,000 points, it is the price per point needed to reach a total payment of $17,500,000.
14. Traditional offshore outsourcers in India enjoy this change-request model because they make so much profit from it; they know very well that their FPFS contracts do not deliver what the customer really needs, and they look forward to the 'rent' (as they call it in India) they obtain from evolving the unsatisfactory system to meet the true needs.

a reduction in transparency and increased gaming during project execution because the supplier wants this premium as profit rather than consumed budget. Plus, since the early requirement specification that is signed off is almost never what is actually needed (due to myriad factors in this inherently complex domain), the supplier generates further revenue—in India, outsourcers call this 'rent'—through a series of follow-on change requests, each for an additional cost beyond the original fixed price.

Fixed-price projects have been promoted under the guise of various local optimizations (other than project success); we encourage legal professionals to watch out for these:

- As a customer, most important is to know the cost, for financial reporting or budgeting.[16]
- As a supplier, most important is to book the total order value.
- As a sales person, most important is to book the total order value, to get full commission.
- As managers, most important is to avoid using time on the project. We want to order something, go back to other work without 'interruption,' and then get it delivered at the end.

But there are companies, that for one reason or another, still try. In that case, the most frequent question we get is, "How do you do fixed-price, fixed-scope projects with an agile method?" First, it is possible; Valtech India and other agile outsourcing suppliers have done so (because of market demand)—though this is their least-favorite model.

There are often two misunderstandings behind the question of FPFS and agile methods:

- The first misunderstanding is that the overall project release requirements are not known or estimated before the first iteration when an agile method is used. Not true. Rather, in Scrum, before iteration-1, there may be initial release planning ("initial Product Backlog creation") in which all identifiable release requirements are clarified and estimated.
- The second misunderstanding is that requirements must change with agile methods. Not true. Rather, all agile methods provide the opportunity and mechanism to support learning and change, but do not require it. Scrum can be used with a fixed-content Release Backlog—and still provide benefits, thanks to better and more frequent feedback about ways of working, technologies, test results, and smaller batches.

With Scrum or any other approach, there are keys to avoiding ruin when taking on an FPFS project:

15. For this and other reasons, FPFS contracts are also called "latest date, most cost" contracts.
16. See the *Beyond Budgeting* section of the *Organization* chapter in the companion book, for an alternative to traditional budget processes.

- Apply the best possible due diligence in terms of large, detailed upfront requirements analysis, thorough acceptance test definitions, skillful effort estimation of all requirements, all done with experienced people.
- Do not allow any changes in requirements or scope, or only allow new requirements to displace existing requirements if they are of equal effort.
- Increase the margin of the contract price, to reflect the significant risk inherent in FPFS software development—a domain that is fraught with discovery, variability, and nasty surprises.
- Employ experienced domain experts with towering technical excellence.

Note that a lean culture of long-term, hands-on great engineers and manager-teachers who are experts in the work and coach their teams provides the environment to build the experienced people necessary to reduce risk on FPFS projects.

Payment timing

Payment timing for FPFS is usually per iteration, with a final lump sum on completion (if total payments were not previously exhausted). The per-iteration amount is a fixed percentage of the overall price, either based on an estimate of total number of iterations or if the project is also fixed, on the duration of the predefined number of iterations.

Flexibility in FPFS projects with Scrum

There are several areas of low-risk increased flexibility when executing a FPFS project with Scrum; the ability to…

- displace existing requirements with new ones of equal effort
  – this *replaceability* option is important to highlight
- change the order of implementation of the fixed requirements
- improve the “definition of done” each iteration

Schwaber[17] also suggests two other contract provisions:

- Customer may request additional releases at any time, priced with T&M.

17. Ken Schwaber is co-creator of the Scrum agile method.

- Customer may terminate early if satisfied early, for a payment to supplier of 20% of remaining unbilled value.

Legal professionals need to be aware that this flexibility can and should be expressed in clauses of an FPFS contract.

Should we do FPFS projects with a sequential, traditional approach?

A question that is sometimes asked is, “If we have to do an FPFS project, should we use an agile method or a sequential life cycle (waterfall, …) and traditional approach?”

There is evidence that sequential life cycle development is correlated with higher cost, slower delivery, lower productivity, more defects, or higher failure rates, compared with iterative, incremental, or agile methods [MacCormack01, Reifer02, DBT05, MJ05, CSSD05, AV07, PRL07].

Consequently, the *last* thing you want to do with an FPFS project is make matters even worse by applying a traditional sequential development approach.

Quite the opposite: If you execute an FPFS project with Scrum, you will have less waste, less queues, less WIP, and you will gain early realistic feedback about the true nature of the project. Based on that early feedback, you can *adjust early* rather than late. Especially in FPFS projects, you want to know *how bad things are as fast as possible*; agile methods enhance that feedback.

There is a more subtle advantage to using an agile approach on an FPFS project: It may evolve into a collaboration-oriented flexible project. Many customer stakeholders understand that the FPFS model may not solve their problems, but it was imposed on them—perhaps by the Legal or Finance arms. Once customers start interacting directly with the agile supplier on the ostensibly fixed project and see rapid delivery every two weeks of a well-done solution, and realize their ability to change the iteration-order of implementation of their fixed requirements (and to replace requirements), and trust and collaboration builds with the supplier, ‘fixed’ can become ‘flexible.’ The customer relaxes, sees the advantages of “customer collaboration over contract negotiation,” and agrees to pay less attention to the original definition and more attention to evolutionary development to meet their real needs.

Contract Evolution on a Large Agile Project

with Greg Hutchings

An example of multi-phase variable-model contract evolution was a three-year 15,000 person-day project with Valtech (India) and a retail customer. There were four contract phases, created (not pre-planned) in response to learning and adaptation: (1) FPFS per project, (2) progressive, T&M per iteration, (3) progressive, fixed price per unit of work, and (4) progressive, capped T&M per iteration.

(1) FPFS per project—The customer was only familiar with sequential life cycle projects, and new to Valtech—trust was low. Therefore, they expected (and got) a traditional FPFS contract based on a sequential life cycle, with an agreed deadline that was not contractually binding. Despite substantial effort to validate specifications and estimates, much—as usual—was discovered to be unknown and misunderstood. After the first year, costs exceeded budget and delivery was months beyond the *wish*. But, some trust had developed by Valtech's effort to be transparent and responsive, and so the customer agreed to replace the FPFS contract.

(2) Progressive, T&M with bonus/penalty—During the first year, the customer was gradually exposed (by Valtech) to agile development principles, and so was open to a new kind of development and a new contract: a progressive variable-scope contract based on Scrum, priced with T&M per iteration, adjusted (at customer request) with bonus/penalty clauses for (1) quality of iteration deliverables, and (2) velocity. Not surprisingly ([Austin96]) these adjustments subtly affected developer's behavior (a reduction in transparency, more gaming) to "avoid penalties." Progress (now with Scrum) was *much* faster, and confidence and trust improved with a release each iteration and close customer collaboration. (*continued…*)

Finally, after an initial FPFS contract has been completed with Scrum, the customer may be willing to use one of the alternative agile contract models for a later project. Several agile outsourcers have experienced these positive patterns with their customers.

Try…Variable-price variable-scope progressive contracts

In their purest form, progressive contracts[18] imply completely flexible scope that is adaptively defined each subsequent iteration. They are a good candidate for agile projects

18. Also known as, or a variant of, *open-ended variable scope*, *open-ended incremental*, or *indefinite delivery indefinite quantity (IDIQ)*.

(*continued…*)

(3) Progressive, fixed price per unit of work—In the third contract phase, pricing changed to fixed price per use-case point (UCP), because the client felt it could provide better value for money. Further, the bonus/penalty element was removed. UCP was chosen because the customer was very familiar with use cases, and wanted a unit of work estimate related to that. Payment varied each iteration, based on the number of UCPs delivered. Also, by this time, the customer was comfortable with iterative, collaborative acceptance-test definition to drive iterations and acceptance, and this, combined with better broad-theme integration testing, smoothed and improved acceptance each iteration. This contract form was considered by both Valtech and customer the most successful. However, the unit of work estimation method (UCP) required more upfront requirements analysis than otherwise necessary—a rolling wave of detailed use-case specification occurred usually about two iterations before implementation.

(4) (Support phase) Progressive, capped T&M—The final contract form and phase was established after product deployment, for support and minor enhancement. The customer support budget was fixed (per annum), and the scope of work variable. Therefore, capped (per month) T&M was acceptable—simple to administer, and a high level of trust had been established.

[Poppendieck05]. They are master service agreements or umbrella-framework contracts that define the overarching relationship and pricing scheme per iteration, but do not define scope. Progressive contracts do not define a total fixed project price—although one variation has a project cap.

Customer exposure is controlled because termination can occur at the end of any iteration—with a working system. If both parties are happy with the relationship, progressive-contract projects can continue indefinitely.

Usually (but not required) before each subsequent iteration, the customer and supplier define the goals of the upcoming next, perhaps with acceptance tests. Sometimes—recurrent at Valtech—the goals for iteration N are clarified during iteration N-2.

Pricing per iteration runs the gamut of variations: fixed-price per iteration, T&M per iteration, and so on.

Variations—At Valtech, a *capped-price variable-scope progressive* contract is common; there is an overall project price cap. Pricing per iteration is any variation, such as T&M. Also frequent is a *capped-price variable-scope progressive contract with a non-binding Release Backlog*, in which the parties create—*before the contract is written*—a backlog of release goals. This backlog is included as an appendix to the contract. However, it is agreed that *nothing* in the original backlog is binding.

(This is classic Scrum.) Why create the non-binding Release Backlog before the contract is written? It is used to estimate the overall release cap and to provide a starting-point common vision. It is also common to create this non-binding Release Backlog in a prior, separate, *contracted* phase.

Progressive contracts are a common model with agile outsourcers and the long-term customers with whom they have built a relationship of trust. A frequent pattern (not a recommendation) is

1. early contracts that are variations of fixed price and fixed scope
2. later, a shift to progressive contracts with simple T&M or capped T&M per iteration

Try…Increase flexibility in project and contract variables

A variable-scope, variable-price, variable-date, pure-progressive contract is flexible. Any variable (scope, price, date) can vary in flexibility, depending on the level of trust and collaboration between customer and supplier—or otherwise constrained, such as by government regulation. Contract variations that agile outsourcers have created include the following:

Capped-Price,[19] **Variable-Scope**—Discussed in the previous section.

Capped-Price, Partial-Fixed-Scope—A relatively small set of requirements are fixed—leaving room for learning and adaptation.

Fixed-Price, Variable-Scope—The *optional scope* contract [BC99] is a variation of this model and also fixes the end date.

Bounding risk in flexible contracts: The multi-phase model

The multi-phase model is described in the “Try…Multi-phase variable-model frameworks” section on page 39. Briefly, it implements a longer project from a series of shorter contracts.

If trust is low, customers can bound their risk (and fear) by using a series of short-duration, flexible contracts. For example, one *year-long*, fixed-price, fixed-date, variable-scope contract may be viewed with trepidation. But a series of *two-month*, fixed-price, fixed-date, variable-scope contracts—with the ability to terminate at the end any cycle—is more palatable.

In addition, after a few contract cycles, trust can build. At that point, the customer may shift to a simple progressive contract with T&M pricing per iteration.

19. In this section, *price* refers to overall project price.

Models That Share or Adaptively Shift the Pain/Gain or Risks

There are potential risks and rewards in a project—for both parties. And these may shift over time. For example, FPFS projects appear to shift much of the risk to the supplier, although this is an illusion for reasons previously identified.

Some frameworks, explored next, have been explicitly crafted to share these risks and rewards and to shift risks to the appropriate party that can do something about them.[20]

In the best case, these frameworks engender an increased alignment of motivations for the parties since they both have "skin in the game." And they may improve fundamental fairness and relationship building. This philosophy is at the heart of the concept of a win-win approach, and it will create the trust and relationships that will foster further business.

But contracts do not themselves create trust and alignment. In the worst case, these kinds of contracts are abused as part of a blame-game, to shift pain to the other party and only individually gain.

These frameworks include

- target-cost
- multi-phase variable-model
- profit sharing

Try…Target-cost contracts

Target-cost contracts can help align motivations of both parties. They are used in Toyota with their suppliers, reflecting the pillar of *respect for people* in lean thinking, in which Toyota tries to build stable long-term relationships with suppliers, based on trust and mutual support.

This model assumes an initial release planning step in which overall project scope is identified. This is part of *stage one* to establish the **target cost**:

1. In collaboration between customer and supplier, identify, analyze, and estimate all possible project requirements.
2. In collaboration, estimate the cost of change or scope increase during the project. This is important; target-cost contracts must *realistically* account for overall effort and cost as best as possible.

20. That usually means placing requirement-related risks ('what') in the hands of the customer, and placing implementation and technical-related risks ('how') in the hands of the supplier.

3. From these two elements, establish the **target cost**.
4. Calculate **target profit**, based on *target cost* (for example, 15% of target cost).
5. Share all details and results with customer (this is important).

During *stage one*, keys to the success of this model are (1) a best-effort, no-wishful-thinking target cost generated with skillful due diligence, and (2) (supplier) open-book costing, so that the customer transparently sees all details leading to the calculation of target cost.

Stage two in a target-cost contract is project execution—for example, with Scrum. As will soon be appreciated, a vital practice for success is tracking all *actual costs* as they are incurred (for example, developer time spent, meetings, hardware), and *transparently sharing* all cost information with the customer in near-real time.

The key aspect of target-cost contracts is a shared pain/gain formula for an adjustment related to the difference between actual and target cost. There are several variations in the formula.

In the simplest case, an example:

Adjustment = (ActualCost – TargetCost) * CustomerShareOfCostDiff
CustomerPayment = TargetCost + TargetProfit + Adjustment

As will be seen, *Adjustment* may be positive or negative.

Assume the agreement is that 60% share of any cost difference is to the customer, and 40% share is to the supplier. Then:

target cost	target profit	target customer payment	actual supplier cost	adjustment	actual customer payment	actual supplier profit
1,000,000	150,000	1,150,000	1,100,000	+60,000	1,210,000	110,000
1,000,000	150,000	1,150,000	900,000	-60,000	1,090,000	190,000

If costs are higher than estimated, both the supplier and customer share the pain: The supplier's profit is lower and the customer assumes some of the burden of the cost. If there is a cost savings, both parties share the gain: The supplier's profit is higher and the customer pays less then the original target payment.

An implication of this is that both parties may—no guarantee—proactively promote ways to reduce waste during the project.

Payment scheme variations

Contract drafters have birthed myriad variations, including

- capped (ceiling) customer payment
- reduced supplier rate if target cost exceeded

Adjustable target cost and target profit

Another essential element of target-cost contracts, supportive of agile and iterative values, is the ability to adjust the target cost (and profit) by ongoing negotiation between parties. Keys for success in adjusting target cost include

- high transparency and near-real-time, open-book project accounting by the supplier so that the customer sees the true state of expenses within the supplier
- a spirit of working together by both the customer and supplier to continuously improve
  - this is something Toyota works hard at [ISV09]
- early agreement between the parties, expressed in the contract, on the guidelines for target-cost adjustment
- a *moderate* adjustment cycle, to avoid being overly reactive or using excessive overhead on adjustment; for example, once per iteration may be too frequent
- acceptance test-driven development, to reduce ambiguity

These practices reduce but do not eliminate contention during ongoing re-negotiations, because adjustment issues are often inherently fuzzy—variations of “Is that a defect or a feature?”

One group reported[21] that the following pre-agreed classification scheme for adjustments reduced contention:

Type	Description	May Adjust Target Cost?
fix	Changes to an implemented requirement, due to the supplier not doing what should have been ‘reasonably’ understood or done.	No

21. In [EMH05], a story about a target-cost contract for an agile project.

Type	Description	May Adjust Target Cost?
clarification	Changes to a ‘correctly’ implemented requirement, due to customer learning based on feedback.	No
enhancement	New feature.	Yes

We do not recommend this or any other scheme; it could help or lead to long and useless negotiation rather than cooperation.

Try…Multi-phase variable-model frameworks

Projects have a changing uncertainty or risk profile over time, ideally improving—for both parties. This can be reflected in *multi-phase* frameworks that reflect these shifting profiles for the customer and supplier. Any one phase can use any model: FPFS, progressive, target-cost, and so on.

One Valtech multi-phase variable-model example reflects the common Scrum pattern: (1) initial Product Backlog creation (2) adaptive iterative development. It was for a large B2B solution involving stakeholders in 23 countries:

1. *Phase 1—Fixed-price, fixed-duration, variable-scope*. Essentially, this was initial Product Backlog creation. The output of this phase was a Product Backlog—more precisely, a Release Backlog—and various business-analysis (market analysis, vision, …) documents, based on workshops and team analysis. Although a specific list of documents to deliver was identified in the contract, the scope of analysis or content varied.

   – with a Release Backlog, a cost estimate was possible, so…

2. *Phase 2—Progressive contract, T&M per iteration, release cap, non-binding Release Backlog, fixed duration*. Phase two was essentially a classic Scrum project: nothing in the original backlog was binding, but it served to estimate and bound release project cost, provide an initial overview, and kickstart what to do in the first iteration.

Why bother with these multi-phase models instead of simple progressive contracts? Typically, the motivator is (1) lack of trust, (2) a regulatory constraint, (3) a belief by a party they can make or save more money or better reduce their risk, (4) a need to define the vision, high-level requirements, or cost of a later phase (and the effort of this work itself is large), or (5) ‘optimizing’ on a secondary goal (other than project success) such as better cost predictability.

Another example [Larman03]:

1. *Phase one—Fixed-price partial-fixed-scope for three iterations (fixed duration)*. The "mandatory" fixed scope is low (for example, 20% of available effort), leaving plenty of room for variability and discovery when uncertainties are high. Risks to the customer and supplier are both bounded, and both parties learn a great deal about the nature of the continuing project for phase two.
2. *Phase two—FPFS, variable duration*. Risks to the supplier in taking on an FPFS phase are reduced because of increased knowledge and prior reduction of some sources of variability during phase one.

Conclusion

"How can we possibly do agile development when contracts are involved?" This is a question we have often been asked. But the key issues are not with the contract, they are with the contract writers and the clients they serve—reflecting the belief that success revolves more around contract negotiation and less around customer collaboration, or that *project success* is not the *goal* of contract work. That said, to reiterate a systems-thinking aphorism, "there is no blame," implying that the behavior of people within a system is shaped by it—in this case by encouragement of departmental silos, local targets (and rewards) leading to local optimizations, and the subtle message, "lawyers don't need to learn about operational details or new approaches to R&D, that's someone else's job."

Also, when this question is asked, 'contract' is often used as a synonym for "fixed-price fixed-scope" contract, which of course is not at all necessary—there are a wide variety of contract models, including variable-scope progressive contracts, and others.

Legal professionals have a duty to consider the ramifications of a breakdown of trust and collaboration—and other problems—when framing a contract. Just as contract lawyers need to learn more about lean and agile principles, other parties need to learn more about the necessary, valid concerns of lawyers. And just as product work improves with cross-functional *development* teams, further improvement is possible by including legal professionals in even broader cross-functional teams.

Recommended Readings

A first step is to read the suggestions listed in the section "Try…Lawyers study agile, iterative, & systems-thinking concepts" section on page 4.

There are several resources, most on the Web, related to agile development and contracts; however, some of it is speculative rather than experience-based—keep that in mind when reading. Possibilities:

? Mary and Tom Poppendieck, thought leaders in lean software development, have organized several contract workshops over the years, and collected and share "lean and agile contract" papers and presentations at their website www.poppendieck.com. Following a theme similar to this chapter, the Poppendieck's own material on agile contracts emphasizes the underlying issues of trust, collaboration, and transparency related to contracts.

? Susan Atkinson and Gabrielle Benefield have written about agile contracts under the appelation of the "evolutionary contract model"; see their article at www.infoq.com.

? At agilesoftwaredevelopment.com Peter Stevens has written an article summarizing "10 Contracts for your next Agile Software Project."

? Barry Boehm and colleagues have written several articles (available on the Web) on the "incremental commitment model".

? Some people new to the subject assume that contracts that encourage flexibility, collaboration, and alignment of interests ('agile' contracts) are a novel concept, but in fact much has been written and promoted in this area over the years, including within the USA government (for example, see *Administration of Government Contracts*). There are dozens, if not hundreds, of books and websites that discuss a variety of contract models.

? There are several 'public' contract models that we have reviewed, explicitly supporting iterative, evolutionary, or agile development. However, Valtech and ThoughtWorks—and other agile outsourcers that we know of—write their own contracts rather than use these models. We discourage "copy-paste" contract writing, but these are food for thought:

  – The DSDM consortium (DSDM is an agile method) offers a sample contract, available to members at www.dsdm.org. Note that the contract is occasionally revised.

  – The Norwegian PS 2000 Contract was created for iterative and evolutionary development, by an alliance between industry and government, available at www.dataforeningen.no. To quote, "[*The] Contract is designed to be used when it is particularly difficult or unserviceable to draw up a detailed specification prior to tendering, the idea being to leave open for the developer to find the best way to attain the objectives and needs of the customer.*"

REFERENCES

[Austin96] Austin, R., 1996. *Measuring and Managing Performance in Organizations*, Dorset House

[AV07] APLN, Version One, 2007. "2nd Annual Survey. The State of Agile," *VersionOne website* at http://www.versionone.com/pdf/stateofagiledevelopment2_fulldatareport.pdf

[BC99] Beck, K., Cleal, D., 1999. “Optional Scope Contracts,” at www.jarn.com/about/OptionalScopeContracts.pdf

[CSSD05] Ceschi, M., Sillitti, A., Succi, G., De Panfilis, S., 2005. “Project Management in Plan-Based and Agile Companies,” *IEEE Software*, May/June 2005

[DBT05] Dalcher, D., Benediktsson, O., Thorbergsson, H., 2005. “Development Life Cycle Management: A Multiproject Experiment.” *Proceedings of the 12th International Conference on Engineering of Computer-Based Systems*, IEEE Computer Society

[EMH05] Eckfeldt, B., Madden, R., Horowitz, J., 2005. “Selling Agile: Target-Cost Contracts.” *Proceedings of Agile 2005 Conference*

[Gilb05] Gilb, T., 2005. *Competitive Engineering*, Butterworth-Heinemann

[Herzberg87] Herzberg, F., 1987. “One More Time: How Do You Motivate Employees?” *Harvard Business Review*, Sept/Oct 1987

[ISV09] Iyer, A., Seshadri, S., Vasher, R., 2009. *Toyota’s Supply Chain Management: A Strategic Approach to Toyota’s Renowned System*, McGraw-Hill

[Kohn93] Kohn, A., 1993. *Punished by Rewards*, Houghton Mifflin

[Larman03] Larman, C., 2003. *Agile and Iterative Development: A Manager's Guide*, Addison-Wesley

[LV08] Larman, C., Vodde, B., 2008. *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*, Addison-Wesley

[MacCormack01] MacCormack, A., 2001. “Product-Development Practices That Work,” MIT Sloan Management Review. Vol. 42, No. 2.

[Martin04] Martin, R., 2004. “Estimating Costs Up Front,” *Extreme Programming mailing list*, at http://groups.google.com/group/comp.software.extreme-programming/msg/9a203fad85f3d363?hl=en

[MJ05] Moløkken-Østvold, K., Jørgensen, M., 2005. “A Comparison of Software Project Overruns—Flexible versus Sequential Development Models,” *IEEE Transactions on Software Engineering*, Vol. 31, No. 9, Sept 2005

[Moore91] Moore, G., 1991. *Crossing the Chasm*, HarperCollins Publishers

[Parkinson57] Parkinson, C., 1957. *Parkinson’s Law*, Buccaneer Books

[Poppendieck05] Poppendieck, M., 2005. “Agile Contracts” Agile 2005 Conference Workshop, at www.poppendieck.com/pdfs/AgileContracts.pdf

[PRL07] Parsons, D., Ryu, H., Lal, R., 2007. “The Impact of Methods and Techniques on Outcomes from Agile Software Development Projects,” *IFIP—Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*. Springer (draft)

[PS06] Pfeffer, J., Sutton, R., 2006. *Hard Facts, Dangerous Half-Truths And Total Nonsense*, Harvard Business School Press

[Reifer02] Reifer, D., 2002. “How Good are Agile Methods?” *IEEE Software*, July/Aug 2002.

[Smith07] Smith, P., 2007. *Flexible Product Development: Building Agility for Changing Markets*, Jossey-Bass