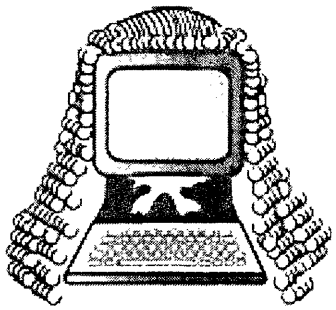

COMPUTERS & LAW

Journal for the Australian and New Zealand Societies for Computers and the Law

Editors: Claire Elix, Melissa Lessi and Laura Seeto

ISSN 08117225

Number: 58

December 2004

Managing Software Development Agreements: A Practical Guide*

Alec Christie, Middletons Lawyers

Alec Christie is a partner of Middletons Lawyers specialising in Information Technology and Intellectual Property matters. He has practised extensively throughout the Asia Pacific region and currently heads a team in the Sydney office of Middletons Lawyers with a strong software industry focus.

Software Development Agreements address situations in which companies outsource the development of a software application to a third party and establish, prior to the commencing of the development work, the rights and obligations of the parties in relation to both the product and the process.

Software development may be required in many scenarios including where:

- a company is seeking to lift its performance and, for example, requires a procurement management system that streamlines, records and prompts each stage of its procurement cycle in order to deliver better savings (Performance Enhancement);
- a company has identified an opportunity within the market to introduce a new product (Commercialisation); or
- an application is required in order to comply with various laws, for example relating to tax management or file retention (Compliance).

These are the most obvious categories of software development and any business would probably require a formal agreement to address the contractual commitments associated with these situations. However, software development also arises in less formal or clear-cut instances, such as when a company's contracted IT consultant develops a minor patch to interface between the company's intranet and a database. While payment for the development work in this case may be covered by a general retainer agreement, the development work itself may well be the subject of a separate agreement, whether or not written.

Continues page 3

In this issue:

Continued from page 1

Contract management is crucial to any software development. However, it is important to draw a distinction between contract management and contract administration. While contract administration relates to ensuring that agreements are properly on foot and that key dates are not forgotten, managing IT contracts (in particular software development agreements) is a process that should begin from conception of the "deal" to ensure that the transaction is properly tracked, monitored, documented and controlled to deliver optimal outcomes, using the agreement to steer the relationship and the obligations of the parties.

Good contract management begins with the actual agreement because it is through the process of negotiating and documenting the agreement that the parties address issues, resolve concerns and, most importantly, maintain high levels of trust and confidence and achieve both their shared and independent objectives.

This article considers the key issues that one should consider prior to embarking on the software development exercise and which should be addressed in the Software Development Agreement, where possible, highlighting the distinct concerns of both customers and software developers. This article also considers two recent court decisions in relation to software development agreements and their practical implications for software developers and customers.

Planning

Planning is the most important stage and, from a contract perspective, it is very helpful to have an understanding of the following matters:

- What are the customer's operational needs (**Requirements**)?
- How is this product going to be used (**Objectives**)?
- What are the technical and hardware limitations (**Specifications**)?
- Does the customer seek to own the intellectual property rights (**IPR**)?
- Who will ascertain and scope the requirements and specifications (**Scoping**)?
- What critical events relate to the development project (**Milestones**)?
- How will the software be tested and by whom (**Testing**)?
- How will the acceptance plan be formulated and applied (**Acceptance**)?
- How will the software be delivered to the customer and who is responsible for the roll out (**Implementation**)?
- How will on-going support and maintenance be coordinated and will service level agreements apply (**On-going Service**)?
- What are the key risks that the customer is seeking to avert (**Warranties/Indemnities**)?
- How has the deal been sold and are the expectations reasonable (**Bid/Representations: RACV v Unisys**)?

Nature of Customer

The more sophisticated the customer, the more likely the customer is to have a view on the above issues. However, often the customer will rely heavily on the developer for guidance and assistance.

Conversely, developers will often incorrectly assume that a customer knows what it wants. This is one of the classic mistakes which leads to disputes in many IT agreements.

By considering the above issues from a contract perspective, the parties to a Software Development Agreement may pre-empt any concerns/issues and set out their accepted methods of conduct in such cases, creating the foundation for a much more cooperative relationship.

Benefit of Conflict

Customers not wanting to seem pushy and developers wanting to appear supportive will often avoid initial disagreement, so as not to miss what appears to be a "good deal" at the time. In reality, however, a solid debate of the fundamentals upfront is likely to flesh-out any concerns (of both the customer and the developer), assist to document resolutions in advance and prevent impasses later.

Neither the customer nor the developer should hesitate to vigorously negotiate important issues and requirements.

Requirements

A software development cannot succeed if the customer's requirements are not properly ascertained and documented. Unfortunately, the customer's requirements are often not adequately considered.

Given that Software Development Agreements typically deal with highly customised products, it is also vitally important to ensure that the customer's requirements are linked to the Software Development Agreement. From a contract management perspective, there is no standard way to address and capture these requirements. However, there are a few strategies that are quite effective and they include:

Attaching a schedule to the Software Development Agreement

Such a schedule should set out the specific requirements of the customer and then be referenced in the operative clauses of the Agreement.

Listing assumptions in relation to the Software Development Agreement

The benefit of listing assumptions in the Software Development Agreement is that they may address the concerns and requirements of both parties and they can relate to the product as well as the process. If you choose to list assumptions, however, be sure to also list consequences or resolutions for the failure of the assumptions and make these practical. So, for example, an assumption may be that the application will interface and operate in conjunction with the then current Standard Operating Environment of the customer and, if it does not, the resolution may be that the developer will, free of charge, modify the developed product to ensure its compatibility.

Warranties

If a requirement is particularly

important a customer should request a warranty in relation to that issue, bearing in mind that the value of the warranty may well be controlled by a "limitation of liability" clause. Similarly, a developer may seek a warranty from the customer that the customer has properly considered its requirements in relation to the proposed development and has described them accurately and completely.

During the process of ascertaining the requirements, it is important to ensure that the project is consistent with the customer's business plan, that the stakeholders approve and support the project and that the Software Development Agreement receives appropriate review by the appropriate staff of both the customer and the developer.

Objectives

In practice, there is often little overlap between the objectives of the customer and the developer. It is important, therefore, for each to be sensitive to the objectives of the other during the negotiations, the drafting of the agreement and also during any instance of dispute resolution.

Customer Objectives

As previously stated, the three key reasons that customers seek to enter into software development are performance enhancement, commercialisation and compliance. Some of the main contractual considerations in relation to such objectives are:

Performance Enhancement

If this is the customer's main objective, the parties should:

(a) benchmark the existing performance at the time of the Software Development Agreement and specify clearly what the target performance level to be achieved is;

(b) identify any constraints that apply to the use of the developed application that may prevent the customer from achieving the desired performance level when using the developed application; and

(c) consider how much training will be required and the feasibility of an education campaign.

Commercialisation

If the customer seeks the assistance of the developer to create an application so that it can be commercialised, the following concerns will be pivotal:

(a) the developer must be able to warrant that it is authorised to enter into the Software Development Agreement;

(b) rights related consents (including moral rights) must be obtained in advance;

(c) the confidential information of the customer should be adequately protected; and

(d) the customer may consider asking the developer to refrain from creating competing applications for other members of the same industry.

Compliance

Compliance related developments typically interpret applicable legislation and create tools to allow companies to comply with the legislation. In such cases:

(a) clarify who is responsible for maintaining the currency of the developed application throughout legislative changes;

(b) consider the frequency of updating the developed application; and

(c) ensure that the application will provide the customer enough time to rectify any errors generated by the developed application.

Developer Objectives

Developers will often seek to enter into Software Development Agreements for entirely different reasons, such as:

Market Share

The developer may simply be seeking a good solid sale and timely payment.

Reputation Enhancement

The developer may perceive the software development to be a good opportunity to add a reputable client to their client list, in which case the right to issue press releases and consider the project as a reference site may be central to the developer's needs. The customer may wish to impose conditions in relation to how their name and logo are used, particularly any trademarks that may be centrally controlled by an overseas head office.

New Market

The developer may be keen to enter into the Software Development Agreement in order to gain strategic know-how and to penetrate new markets for services that are up and coming.

The developer should ensure that it can freely use its know-how and that the know-how is not assigned in any restrictive manner to the customer. The customer should ensure that the developer has the financial means to complete the project, given that the developer may have made a very competitive (and possibly unprofitable) bid and ultimately find that the project was not properly costed.

Specifications

The specifications typically refer to the technical requirements of the application that are required for the application to function as intended.

Hardware required

From the outset, once the customer's requirements have been addressed, the developer should be able to define the hardware required for the developed application to properly function. Alternatively, sometimes the developed application will be created to match a particular hardware system, especially when the customer's IT infrastructure is costly and inflexible.

SOE Compatibility

Many larger organisations maintain a Standard Operating Environment (**SOE**) which is a template list of applications that are tested for the organisation's environment and delivered as a SOE to all desktops in that organisation. Subject to the sophistication of the customer, the developer will sometimes assume the role of testing the application for compatibility with the customer's

SOE.

If the developer is responsible for managing SOE compatibility, as well as the developed application, it is reasonable to expect that all relevant compatibility testing will be conducted by the developer. Otherwise, the developer will likely want relevant assurances from the customer or, alternatively, to expressly exclude this responsibility.

When considering the SOE, it is important to consider the actual products comprising the SOE, as well as future versions of those products and the means of distribution. These issues should also be considered in relation to the developed application.

Adaptability

The customer should clearly explain its requirements in relation to scalability and upgradeability. This links back to the importance of consulting with relevant stakeholders and the business plan of the customer. That is, is expansion of the business expected and will the developed application be able to grow with the customer?

Intellectual Property Rights (IPR)

The importance of ownership of IPR in a developed application varies. Often this issue will be the subject of significant negotiation, notwithstanding the fact that often neither party has a meaningful strategy in relation to IPR. Other times, even when IPR ownership is critical, the topic is often ignored by the agreement in which case, under copyright law, a lot of the IPR will remain with the author, which is likely to be the developer.

Ownership v Licensing

If the developed application is intended to assist the customer with issues of compliance or performance enhancement then the developer giving the customer a worldwide, perpetual, fully paid up, royalty free, non exclusive and non transferable licence may suffice. However, if the application is required for purposes of commercialisation then the customer is likely to require exclusive rights to the IPR or unencumbered ownership.

If licensing is the preferred approach, consider how this may affect the pricing. Typically, when the customer owns the IPR in the developed application the developer may charge higher fees, representing the developer's lost opportunity to licence the product to other customers.

Another very important aspect to the IPR negotiation in a software development is the treatment of pre-existing and third party IPR's, which should be appropriately licensed as required to meet the parties' respective objectives.

Tax treatment

Software development may be able to assist a customer to secure valuable tax concessions. A tax specialist should be consulted in this regard to ensure that the Software Development Agreement reflects the requirements of the legislation, if this aspect is important to the customer.

As explained in the case of *Industry Research & Development Board v Unisys Information Services Australia Pty Ltd (formerly Synercom Australia Pty Ltd)* [1997] 777 FCA (19 August 1997), the main test as regards the tax concessions is whether the activity involves innovation or technical risk. However, as is demonstrated by this case, this is not a simple test to apply and expert advice should be obtained.

Exclusivity

Given that Software Development Agreements will often be geared to enhance or create competitive advantages, issues of exclusivity arise during negotiations or, if not, they can arise later as a matter of dispute. It is important to consider issues of exclusivity in the very early stages. Referring to the customer's business plan and understanding the customer's objectives should assist in this regard.

Assignment of Rights

Any assignment of IPR in the developed software (or any component part) should be in writing, complete and effective from the relevant date. The parties should be careful not to neglect associated rights of privacy, confidentiality, exclusivity and moral rights.

Confidentiality

While seemingly obvious, confidentiality provisions should be included in each Software Development Agreement and, more importantly, the parties should administer such provisions rigorously to ensure that confidentiality is maintained.

Non-compete clauses

A non-compete clause is really an extension of the exclusivity and confidentiality provisions, effectively seeking to ensure that the parties to the Software Development Agreement do not encroach on each other's commercial territory.

If the Software Development Agreement is drafted such that the developer creates an application for the benefit of the customer, which it then licences to the customer, then the customer may require a non-compete clause which prevents the developer from directly targeting clients of the customer or providing the application to the customer's competitors. In other cases, the developer will insist that the customer not compete with the developer. This may be the case when the developer is seeking to recover costs and make a profit by on-going licensing fees for the application.

Scoping

While the key task in the Software Development Agreement is the development, many critical ancillary tasks are often included that are likely to impact the successful management of the software development. The most preliminary of these is scoping the development.

Scoping is a very important phase, particularly when the customer is not certain as to its own requirements.

Consultancy Agreement

One option for the scoping is to enter into a separate preliminary consultancy agreement. Typically these are on a time and materials cost basis. The deliverable of such an agreement may well be a project plan for the development of the application or

perhaps a list of options.

Module of Software Development Agreement

Sometimes the scoping exercise is conducted as a discrete module or phase of the Software Development Agreement. This is particularly the case when the application is derived from the standard product of the developer that is to be modified slightly for the purposes of the customer. Customer sign-off on and payment for the scoping deliverables should be pre-conditions for proceeding to the development phase of the project. The benefit of this approach over the consultancy agreement is that less time is wasted between the phases on the negotiation of the agreements.

Time frame

While at times an additional phase may appear time consuming, it is often the case that time spent scoping, designing and planning is often a fraction of the time that would otherwise be spent arguing, disputing and contesting the Software Development Agreement.

Skill base

Skill-base is a very qualitative aspect of IT contracting. The customer should consider insisting that the same consultants/project managers are involved in the scoping and the development phases and that, in any event, all staff involved are of an appropriate level of experience and education/skill.

Milestones

Milestones can assist the parties to a software development agreement to manage the risk involved in each stage. Milestones effectively provide a cleaner modular structure that facilitates early addressing of issues/termination when a party fails to perform in accordance with those milestones.

Milestones are particularly useful when the parties have priced the software development on a fixed price basis and therefore have increased the risk levels. It is important to ensure that any milestones described are meaningful, so that if the Software Development Agreement is terminated on the failure to attain a milestone, a substitute developer may easily be engaged to complete the project if required.

Aggressive customers will request that payment of any fixed price be linked to the completion of a deliverable which may well be denoted by the attainment of a milestone.

Change Management

If the software development is complex enough to be able to be broken down into milestones, the parties should ensure that a change management clause is included to facilitate any changes that are required and/or to clarify expectations.

Testing & Acceptance

Testing and acceptance are often linked. Given that the developer is likely to have a better knowledge of the developed application than the customer, it is worthwhile conducting the testing in a collaborative fashion and under the guidance of, if not by, the developer in the presence of the customer.

The testing plan should relate specifically to the requirements of the customer and should, if possible, be crafted well in advance so that acceptance can be structured and easily explored.

Testing environment

Testing may be conducted in any number of environments. However, from a customer's perspective, acceptance of a product before it is installed and operating on the customer's system could be risky insofar as it could complicate any later customer claims that the product is not acceptable.

User Testing

Given that the benefits of developed applications are often qualitative rather than quantitative, the parties should consider that more interactive methods of testing may be required. For example, an application may be able to deliver all the required user reports and technically satisfy the requirements of the customer, however, it may be user-unfriendly to the extent that it does not achieve its purpose of performance enhancement. In such a case, end-user testing may be appropriate to determine if the customer's ultimate commercial requirements are satisfied.

Acceptance

As stated above, acceptance should be a consultative process both internally and externally. The same stakeholders who were consulted at the time the customer's requirements were collated should be consulted again at the time the product is being tested.

Most agreements will have a provision that states that if the customer fails to promptly test/accept the application it will be deemed accepted after a certain period. Customers should consider refining such a provision to state that deemed acceptance may only occur after the product is used for a certain period, rather than simply having been delivered for a certain period. Developers will often accept a reasonable timeframe.

Formality

Whether or not the Software Development Agreement provides for such, the developed application should be accepted formally in writing. Similarly, any reservations should be submitted in writing to the developer with instructions as to how it appears that the developed application falls short and a time frame should be negotiated for the rectification of the defects. Issuing such notices will be invaluable to the customer if the matter is ever disputed later down the track.

From a developer's perspective, these notices are just as important. During the testing and acceptance phase, documenting the communications between the parties may be a strong tool to not only record matters as they actually transpired but also to work out resolutions to any issues that arise.

Retesting

It is important to repeat testing if the developed application is modified based on concerns/complaints at the time of testing and acceptance. It may be that the testing plan should also be modified to increase the relevance of the results, but neither party should neglect the duty to re-test.

Hybrid acceptance

If the defects are minor or merely cosmetic, a conditional acceptance may be provided with a timetable to complete the residual tasks, as long as a viable work-around has been delivered.

Warranty Periods

It is a fair requirement that the warranty period extend, at least, until what would have been the period for deemed acceptance (if the customer had not formally accepted the developed application). It is helpful to know that even if the application technically passes testing and is accepted, other defects that were not foreseen may still be covered under a warranty provision.

Fundamental Failure

Developers should be very thorough with their testing. The recent case of *Unisys Australia Ltd v RACV Insurance Pty Ltd & Anor [2004] VSCA 81 (14 May 2004)* (discussed below) demonstrates that even an "accepted" product can be the subject of litigation against the IT provider. Notwithstanding that a product has been accepted, a fundamental failure to meet the customer's requirements may still be considered by the courts.

Implementation

Similar to the discussion in relation to scoping, implementation of a developed application can be either tacked on as a phase of the total project or conducted under a separate agreement.

Given that many projects fail for poor implementation and transition, it is often in the interests of both the customer and the developer that a competent developer project manage the implementation. This assures the customer that a knowledgeable professional is rolling out the developed application and the developer can reduce exposure to a disgruntled customer suing the developer if the customer conducts a substandard implementation.

Project and Risk Management Practices

If the developer is tasked with rolling out the developed application, both parties should ensure that sound project and risk management tools and processes are deployed and the customer should seek a warranty that the developer's staff are sufficiently experienced and trained.

Licensing Third Party Software

If the application relies on any other software such as third party software, it is important to ensure that the appropriate licenses have been arranged by this stage. This is important for both parties, given the risk of software infringement and piracy claims.

A matrix of responsibilities is often helpful. Sometimes a developer may have access to competitive rates for the relevant licenses. However, on occasion the customer may already have a strategic relationship with a software vendor for the same product. In any event, the responsibility for the licensing of third party software should be clearly spelt out in the Software Development Agreement and the deadline for attending to this issue should be well before implementation of the developed application.

On-Going Services

One of the most important questions, post-implementation, is that of on-going support and maintenance. If the parties intend for the developed application to be supported by the developer then they should (either as a module to the Software Development Agreement or by separate agreement) enter into a support and maintenance agreement.

Support and Maintenance

The Software Development Agreement should specify whether or not maintenance will be provided. That is, what regular servicing, updates and upgrades, patches and modules will be provided and whether such maintenance will be pre-emptive and/or remedial.

Another option which may be delivered, in tandem with maintenance, is on-going support through a help desk with a dedicated hotline and response methods that are suited to the needs of the customer. Alternatively, the developer could offer a facility (often described as "no support") where support is provided on a reasonable efforts basis during business hours only at the developer's standard hourly rate. This option may be preferred if the developed application is considered very easy to maintain or, in other words, it is very stable.

Service Level Agreement (SLA)

If the developer does support the developed application the customer should consider requesting a service level agreement from the developer to ensure that its support needs are met. The developer also benefits from an SLA because the developer can exclude support in unreasonable circumstances, cap the obligation to support at achievable levels and manage customer expectations.

Code Updates

If the developed application is supported by the developer, the customer should request that the developer regularly consolidate and update a separate back-up copy of the code that the customer holds. This is also helpful if the customer ever needs to seek support from a third party.

Alternatively, if the application is only licensed to the customer, the parties may agree to leave a copy of the code (which is updated from time to time) in escrow with an agent on standard escrow agreement terms.

Future development

A Software Development Agreement should also address the customer's requirements for additional development services. However, these could also be included as variations to the scope or, once again, as additional phases or even separate agreements.

Unisys Australia Ltd v RACV Insurance Pty Ltd & Anor [2004] VSCA 81 (14 May 2004)

In March 1993, RACV issued a Request for Information (RFI) for the development of a real time/near-line storage system. Unisys, having demonstrated relevant solutions to RACV, delivered various documents

as well as a formal response to the RFI. The parties signed a contract in December 1993 that did not refer to the RFI or the other documents provided by Unisys. The executed contract included the usual exclusions and liability limitations.

Two years later, in March 1995, Unisys delivered a system that did not meet RACV's expectations for response times and document availability. According to the RACV, the project had failed. While RACV afforded Unisys an opportunity to rectify, the system still did not meet RACV's expectations and in June 1996 RACV finally terminated the contract and sought damages from Unisys.

The Basis of the Claim

RACV's main claim was that Unisys had breached section 52 of the Trade Practices Act by engaging in misleading and deceptive conduct in trade or commerce.

The Court at first instance held that Unisys knew of RACV's requirements for a system with rapid response times and that RACV would not have appointed Unisys if they did not believe that RACV would be receiving such a system from Unisys. The Court held that Unisys had represented that it would be able to deliver a system which met RACV's response time and other requirements, this representation had not been met and therefore Unisys was liable.

The matter was appealed to the Court of Appeal of the Victorian Supreme Court and Phillips J, who rejected the appeal and was supported by Batt and Ormiston JJ, quoted the trial judge as follows:

> *"The fact is, I find, that a fundamental premise of the engagement, understood by Unisys, was that the system implemented would provide retrieval in a timely manner. So much was obvious as required to meet the business purposes of RACV. But over and above that was the requirement of an on-line system in which the retrieval times specified in the RFP were inherent. This was understood by Unisys and hence the explanations and representations of Josephson and Unisys' cognisance of the importance of retrieval time signified by the above references. I reject the submission that under the contract Unisys was to deliver, and RACV was to accept, a system with whatever retrieval times it might come to deliver."*

Defences

The initial response of Unisys to the RFI stated that further investigation was required before the solution could be properly scoped. This is a common device used to give the supplier room to move. The Court held that this was a general statement which could not override Unisys' specific promise about delivering a system which would perform.

Unisys also argued that it had disclaimed response times in its RFI, some 70 or so pages after its impressive statements about what it would deliver. The judge was unimpressed with this hidden disclaimer indicating that this, in itself, could be seen to be misleading. In any event, he found that in the face of Unisys' earlier statements in the document about meeting RACV's requirements, the disclaimer could not be read as meaning what Unisys contended that it meant.

Unisys pointed to the written contract and said that nowhere was the response time requirement specified. Unisys argued that the contract superseded the earlier discussions by its express terms and RACV could not rely on the earlier documents. The Court held that the contract was not effective to exclude the operation of section 52 of the Trade Practices Act and that Unisys remained fully liable for its earlier unfulfilled promises. There appears to be no neat contractual band-aid solution for earlier promises made to secure the project.

Finally, Unisys claimed that under the signed contract it had committed to building a system based on a particular functional specification which did not cover response times. RACV argued that Unisys had committed to build a system which was fit for RACV's purposes. The Court held that the only way to make sense of the functional specification was to look back to the RFI. The Court therefore rejected Unisys' argument on this point.

Conclusions

There are a number of practical lessons that software developers and customers can learn from the decision in *RACV v Unisys*, which are briefly noted below:

Developers

(a) Avoid vast and un-costed pre-contractual promises that cannot be fulfilled. The existence of a clever contract will not necessarily save a developer from the consequences of breaching section 52 of the Trade Practices Act.

(b) Ensure that disclaimers and assumptions are clear and are positioned near the statements they are designed to limit. Courts may frown upon "hidden" limitations and disclaimers.

(c) Always involve your business assurance managers, technical stakeholders and lines of service when bidding for a project to ensure that your promises may be delivered with certainty.

(d) Actively control the customer's expectations at all times. This is also part of sound project management.

(e) Ensure that change control is properly conducted and explain to the customer how change may impact the projects objectives and the attainment of the customer's requirements.

Customers

(a) As previously discussed, do not cut corners by not documenting your requirements. Make sure this is done in the most accurate and precise fashion that is feasible.

(b) Conduct due diligence on your developer and bear in mind that most demonstrations are not in a live environment and do not deal with real data and systems.

(c) Closely examine all assumptions of the developer, the responsibilities that have been delegated to you, the constraints and any documented processes regarding how these are dealt with if they fail. These provisions will often mask disclaimers.

(d) Make sure the contract reflects

both the agreement negotiated as well as the ancillary promises and any expectations that the developer has lead you to hold during negotiations or in a response to a tender request etc.

(e) If possible, break the implementation into milestones so that damage control can be used early rather than waiting for a fundamental failure to arise.

(f) No matter how large or small the development, plan and document it well. The smallest project can carry the same amount of risk as the largest if it becomes the weakest link in your IT systems.

Ateco Automotive Pty Ltd v Business Bytes Pty Ltd & Anor; Business Bytes Pty Ltd & Anor v Ateco Automotive Pty Ltd [2003] NSW SC 197

Ateco Automotive Pty Ltd ("Ateco") distributes cars, their parts and machinery throughout Australia and offers warehousing facilities to other automotive companies. Ateco had implemented an inventory management system as its profitability relied on efficient inventory control.

Having been supported on an ad hoc basis by Mr Maurice Villari for around 8 years, Ateco appointed Mr Villari's company, Business Bytes Pty Ltd ("Business Bytes"), to provide regular services as well as technical support for Ateco's computer systems in 1993.

The agreement for the services was formed by correspondence and various conversations exchanged over the period of a few years.

Project management was poor and, more critically, no clear statement of requirements was provided and Ateco's staff were rarely available to be trained on systems.

While one of the core modules was defective, the system delivered was substantially effective and its appropriateness was confirmed by an expert. Ateco paid $724,000 to Business Bytes as well as $155,613 to third parties for that system. However, due to Ateco's dissatisfaction with the system it decided to purchase an alternative system from another vendor for $967,355.

The Basis of the Claim

Ateco sued for breach of contract, negligence and claimed under section 52 of the Trade Practices Act seeking a refund of fees and Business Bytes cross-claimed against Ateco for $222,552 of outstanding fees.

The Decision and its Implications

The Court held, dismissing the claims under the Trade Practices Act, that the mere delivery of an imperfect system did not mean that the contract failed for want of consideration and that, had Ateco been more cooperative, many of the problems could have been avoided. The Court ordered Ateco to pay the outstanding fees.

This decision illustrates that the Customer cannot reasonably expect the developer to be responsible for every aspect of the project without providing reasonable cooperation and assistance. The best approach to managing a relationship is joint management, particularly when the customer's requirements are dynamic.

To Conclude

Managing IT contracts in general, and Software Development Agreements in particular, is all about being proactive. The key is to know the agreement as well as the transaction and ensure that the two are consistent from bidding and negotiation through to drafting and implementation.

Potential conflict and disagreement are better dealt with early, reasonably and diligently rather than avoided until the matter becomes a serious issue. Recent cases demonstrate an increased willingness of parties to IT agreements to seek the assistance of the courts, even in light of pre-existing lengthy and fruitful business relationships. As a result, good contract management is now, more than ever, linked very closely to good risk management. By actively planning the relationship, carefully drafting the agreement and responsibly managing the contractual processes, it is likely that a software development will be successful for all concerned.

This article is based on a paper presented by Alec Christie at the seminar "Drafting & Negotiating I.T.Contracts" conducted by Legalwise Seminars in Sydney on 26 August 2004."

* The assistance of Lirun London Rabinowitz, Senior Associate of Middletons, in writing this paper is gratefully acknowledged.