
Contracts

Can There Be Trust Between Firms?

“Agile development sounds good, but how does it apply to me? I have to work under contract.” Without doubt, the biggest barrier to using agile practices is the sharp line between one firm and another. Each firm is expected to look out for its own interests, with the understanding that the other firm will be doing the same thing. So it would seem that the only safe approach is to write an airtight contract, because people move to new jobs, rules change, and then the only thing that matters is what’s in the contract.

Actually, there is a better way, one that was pioneered by Toyota when it started working with US suppliers in 1988, and documented by Jeffrey Dyer in *Collaborative Advantage*.ⁱ Of course, Toyota negotiated contracts with its suppliers, but the contracts were not the primary vehicles that protected the suppliers’ interests. In a surprisingly short time, suppliers developed *trust* in Toyota, and in 1998, Toyota was rated by auto suppliers as the most *trusted* automaker in the country, scoring twice as high as General Motors.ⁱⁱ *Trust* in this case has a specific meaning:

- ✓ The extent to which automaker can be trusted to treat a supplier fairly
- ✓ The extent to which automaker might try to take unfair advantage of the supplier
- ✓ The automaker’s reputation for fairness among the supplier community

This kind of trust does not come from individuals trusting each other. Suppliers may trust an individual purchasing agent completely. But they can’t trust that the same person will be there a year later, or that whoever is there will be still playing by the same set of rules. Suppliers developed “a greater trust in the fairness, stability, and predictability of Toyota’s routines and processes”.

Dyer notes that suppliers share proprietary information with Toyota, confident that it will not find its way to their competitors, as sometimes happened with General Motors. Suppliers invest in specialized equipment for Toyota, knowing that Toyota does repeat business with its suppliers 92% of the time, while they had only a 52% chance of repeat business from GM. Suppliers let Toyota experts into their plants to teach them the Toyota Production System, understanding that Toyota will not demand price reductions based on their findings, as GM has been known to do.ⁱⁱⁱ

It’s not that Toyota doesn’t look out for its own best interests; it’s just that Toyota understands that a strong supplier network is far more beneficial to its interests than short-term gains that come from taking advantage of a supplier. Toyota, in the US, obtains about three quarters of its components from suppliers, while US automakers obtain less than half of their components from suppliers. Yet Toyota spends half as much money and half as much time on procurement as GM. In addition, suppliers are more productive and produce better quality in manufacturing cells devoted to Toyota.^{iv} As an organization, Toyota is keenly aware that partnership relationships rather than arm’s-length relationships with the bulk of its suppliers better serves its best interests.

Another company that achieves high value from partnerships with suppliers is Dell, which thus far has outperformed and outlasted most of its competitors in the highly competitive market of selling

personal computers. To do this, Michael Dell focused company efforts on understanding the perception of value in high margin customer segments and then delivering that value as rapidly as possible. This means the company does not focus on making hardware or software; that would be a distraction. Instead, Dell has worked to establish sophisticated win-win arrangements with its suppliers, which operate to the mutual benefit of both parties.^v

But Software is Different...

You might be saying to yourself, good supplier relationships are important when the supplier is developing something they can manufacture many times over, like a disk drive or a taillight. But in software, we develop a system only once; it is complex and expensive; it is subject to many changes; and if not done right, the financial impact can be tremendous. Where is the parallel to *this* in manufacturing?

At the beginning of Chapter 3, we discussed the large and expensive metal dies used to stamp out vehicle body panels. The cost of these dies accounts for close to half of a new model's capital investment. They are complex, expensive, and subject to many changes, even after the design is supposedly frozen. Correcting a mistake made in cutting a die is very time-consuming and it's expensive to start over again. Yet in the late 1980's Toyota developed dies for half the cost and in as little as half the time using concurrent development practices, compared to the typical U.S. company using sequential development. Moreover, the resulting dies gave Toyota a 30% cost advantage in the manufacturing process.^{vi}

Tool and die makers are supplier companies in both the US and Japan. US automakers waited until the design specs were frozen, and then sent the final design to the die cutting supplier, which triggered the process of ordering the block of steel and cutting it. Changes had to be approved and officially sent to the supplier by the purchasing department. Since suppliers had to bid low to get the job, they made most of their profits from the change orders, which amounted to 30 – 50% of the die cost.^{vii}

In Japan, the tool and die suppliers start working on a die at the same time the car design is started. Die cutters are expected to know what a die for a part will involve, and they are in constant communication with the designer. Suppose that a body engineer wants a change made. The body engineer goes directly to the die cutting shop, discusses the proposed change with the die engineers, checks production feasibility, and together they decide what to do. The die shop makes the changes in the milling machine and keeps on cutting the die. Paperwork and approvals follow later.^{viii}

In Toyota, tool and die contracts are *target cost* contracts; the supplier and automaker agree on the total target cost of the tools, including all changes. Typically, changes add 10–20% to the base cost, and this is covered by in the original contract. If the target cost cannot be met, the parties negotiate who is to bear the added cost, and generally, Toyota ends up with the larger share. This kind of arrangement gives the engineers in *both* companies incentives to work together to keep the cost within target.

In the US, toolmakers had fixed price contracts that went to the lowest bidder, so they viewed engineering changes as profit making opportunities.^{ix} To contain costs, automakers put a rigorous change

approval process in place, similar to the change approval processes found in many software development contracts. When you look at the overall result, the US approach almost doubled the cost and time necessary to make a die.^x Moreover, it resulted in a lower quality die.¹

We believe that the overall impact of many contracting and scope control policies in software development is in the same ballpark. That is, a fixed price contract with a vendor hoping to profit from changes, combined with rigorous change approval mechanisms to contain cost, may approximately double the cost and time it takes to develop the software, while producing a lower quality result.

The Purpose of Contracts

Dyer defines *trust* as “one party’s confidence that the other party ... will fulfill its promises and will not exploit its vulnerabilities”.^{xi} Many people think that the reason for contracts is to substitute for this trust. Conventional wisdom says that all eventualities should be spelled out in a contract, so the parties cannot possibly take advantage of each other.

Many enterprises find it almost impossible to select suppliers using a process that values good faith or write contracts that assume that other party will act in good faith. It is widely held that the purpose of contracts is to limit the natural tendency of one party to take advantage of the other party as it looks out for its own interests.^{xii} However, if damaging behavior can be limited through the relationship rather than the contract, all manner of benefits in terms of speed, flexibility, cost, and information exchange can result. Unfortunately, these benefits are counterintuitive and difficult for a public official to explain in a to a newspaper reporter.

Let’s take a step back and examine why companies work with suppliers in the first place. As our world gets more complex, there is a great value in specializing. If you were going to have a rare kind of surgery, you want to go to a hospital that specializes in it. If Dell wants the best video display card, it collaborates with the company that makes that card. If you want the best software for a particular area, you are likely to seek out the companies that are experts in providing that kind of software.

Another reason to outsource software development is to reduce costs and improve the likelihood of success. For example, an organization might find that salaries at a vendor are lower than their own salaries. Or it might negotiate a fixed price for a system that is lower than the internal cost to do the same work. They may find that an experienced software development vendor might have skills sets that are not available internally.

Let’s examine the cost side of the equation. Money actually paid to vendors is only part of the story. In addition, there are transaction costs – the cost of selecting potential vendors, negotiating and renegotiating agreements, monitoring and enforcing the agreement, billing and tracking payments. As we demonstrated in the die cutting example in the previous section, the cost of trying to control changes can add huge hidden costs to a contract, and you can expect such costs to escalate in an evolving domain.

¹ Due to superior die quality, typical Japanese stamping in 1990 took 5 shots per panel, compared to 7 in the US, saving manufacturing time. Clark, *Product Development Performance* (1991) page 186

Dyer finds that the second kind of costs, transaction costs, dominate most vendor-supplier relationships.^{xiii} So when evaluating the cost of outsourcing, it is imperative that all costs are considered, direct costs, obvious transaction costs, and hidden costs that come from arm's length relationships and change intolerance. These costs will be especially high in an environment that is going to change despite heroic efforts to keep change at bay.

Let us turn our attention to the third cost of outsourcing, the lost opportunity cost that may result if the communication bandwidth between customer and vendor is narrow. As we saw in Chapter 6, system integrity depends on broad, early, and frequent communication between customer and developer. Lack of communication between customer and vendor is a frequent cause of system failure.^{xiv} Bear this in mind if increasing the chance of success is a reason for outsourcing.

Contracts that focus on keeping parties from taking advantage of each other have a lot of built in control mechanisms and communication gates that have a tendency to raise costs and reduce the collaboration critical to success. Contracts that focus on supporting collaboration are more likely to reduce costs and result in successful contracts.

Fixed Price Contracts

Let's examine the most commonly used contract designed to protect the customer, the fixed-price contract. Sometimes corporate budgeting cycles and related processes require fixed price contracts. For many government entities, law requires fixed price contracts – often awarded to the lowest bidder. As we saw in the die-cutting example, this practice encourages vendors to bid low and make their profit on changes. Another motivator for fixed price contracts is the desire of a customer to transfer risk to the vendor. In practice, the customer can't really transfer the bulk of the risk. If the contract doesn't work out, the customer will suffer.

As we noted in Chapter 2, it is a good idea to develop software in short iterations driven by immediate customer needs, developing high priority features first and stopping when resources run out. However, this approach is very risky for vendors working under fixed price contracts, because they frequently have difficulty obtaining customer agreement that the work is done when the money runs out. Therefore, vendors tend to protect themselves by creating a detailed specification and keeping it under strict change control, charging extra for any changes. The result may be a substantial increase in cost or a very disappointed customer.

Fixed Price – Unhappy Customers

“I ran a software development company which prided itself in not exceeding the price and schedule we quoted at the beginning of an engagement. In a three-year period, we had 78 projects, and 77 of them were delivered on time, on budget, and in scope. Then I surveyed the customers and found out that none of them was happy! The systems that we delivered did not solve their problems. Sure, we had protected ourselves by being on time, on budget, and in scope, but in doing this, we could not deliver what the customers really wanted. That's why I sold my business.”

– *A colleague (who wishes to remain anonymous)*

Risk should be born by the party best able to manage it, and in a fixed price contract, risk is seemingly transferred to the vendor. If a problem is technically complex, then the vendor is most likely to be in a position to manage the associated risk, so it is appropriate for the vendor to assume the risk. However, if a problem is uncertain or changing, then the customer is in the best position to manage the risk, so fixed price contracts should be avoided. If a fixed price contract cannot be avoided, then the customer should be willing to incur a substantial cost beyond the fixed price, due to the certainty of changes.

Fixed price contracts may involve significant risk in estimating the cost prior to doing any work. A competent vendor will include this risk in the bid. A vendor that does not understand the complexity of the problem is likely to underbid. The process of selecting a vendor for a fixed-price contract has a tendency to favor the most optimistic – or the most desperate – vendor.^{xv} Consequently, the vendor least likely to understand the project's complexity is likely to be selected. Thus fixed price contracts tend to select the vendor most likely to get in trouble.

Therefore, it is quite common for the customer to find a vendor unable to deliver on a fixed price contract. By the time this becomes apparent, the customer rarely has the option to choose another vendor, so the customer must often come to the rescue. Alternately, the vendor may attempt to recoup their loss through change orders, which leads the customer to avoid aggressively any change to the contract. Faced with no other way to recover a loss, a vendor will be motivated to find ways to deliver less than the customer really wants.

A fixed price contract is biased in favor of the customer at the expense of the vendor, making it necessary for vendors to protect their interests aggressively, at the expense of the customer. It is not a climate in which organizational trust has much soil to grow.

Time-and-Materials Contracts

“Customers should prefer flexible-price contracts to fixed-price contracts where it is cheaper for the customer to deal with uncertainty than it is for the contractor to do so or where the customer is more concerned with the ability of the contractor to provide a product that works than with price,” writes Fred Thompson in the *Handbook of Public Administration. (Second Edition)*^{xvi}

The flexible-price contract, also known as a time-and-materials or time-and-expenses contract, is designed to deal with uncertainty and complexity, but it does not do away with risk, it simply shifts it from the vendor to the customer. In the 1970's, the U.S. Department of Defense (DoD) experienced some very high profile bailouts on fixed price contracts, so it began to use more time-and-materials contracts in situations where the government was better able to manage the risk.

On the downside from a vendor perspective, time-and-materials contracts offer less security than fixed price contracts. However, these contracts are usually considered a good deal for vendors for as long as they last. In fact, vendors generally have little incentive to be efficient, because the longer the work takes, the more money they make. To control self-serving behavior on the part of time-and-materials vendors, DoD developed extensive vendor control mechanisms, which contributed to the development of the discipline of project management.

Time-and-materials contracts mark a significant increase in contract transaction costs. Companies with DoD contracts not only hire administrators to oversee compliance with contract requirements, they also add accountants to sort out allowable and unallowable costs. High transaction costs would be reasonable if they added value, but in fact, transaction costs are by definition non-value-adding costs. Thompson notes, “Controls contribute nothing of positive value; their singular purpose lies in helping us to avoid waste. To the extent that they do what they are supposed to do, they can generate substantial savings. But it must be recognized that controls are themselves very costly.”^{xvii}

One way to avoid the high cost of controls is not to use them. Thompson^{xviii} suggests that when the costs of controls are high, it might be better to keep work inside a vertical organization, where presumably administration will control self-serving behavior. Unfortunately, vertical integration does not always work to minimize control costs. In fact, many organizations find themselves using DoD-style project management controls internally. It seems incongruous that cost, schedule and scope control mechanisms, that add cost but not value, and that were invented to prevent contractual parties from taking advantage of each other, would come to dominate development inside of companies – the very place where they should not be needed.

Time-and-material contracts can be used for agile software development as long as the contract allows for concurrent development and collaboration between the parties. The first step is to change the control mechanism from one that favors sequential development to one that favors concurrent development. After establishing a conceptual design and the overall capability of the system, sketch out a tentative release plan and begin iterations as soon as possible, so the customer can see working code and offer concrete, timely feedback. As velocity becomes established, modify the release plan and level of resources if necessary.

The problem with time-and-material contracts is that once the system is partially deployed, the customer is dependent on the vendor, while the vendor has limited incentive to reduce costs. Agile development mitigates this bias in favor of the vendor by having the vendor deliver value for the money spent at the end of every iteration. Each iteration, the customer schedules the most valuable remaining features, insists on delivery of working, integrated code, and evaluates the value delivered. This gives the customer the option to terminate the contract at any point and still obtain value for their investment up to that time.

When you think about it, concurrent development is a safer approach for time-and-materials contracts than sequential development and its associated controls. Exchanging incremental value for incremental pay protects both vendor and customer. However, this approach requires that the project management systems commonly used for sequential development be set aside. More importantly, there must be on-going collaboration between working-level people in the vendor and customer shops.

Multi-Stage Contracts

Multi-stage contracts attempt to deal with the unknowns and risks inherent in fixed price contracts, matching the risks to the dollars spent over time. There are two types of multi-stage contracts: those intended to lead to a large fixed price contract, and those that retain their multi-stage character throughout.

Multi-stage contracts that morph into large fixed price contracts start with one or two short contracts for learning enough about the problem to enable a fixed price bid on the overall system. Usually only one vendor is involved, so this kind of contract is not generally appropriate when bidding is required.² Assuming the vendor remains the same throughout, the customer and vendor increase their learning, reducing the risk of big surprises on either side. However, the incentive to freeze the specification and not allow changes in the final stage is, if anything, higher. There will be less sympathy for a change in the specification if the vendor was paid to get it right in the early stages. Thus, this type of a multi-stage contract retains the problems of a fixed price contract if uncertainty or change is involved after the body of the contract is awarded.

The second type of multi-stage contract, which retains its multi-stage character throughout development, presents a good opportunity for agile development, because it is easy to adapt to iterative development. However, these contracts are not without risks, the biggest risk being the fact that each party has frequent opportunities to abandon the relationship. Multi-stage contracts create what might be called a bilateral monopoly,^{xix} that is, both sides come to depend on each other. If one party ends their involvement, the other party may have a lot to lose.

One way to mitigate the risk posed by the bilateral monopoly in multi-stage contracts is to deliver value with each increment in proportion to the money spent. As in time-and-material contracts, it is a good idea to implement the highest priority customer features first and deliver working, integrated code with each iteration.

Another way to mitigate the risk of termination in a multi-stage contract is to address the risk through the relationship – that is, the parties develop a trust that the relationship will continue as long as expected value is delivered. In *Agile Software Development Ecosystems*,^{xx} Jim Highsmith discusses delivered-feature contracts. These are fixed-schedule, variable-scope contracts in which the customer evaluates the value delivered after each iteration. If the work is acceptable, the contract continues into the next iteration. Although there is no contractual obligation for both parties to continue working together, their trust in each other builds at the same rate that their dependence upon each other deepens.

Multi-stage contracts will rapidly get expensive if a contract must be negotiated for each stage. Thus, these contracts are usually governed by a master contract negotiated at an early stage, with work-orders executed against the master contract for each iteration.^{xxi}

Tailoring Multi-Stage Contracts to the Domain

Tim works in a company that sells cutting edge software to large companies. The software supports innovative hardware that is constantly evolving. In such a changing environment, you would think that development would be kept internal, but such is not the case. Small,

² Sometimes, multi-staged fixed price contracts are set up so that one vendor is selected to write a specification, and that specification is let out for bid. In this case, there is a decision to make – is the vendor who wrote the specification allowed to bid? Behind the question lies the assumption that this vendor has obtained superior knowledge that is not found in the specification. Of course, this is the case, because a great deal of domain knowledge is tacit knowledge that cannot be transferred in writing. If the vendor who wrote the specification is not allowed to bid, then all of their tacit knowledge has been wasted, and to the vendors allowed to bid on the contract, this is no different from a single fixed price contract.

venture-funded startup companies develop new techniques faster than Tim's company can internally. It is Tim's job to contract with these small companies to develop portions of the software his company will sell.

One of the big issues in contract negotiations is ownership of intellectual property, which tends to make contract negotiations arduous and not something you would want to do every few weeks. Yet the technology is changing so fast that Tim's company doesn't know exactly what they want a supplier to do beyond the next three or four months. Further, the small companies are eager to have extended contracts to show their venture funders.

So the first principle Tim employs is to split the risk by splitting development into two contracts. The first contract is for proof of concept, and the second is used to finalize the product. Prior to negotiating the first contract, there would be a short (2-3 week) collaboration period between both parties, using a time and materials contract, to establish an overall plan. This makes the first contract easier to negotiate.

Since the first contract is for a proof of principle or 'rough draft,' the supplier has full responsibility for the system from architecture to implementation. This is a fixed price contract, and the supplier is usually expected to work for cost plus a small profit, with the assumption that success at this stage would lead to future profits. This contract can be canceled if it is not proceeding satisfactorily.

Assuming the first contract goes well, Tim's company has learned enough about the vendor's work to commit to buying a minimum number days on a time and materials basis. Of course, given the rapidly changing technology, they do not attempt to specify exactly what is to be done, but generally, they find that they can keep their vendors usefully occupied at the guaranteed level of commitment. During this second contract, Tim's company takes over more responsibility in guiding development, since the objective is to deliver a quality product without defects.

Once the software is released, the vendor is expected to provide a warranty, for example three months of free defect fixes, to ensure they deliver high quality by delivery date. For the warranty, Tim prefers a fixed price support fee with a guaranteed service level. Tim is careful separate out warranty requests from upgrade requests, even when the vendor is contracted with separately to provide upgrades.

– Based on conversations and e-mail with Tim Ocock

Target Cost Contracts

The problem with traditional fixed price contracts is that they encourage self-serving behavior on the part of the customer and defensive behavior on the part of the vendor. The problem with traditional time-and-materials contracts is exactly the opposite: they encourage self-serving behavior on the part of vendor and defensive, control-oriented behavior on the part of customers. What we need is a middle ground, one in which risk is shared and both parties have incentives to look out for the overall interests of the joint effort.

There are no canned answers to the contract dilemma, because in the end, no contract can fully prevent parties from taking advantage of each other. Contracts do not create confidence that the other party will honor its commitments and not exploit vulnerabilities (Dyer's definition of 'trust'). There are, however, contract forms that make it easier for parties to share in the problems and rewards brought about by their relationship. One example is a target cost contract. While the target cost contract is not a panacea, it is at least a platform on which a partnership can be built.

Target cost contracts are structured so that the total cost – including changes – is the joint responsibility of the customer and vendor. What makes a target cost contract different from a fixed price contract is that if target cost is exceeded, both parties will end up paying more, and if total cost is under the target cost, both parties will share in the benefits. What makes a target cost contract different from a time-and-materials contract is that vendors do not gain added profit if they work longer, but they may receive a benefit if they are under cost or schedule.

In a target cost software development contract, the parties start with a general agreement of what is to be accomplished, recognizing that the details cannot be known until mutual work is done. They then come to an agreement of the target cost for the system, and agree upon a schedule. In this type of contract, the target cost is understood to be very important, so the design and detailed features will be focused on meeting the target cost. There is a commitment on the part of both parties to meet target cost and this is understood to require a joint effort of both the technical people and users on both sides.

A target cost contract recognizes that the actual costs will not necessarily be the same as the target costs, so it provides for a fair allocation of any costs over the target costs, or a fair sharing of any benefits if costs are below target costs. These contracts must give the customer an incentive to keep their demands for features in line with target costs, while giving the vendor incentives that favor completing the work under the target cost. Usually the customer incentive is provided for by a clause triggering equitable cost-sharing negotiations should the actual cost vary significantly from the target cost. One of the following usually provides for the vendor incentive:

- ✓ **Cost-plus-fixed-fee:** The target cost does not include profit for the vendor; a separate fee is included to provide vendor profit. The fee is generally paid after the work is successfully completed. If total cost exceeds target cost, the vendor works at cost for the remainder of the contract. If total cost is lower than target cost, the vendor receives a higher profit margin. A bonus for coming in below target cost may be included.
- ✓ **Profit not to exceed:** The target cost includes the vendor profit. The vendor agrees to reduce rates and exclude profit after the target cost is reached. If total cost exceeds target cost, the vendor works at cost. However, in this case the vendor has no incentive to come in under target cost, unless there is a bonus for early completion.

The most valuable part of target cost contracts is that they more accurately communicate management intent to the front line workers of both parties, and encourage them to work together to achieve this intent. If cost expectations are not made clear to the working teams from the beginning, the resulting design is unlikely to meet the target. Target cost contracts must leave the details of the

scope to the discretion of the technical teams, because reducing scope is the most fertile ground for cost control.

Target Cost Contract Example

The customer had a fixed budget for the project, and that was not going to change. They wanted two data entry applications moved to a Windows environment, plus a web front end developed so that their customers might enter some of their own data. The legacy database needed to be modified to support current practice or converted to a new database system.

The problem was initially divided into four components: two applications, the database, and the web interface. A team was formed for each component and given a budget expressed in terms of staff-days. Team membership included the customer manager responsible for the area, a master developer, an analyst/ tester, and an operations/help desk representative. Each application team got 35% of the budget, the database team got 15% of the budget, and the web team got 10% of the budget. 5% was held in reserve for contingencies.

Each team was chartered to figure out how to develop and deploy their portion of the system within their budget. As teams developed a preliminary release plan, they started making tradeoffs immediately to keep within their staff-day budgets. The application teams realized that their jobs would be a lot easier if the database were converted rather than wrapped, but the database team did not have enough staff-hours in its budget. The DBA convinced the application teams that they would be better off with a lower budget and more sophisticated database support, so each application team gave the database team a portion of their budget.

With that decided, the teams got to work on iterations, with highest priority items first. The database team was particularly devoted and quickly began populating two new development databases with sample legacy data, one for each application. They would merge the two databases later.

The application and web teams had a useful database starting with the first iteration, so they had a reasonably good rendition of the main data entry screen at the end of the first iteration. Each iteration resulted in working, tested software, but all teams decided to delay moving the system into production until a more complete system was available. There was no good way to integrate the old and new systems, so going into production would require more or less complete functionality. However, it was agreed that the applications could go live independently, and that the web front end could follow either application.

As time went on, the applications and web teams discovered they all needed the same financial features, so they agreed to pool some of their staff-days and charter a sub-team to develop the joint financial functions.

As the budgets approached 50% depleted, the teams took a close look at their velocity and got a good picture of how they were doing on their staff-day targets. They did some hard thinking about what they really needed. At this point, it was especially important for the customer managers to feel obligated to negotiate. If this had been a fixed price contract, they probably would not have felt the need to dig deeply to find features they could do without. However,

the customer managers felt responsible for meeting their team's staff-day targets, and being managers, they were used to that. So they were quite aggressive in discarding features.

With 70% of its budget used up, one application team decided it was ready to go live and spend the remainder of its budget after startup. They found that deployment was more difficult than expected, especially because of the new financial features. But after all the problems were resolved, they still had 10% of their budget left to deal with issues uncovered by production. At about the same time, the web team went through an easy startup, which was lucky because they were almost out of hours.

The remaining application team had a challenging problem with the legacy database, but on the bright side, the other application team had gotten the financial system working. They burned up 90% of their budget before they went live, and needed some of the 5% contingency to complete deployment. This left them with scant funds to do any improvements after production started, so it seemed like they would have to wait until the next budgeting cycle and get a special allocation.

Fortunately, the local maintenance programmers had been involved in the effort, and they were ready to take over more responsibility. Their time was not charged to the project, so they could work on the system without jeopardizing the budget. They were able to add critically needed features with some guidance from the original developers, and in the process, they became confident of their ability to support the system.

– *A Business Novelette*

Target Schedule Contracts

Sometimes schedule is more important than cost, although cost is rarely unimportant. If the number of people working on the system does not change and no components are purchased or licensed, then target cost and target schedule are the same thing.^{xxiii} Software product companies often meet hard schedules for product upgrades by fixing the resources and the schedule, and working on the highest priority items first. When time runs out, the low priority features are left undone, but the release meets the overall intent of product marketing.

In the same way, a target cost contract can usually be run as a target schedule contract by fixing the resources and schedule. Features should be addressed in priority order, and each iteration should deliver working, tested, integrated, deployable software. Well before the deadline, the software should actually be deployed. Then iterations can continue to deal with issues that arise in production. With this approach, the completed work will be on schedule and on budget by definition, and the delivered features should meet the overall intent of the contract.

If schedule really is the only thing that is important, then a target schedule contract is more appropriate than a target cost contract. This allows the team to add resources or license components as needed to meet the schedule. The more degrees of freedom that a target contract leaves to the workers, the easier it will be for them to figure out how to meet the target.

Shared Benefit Contracts

Target cost and target schedule contracts set up an environment in which teams work effectively across company boundaries because it is clear that both companies will share the risks and rewards of the work. This is the key to collaborative contracts; the people doing the work must perceive that both parties have a stake in the results of their efforts. A *profit-sharing contract* is another effective mechanism for sharing risks and rewards if you are developing products for sale. Tim Ocock's company (see sidebar, multi-stage contracts) frequently uses profit sharing contracts.

In a *co-source contract*, both companies share responsibility for developing a system, and the vendor is also expected transferring their expertise to the customer. A co-source contract is successful if the vendor works itself out of a job by helping the customers develop the capability to do the work themselves. Co-sourcing is a fundamentally collaborative approach, so co-source contracts do not tend to create motivation for self-serving behavior. Bruce Ferguson's company (see sidebar, below) prefers to use a co-source arrangement whenever possible.

Agile Contracts Make Business Sense

Bruce is the VP of sales in a company that prides itself on using agile practices to develop systems in large companies. Their preferred approach is to co-source the work – that is, half of the work will be done by Bruce's company, and the other half by people in the client company. In this case, Bruce's company quite often does not manage the project, although they have a project leader for their team who works closely with the client project manager.

Bruce finds that no two situations are the same, so one needs to take an agile approach to establishing a contract for software development. The first thing to determine is whether the client is sold on an agile approach, whether they can be sold, or at least, can they trust an agile approach. Bruce works from three levels of estimates: ballpark, budgetary, and bull's-eye. He notes that everyone starts with a ballpark estimate. It's when you get to the budgetary estimate that you switch to an agile focus and convince people that they will get more for their money if they do not attempt to define all of the functionality and do all of the planning up front. Bruce finds that 60-70% of the time he can sell an agile approach.

Bruce notes that agile must be sold at a high enough level to influence procurement practices, so it is important that the person agreeing to try an agile approach is willing to champion the agile approach to his or her management. If people back down from an agile approach when they encounter difficulties, then they haven't really been sold on the approach.

Bruce tries to avoid tying pricing to deliverables; if pricing is tied to a deliverable, it must be a very small chunk of work. This is the essence of an agile approach, and it often runs counter to the procurement practices of the client. However, if a high enough person in the client company has agreed to use an agile approach, then these procurement and legal issues will be addressed by that champion.

If the client has gotten to the point of agreeing to an agile approach at a level sufficiently high to precipitate a change in procurement practice, then Bruce can rely on the client to put together an appropriate contract. The secret is not in the contract wording itself, but in having

a sponsor at a high enough level who understands that they can get more if they allow the system to evolve rather than be specified in detail at the beginning.

Bruce has found that once a client has experienced an agile project, the nature of the contract is not much of an issue for subsequent agile projects. Results talk!

– *Based on conversations and e-mail with Bruce Ferguson*

The Key: Optional Scope^{xxiii}

We have noted several types of contracts that can work for agile software development:

- ✓ Time-and-material contract using concurrent development with highest priority features implemented first, and working, integrated code delivered at each iteration, so that the customer may easily manage cost by limiting scope
- ✓ Multi-stage contracts using a master contract and work orders to release each iteration, with similar emphasis on concurrent development, highest priority features first, and working, integrated code delivered at each iteration.
- ✓ Target-cost contracts which charter the front line workers of both parties to work together to come up with a solution to the problem that meets a target cost, giving them the freedom to limit scope as a primary mechanism to achieve the target cost
- ✓ Shared Benefit contracts that assume the parties will modify what they are doing as time goes on to achieve mutual benefit.

There is a common theme here: all of these contracts are mechanisms that avoid fixing scope in detail. This should not come as a surprise. Jim Johnson of the Standish group noted 64% of the features in a typical system are rarely or never used, suggesting that the most fertile ground for productivity improvement in software development lies in not implementing features that are not needed.^{xxiv} As Boehm noted in 1988,^{xxv} the best way to develop low cost, high quality software is to *write less code*. Chartering a software development team to accomplish a purpose within cost and schedule constraints is about the same as asking them to figure out which features to leave out of the system.

Conventional wisdom holds that specifying and controlling scope in a contract is necessary to protect an organization from self-serving behavior on the part of the other party. However, the effect of this protection is a sub-optimized value stream. Although it seems counterintuitive, rigid control of scope tends to expand, not reduce the scope. This in turn leads to a significant increase in the cost of the features as well as the cost of the control system. The bottom line? Organizations that use outsourcing as a way to save money will save more money overall if they collaborate with vendors by using some form of optional scope contract.

Establishing a partnership relationship with vendors generally happens at the initiation of the customer, and it is not as simple as using any specific form of contract. Both parties need a clear understanding of the value they could bring to each other if they focus on mutual benefit instead of indi-

vidual benefit. Partnerships require consistent practices so partners develop confidence that commitments will be honored and vulnerabilities will not be exploited, even if individuals change. This in turn requires creative agreements that do not try to cover every eventuality, but instead provide ways to deal with unpredictable future events in a manner that both sides will perceive as fair and equitable.

ⁱ Dyer, *Collaborative Advantage* (2000)

ⁱⁱ Dyer, *Collaborative Advantage* (2000) page 90

ⁱⁱⁱ See Dyer, *Collaborative Advantage* (2000) pages 94, 97, 101-103

^{iv} Dyer, *Collaborative Advantage* (2000) pages 5-7

^v See Magretta “The Power of Integration; An Interview With Michael Dell” (1998)

^{vi} See Clark, *Product Development Performance* (1991) pages 187 and 234-237, and Womack, *The Machine That Changed the World* (1991) page 111.

^{vii} Clark, *Product Development Performance* (1991) page 187

^{viii} Clark, *Product Development Performance* (1991) pages 236-237

^{ix} Clark, *Product Development Performance* (1991) page 187

^x See Clark, *Product Development Performance* (1991) pages 187 and 234-237, and Womack, *The Machine That Changed the World* (1991) page 111.

^{xi} Dyer, *Collaborative Advantage* (2000) page 88

^{xii} Thompson, *Handbook of Public Administration* (1998)

^{xiii} Dyer, *Collaborative Advantage* (2000) pages 91-96

^{xiv} See Ripin, *Insider Strategies for Outsourcing Information Systems* (1999) pages 43, 58-59

^{xv} Thompson, *Handbook of Public Administration* (1998)

^{xvi} Thompson, *Handbook of Public Administration* (1998)

^{xvii} Thompson, *Handbook of Public Administration* (1998)

^{xviii} Thompson, *Handbook of Public Administration* (1998)

^{xix} See Thompson, *Handbook of Public Administration* (1998)

^{xx} Highsmith, *Agile Software Development Ecosystems* (2002) pages 74-75

^{xxi} See Pitette, “Progressive Acquisition and the RUP” (2002), and Wideman, “Progressive Acquisition and the RUP” (2002), (2003)

^{xxii} If components will be licensed or purchased, see Hohmann *Beyond Software Architecture* (2003).

^{xxiii} See Beck, “Optional Scope Contracts” (1999)

^{xxiv} Johnson, Keynote, Third International Conference on Extreme Programming (2002)

^{xxv} Boehm, ‘Understanding and Controlling Software Costs’ (1988)