In-house programming: an option for small and medium sized niche companies[1]

Kai A. Olsen,
Department of Informatics, University of Bergen and Molde University College,
Norway
kai.olsen@himolde.no

Abstract

In an inter-galactic battle in a science fiction novel by Isaac Asimov (1958), computers are an important weapon. As long as anyone can remember, computers have been used for virtually everything. Then, a simple technician presents his hobby: the ability to add, subtract, multiply and divide using pen and paper. It was hard to believe: "Computing without a computer is a contradiction in terms." The military sees the possibilities at once. The new method offers flexibility and the ability to put humans into the loop, as opposed to the rigidity of the automatic systems. With this new, top-secret tool, they can think out of the box – in the end - defeating the enemy.

Today, we don't have a war between galaxies, but we do have strong competition between businesses. Computers are an important tool in this struggle, but do not offer any strategic advantage since the technology is available to all. Businesses use similar machines and similar software, sometimes even identical ERP systems implemented by the same experts. This raises the question of whether programming could be a way for small and medium-sized enterprises to get "out of the box."

1 Introduction

An enterprise that is considering a computer system has three alternatives. It can buy a set of off-the-shelf packages, install an ERP (enterprise resource planning) system, or program its own system. Many small and medium-sized enterprises (SMEs) choose the first option. A wide range of packages are available that can deal with order handling, accounting, procurement, keeping track of inventory, etc. The problem is that few of these packages cover all functions, and those that do are unlikely to handle the core applications for a niche-oriented company. Large companies often choose the ERP solution. These systems have embedded standard business practices and can encompass all administrative functions, but may not suit smaller companies that need to be flexible in order to survive in a dynamic market. The third option, in-house development, is rarely considered because it is viewed as expensive and time-consuming.

This paper challenges this view, using three cases to show that there are many possibilities for in-house development. The solution is to use modern programming tools to simplify development. Most importantly, we shall see that existing standards make it feasible to interconnect off-the-shelf packages and system modules that are developed in-house; that is, to create a customized "ERP" system.

The target of this study is small and medium-sized niche companies whose business idea may be to perform a specialized task or deliver a specialized product. Their customers are often

[1] This paper was presented at the NIK-2011 conference; see http://www.nik.no/.

larger companies. Small companies have little control over the market; they survive on their skills and by being able to respond quickly to market needs. Therefore, as we shall see, small organizations may have different computer system needs than their larger counterparts.

There are clear and significant differences between the ways in which small and large enterprises operate. The latter are often characterized by large numbers of employees, customers, products, orders, inventory, invoices, etc. These high volumes reduce the company's ability to operate in a flexible manner. There is a need for fixed (and often quite rigid) procedures, just to maintain control and efficiency. Large enterprises may operate in many countries and must therefore adjust to different tax systems, laws and business practices. Consequently, these companies need an overall system that can handle all types of transactions and can integrate data from all functions and departments. At the same time, they will have the degree of formalization required for these systems to work well.

In contrast, small and medium-sized companies will have fewer employees, fewer orders, fewer products, etc. One employee may always handle all tasks of one type. This allows for flexibility, as the employee in question will know when to stick to the procedures and when they can be set aside. Of course, there may not be any rigorous rules in the first place. This flexibility is necessary in order for such companies to survive. The market niches of small companies are often defined and bordered by the areas in which the large companies operate. Survival is based on being agile and able to quickly adjust to outside pressure and change. Many small firms are suppliers to big firms. In such cases, the larger customer may set the agenda, defining the form in which orders will be sent, adding specifications in formats of their choice, and requiring documentation that is in accordance with their needs. The small firms must comply if they wish to keep their customer. Therefore, niche companies will need an IT system that allows them to retain their own proprietary business processes.

1.1 The default solution – a portfolio of off-the-shelf packages

Today, the most common alternative for small companies is to acquire a set of off-the-shelf software packages, for functions such as accounting, finance, inventory management, or production planning. The large number of such packages on the market allows a company to choose a set of systems that fits its needs. In contrast with the most commonly used ERP systems, many of these packages have a recent design; that is, they use state-of-the-art functionality. Most of the packages come at reasonable prices, and some are even free.

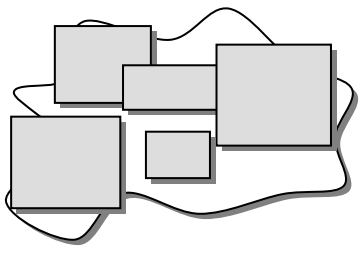

Figure 1. Satisfying special needs by standard packages.

However, these standard packages will not perform all functions in a company. Figure 1 illustrates this point: the outer border describes the needs of the company, and the grey boxes represent the functionally that the standard packages cover. The parts that are not covered are handled manually or, more commonly, by manual processes that are supported by general programs such as word-processing and spreadsheet programs. The more niche-oriented a company is, the lesser the chance that standard programs can fulfill all of its needs.

This is the situation for many SMEs. The fact that manual routines cover most of the company's functions helps maintain flexibility. Efficiency is not so good, however, as manual handling consumes time and money. The same data must often be entered in more than one system, and it is difficult or impossible to create reports that are based on data from all systems. This inefficiency goes against the wishes of most companies to be as "lean" as possible; that is, avoiding unnecessary work both in administration and in production

(Womack and Jones, 2005). Lean processes require an integrated system; that is, where data is entered once and where entire processes can be automated.

1.2 ERP systems

The idea behind ERP systems is to have one system for the whole enterprise. Data is stored only once, shared by all, and has a common user interface. It becomes easier to keep track of inventory, to make reports based on data from all departments, follow orders through the company, and gain a better view of revenue and costs.

However, such systems can become huge and complex, and implementation is risky (Davenport, 1998; Parry, 2005). Since they are generic in nature (that is, designed for a general enterprise), extensive configuration is required in order to mould the system to a particular enterprise. In fact, it is often necessary to mould the company to the system; for example by replacing the company's own functions with those of the ERP system. This may be an advantage when the "best practices" of the ERP system are an improvement, but competitive advantages may be lost when idiosyncratic functions are replaced by more general ones (Akkermans et al., 2003). This loss may be greater in a small company, such as one in a niche industry, than in a larger company.

All ERP systems allow customers to build extensions that can work together with the system. Service-oriented software architecture has made it easier for third-party vendors to program add-ons (Dörner et al., 2009). Still, such modules are expensive to build, especially if they are closely integrated with the ERP system. Also, these extensions may have to be adjusted to every new version of the ERP system. The systems are often quite complex and developed over many years. Even if vendors can offer cheaper versions for SMEs (Olson, 2004; Johansson and Sudzina, 2008), the systems are expensive and implementation may be challenging (Malhotra and Temponi, 2010; Hasan et al., 2011). Expensive consultants are needed for configuration, setting up the basic system, and customizing the system to the company. After implementation, many companies find that the system is rigid and difficult to change. An agile business wants its IT systems to move along with the company. The loss of flexibility may come at a high price.

2 The third alternative: In-house development

Both of the solutions presented above can have serious drawbacks for niche companies. The third alternative is for the company to develop its own software. In the early days of computing, of course, this was the only alternative. There were few packages and ERP systems were in their infancy. In those days, most applications had to be developed in-house. Many managers remember this period as problematic. They recall projects that went over budgets; expensive systems that were never finished; and systems that were slow, difficult to use and prone to errors.

Therefore, many companies today do not view in-house programming as a realistic alternative. However, programming has changed over the last 30 years, from programming languages such as C, FORTRAN, and COBOL to today's excellent development environments and application generators. An abundance of open-source code offers modules that can be inserted into the code, either directly or after modification. Now, whole systems can be put together with little effort (Mufatto, 2006). In addition, software engineering principles can be adapted to smaller companies.

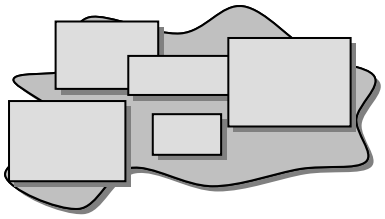

Figure 2. In-house systems as "glue" between the standard packages.

Most importantly, it is possible to avoid developing huge systems. Instead, a company's strategy can be to assemble a portfolio of off-the-shelf IT packages, only using in-house development for core functions. The proprietary part will then not only encompass niche-oriented functions, but can also act as "glue" between the in-house systems. This is illustrated in Figure 2, where the dark grey area is the in-house system, which performs kernel functions and communicates directly with the standard packages. Ideally, the system would cover all functions that a company needs (as visualized by the outer border).

This strategy will accommodate the company's need for flexibility and for maintaining the idiosyncrasy of the enterprise. In other words, the IT system will be aligned with the business strategy (McGovern and Hicks, 2004; Yen and Sheu, 2004). The proprietary part should only perform functions that the off-the-shelf packages cannot handle in a satisfactory manner. Such a strategy allows the IT systems to move along with the company and, as we shall see, IT can be used as a tool for implementing novel ways of doing business.

Behind this portfolio idea is a set of IT standards. These make it possible to let the different parts share data and to behave as one system. Although each part may have its own user interface, the fact that all modern programs are developed according to the same basic ideas of human computer interaction means that the various interfaces will have a similar "look and feel." Therefore, even if many systems are in use, the employees will see them as one system.

2.1 IT standards as the facilitator for proprietary development

Over the last 40 years, a huge range of IT standards have been used. Examples include SQL (Structured Query Language) for database operations; ODBC (Open Database Connectivity) for connecting to a database; TCP/IP (Transmission Control Protocol/Internet Protocol) for communication; SMTP (Simple Message Transmission Protocol) for sending mail; MIME (Multipurpose Internet Mail Extensions) for adding attachments to email; HTML (Hypertext Markup Language) for describing layout of web pages, XML (Extensible Markup Language) for describing the structure of web-based documents, and HTTP (Hypertext Transfer Protocol) for transmitting web pages. These standards have made it possible for SMEs to implement new and competitive IT strategies.

When a company chooses off-the-shelf packages that it will use together with its proprietary parts, the packages must support existing standards. For example, in order to achieve full integration between the proprietary program and an off-the-shelf package, the programmer must read from the package's database, store data into this, and perhaps also execute functions in such a package from the proprietary system. The company's aim is to build a complete system, its own "ERP," where these packages act as components. This can only be achieved if the packages support basic database standards and if the architecture of the database is open; that is, if it is possible to obtain a full description of the database structure that the package uses. In addition, many modern systems come with extensive macro facilities. Macros can be used to extend the functionality of a standard program so that it can be adjusted to suit the user's needs.

2.2 Software Engineering

The traditional processes of software engineering, which were often cumbersome and rigid, have been challenged recently. Many researchers advocate a more agile approach that emphasizes individuals as opposed to processes, working software over extensive documentation, customer collaboration over contract negotiation, and responding to change more than following a plan (Williams and Cockburn, 2003). Agile development covers methods such as Extreme Programming, Adaptive Software Development, Feature-Driven Development, and Dynamic Systems Development Methodology. Several authors have offered compelling arguments for the new methods (Beck and Andres, 2005; Cockburn, 2002).

For example, extreme programming stresses the importance of early prototypes and rapid feedback, with an emphasis on testing (Beck and Andres, 2005). Agile methods are claimed to work best when rapid change is the norm. Most advocates of extreme programming find that agile methods work best with smaller teams (Williams and Cockburn, 2003) and avoiding life-critical software. Lippert et al. (2003) discussed the complexities involved when using extreme programming for projects with multiple customers and multiple programming teams. However, the discussion in this paper is limited to proprietary development; that is, where there is only one customer and probably only one installation.

2.3 Proprietary software development

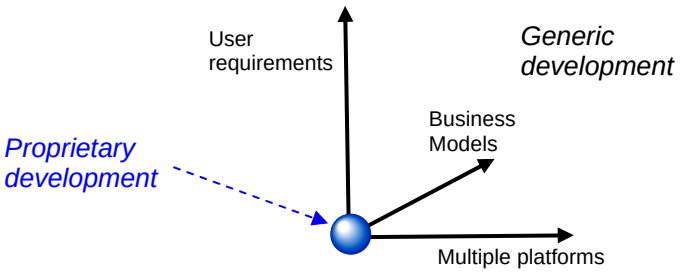

Figure 3. Software development in a 3D space.

Most ERP systems can work on multiple platforms (UNIX, Windows, Linux, etc.), fulfill requirements for a large set of users within many industries, and be applied to different business models (customization). This implies development in a three-dimensional space, as shown in Figure 3. However, proprietary development is often performed on one platform, with requirements from a limited set of users and with only one business model. This is especially the case when developing systems for a SME. Therefore, less effort is required than is the case when developing more generic systems – the 3D space has imploded into the origin, as indicated in Figure 3.

When developing for one business only, the "customization" is built directly into the system. This not only limits the complexity and the development effort, but it also simplifies training. Instead of a generic system, where users must be told how the system is to be employed for their tasks, users get a system that is tuned from the start to the processes in the company. Workflow in the system will match workflow in the company. On a more detailed level, the commands, forms, messages and help system will use the company terminology.

If the system is to have a strategic advantage, it must evolve with the company. That is, whenever business process changes, the system must be updated to accommodate the new practices. In such cases, rigid testing of each change may be impractical, which is where there is a clear advantage in having few users. In a large company, errors in the software can have severe consequences, as many users and large data sets may be affected. In a small company, it is possible to install software with only superficial testing, which increases the risk of bugs in the system. However, the author's experience is that, in a system consisting of software, hardware, machinery, human beings, customers, data entry, and data files of different formats,

software is the least likely part to fail. (Of course, some task critical systems are not allowed to fail; in these cases, rigid testing is the only possibility.)

Nowadays, IT can follow along with the agile company. When errors occur, they are fixed on the spot. This is often acceptable in a system with few users and relatively few transactions. The alternative is to leave modifications to periodic revisions, perhaps once a year. That is not a good alternative for an agile business. Of course, this flexibility is only achieved in the proprietary part. It will be more difficult to change the off-the-shelf packages. However, these packages are most often used for standard applications, such as accounting, invoicing, and inventory management. These are less agile than the core parts that implement the idiosyncratic business ideas. Also, if a change is needed in a standard application, such as accounting, the change can often be handled by reprogramming the proprietary part; for example, by modifying the data sent to the accounting system. As the cases below show, macros can also add to the flexibility of the standard packages.

3 Cases and research methodology

Three case studies of three Norwegian companies are present here:

1. A small publishing house
2. A company that produces furniture for ships
3. A foundry that manufactures propeller blades

Each of these can be described as a niche company. Their IT needs and solutions are presented below. The research methodology presented here can best be described as *action case*. Traditional research, whether positivist or interpretative, seldom allows researchers to delve deeply into the cases and spend an extensive amount of time with each company. However, as consultants (which is the role that the authors experienced in these cases), one can work closely with each company, thereby gaining a thorough understanding of the company's niche, business processes, and how its current IT systems implement these processes. The danger of this approach is that it may become difficult to report objectively (Hirchheim, Klein, Lyytinen, 1996).

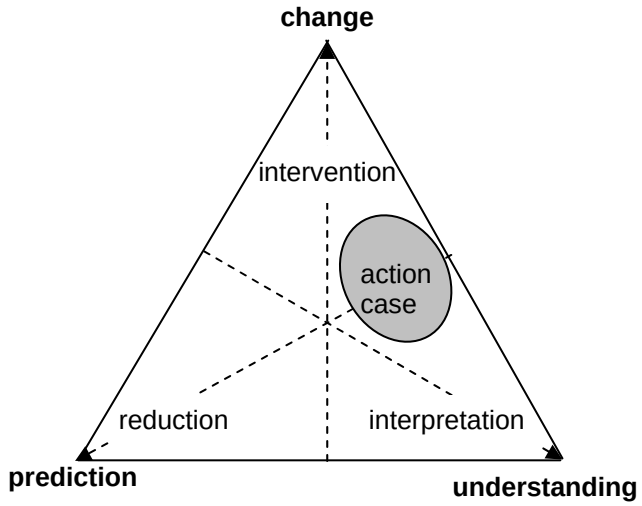

Figure 4. Research framework

Braa and Vidgen (1999) presented an idealized framework for information system research identified by prediction, understanding, and change (as illustrated in Figure 4). *Prediction* is the aim of a positivist approach, *understanding* the aim of an interpretive approach; finally, *change* is connected to an interventionary approach. The action case is positioned as indicated, with both an interventionary and interpretative element. While our work has an interventionary aspect, the aim has never been to study the effects of these changes in the same way as more formal approaches to action research. The research performed here has more in common with what has been labeled participative action research (Whyte, 1991).

4 Case description

Each of the cases is presented below. A short introduction of the company and its IT needs is followed by a presentation of the proprietary solution, together with the advantages and disadvantages of the solution. We start with a simple case and move to the more complex ones.

4.1 Case 1: A small publishing house

The company produces visual tour guides for hiking and mountaineering. The business idea is to produce books in which pictures are the important part; that is, the reader is directed along the trail visually. Each guide may have as many as 1500 images.

Bookstores are often reluctant to order books from a small publisher. To gain access, this publisher offers customized accounting models; for example, a model that guarantees a store that it will not have to pay for books that are not sold. The company has four part-time employees, two of whom have IT knowledge and programming skills. The company's IT needs are twofold: for administrative functions and for production. The administrative needs are fairly standard: keeping track of customers, products, inventory, deliveries, prices, invoicing, and accounting. At first, the idea was to acquire a standard accounting system that could perform all these functions, but the company felt this was overkill. In addition, some setup was needed in order for each system to handle the company's idiosyncratic accounting model. It was therefore decided to develop an in-house system. Using an application generator (Microsoft® Access in this case), a full system was developed with limited effort.

Microsoft Access comes with a programming language, a form builder, a database system and a report generator. With only 100 man hours spent on specification and programming, the cost of the proprietary system was found to be less than that of the off-the-shelf solutions. However, the main advantage is that the company is now fully in control. For example, when the major national bookseller chain in Norway installed an ERP system and changed its order routines, the case company was able to adjust to the new situation immediately, simply by implementing some small changes to its system. While the company experiences the benefits of being in control, by adjusting its administrative system according to the needs of the day, it also acknowledges that an off-the-shelf package could have performed these administrative tasks. When it comes to production, however, the company is convinced that it has found the ideal solution.

The books are typeset using a word processor (Microsoft® Word), which is unusual in a business that has numerous professional typesetting systems to chose from. However, by using the macro concept embedded in MS Word, the company has been able to customize the standard system to its needs. For example, the company has macros to adjust a set of photos to the page, to insert captions and symbols on photos (such as arrows that show hikers where to go), for drawing maps, and for checking the consistency of the guide. By adding some thousands of lines of macro code, the company has turned the standard word processor into a system suitable for producing visual tour guides. Many modern software products, including MS Word, offer a "macro recorder" that can generate code based on manual operations performed on the user interface. Therefore, a typical macro program development process will start by mimicking the operation that is needed; that is, by performing the operation manually. The recorded code can then be adjusted and generalized so that it will work in all situations. This simplifies coding to the point where anybody with basic programming skills can quickly create macros. No specialized knowledge of the inner workings of the word processor is needed.

4.2 Case 2: Manufacturing furniture for maritime applications

This company is Norway's leading producer of maritime furniture. It has 50 employees and delivers furniture to approximately 100 ships around the world each year. The company's business model has much in common with that of IKEA, relying on flat packages of standard furniture. This is a novel idea for the maritime industry, where furniture was traditionally manufactured for each cabin. However, in order to satisfy customers, and since cabin shapes vary considerably, a high degree of flexibility is offered, to the point where the cabin should look as though the furniture had been custom-made.

The part that we focus on here is the process that starts with the designer who models the interior of the ship. This person may be employed by the shipyard and will use a graphical system (most often AutoCAD®) to draw each deck (floor), with the bulkheads (walls) between cabins. Using a set of templates for different types of furniture, such as bunks, tables, chairs, desks and cupboards, the designer will then furnish the cabin. Once completed, the entire drawing of all decks will be send to the case company, where the data on the drawing will be entered manually into the production system. This is done by a person using two displays, with the drawing on one display and a data entry system on the other. This is not a formalized task; the designer may have used furniture that does not exist in the range offered by the company. For example, a bunk may be indicated as being 205 centimeters long, but this bunk may only come in 200 or 210 cm options in the standard product range. As we see it, the problem is that the designer has too much freedom when using the graphical templates to furnish the cabin. This problem is usually solved by choosing a standard component, but there will often be a need to ask the designer decide what to do (for example, the company can offer a 205 cm bed, but at a much higher price than the standard alternatives). Once all of the components have been registered, the complete furniture list is returned to the designer for confirmation.

The challenge here is to study how communication can be improved between the designer and the company. What is needed is a system that restricts the designer in such a way that the final furniture components in the drawing are represented as physical objects in the real world. In fact, it is not necessary to develop such a system. AutoCAD®, the most commonly used package for ship designers, already has a concept called *dynamic blocks*. Each block has a visual representation and a set of constrained attributes. For example, a block representing a bunk can be constrained to take only the lengths of 200 and 210 cm. This block concept has much in common with abstract data structures. The blocks have to be "programmed" using the syntax defined for this in AutoCAD. Other functions are programmed in AutoCAD's macro language – AutoLISP.

By providing designers with a palette (another AutoCAD concept) with dynamic blocks for all types of furniture, the company can ensure that the cabins are furnished with components that exist in its production line. In addition, the palette can present a 3D visualization of each block, showing a generated picture of a bunk, sofa, cabinet, etc. This allows the designer to view every piece of furniture or a complete furnished cabin before deciding what to use. The block will also embed an article ID. Instead of mailing a drawing to the company, the designer will now mail a list of all the furniture needed. This can then be imported directly into the production system, increasing efficiency and removing the possibility of errors. The savings are calculated to be 30 percent, which are shared equally between the customer and the company. By using features that are a part of the common package, the company has achieved an efficient proprietary system that offers savings, improved feedback to customers, and shorter lead times.

4.3 Case 3: Manufacturing propeller blades

This company has both a foundry for casting propeller blades for ships and a division that produces complete blades according to international standards. The company has approximately 40 employees and supplies companies across Europe that manufacture complete propeller systems. Although this is a niche company, only parts of the processes are niche-oriented. Like all other companies, it also has more standard parts, such as accounting and invoicing. While the standard parts are handled by off-the-shelf packages, the more proprietary parts are covered by in-house systems. Nonetheless, these different programs act as one integrated system.

The important processes may be described, in a somewhat simplified form, as follows. An order for a number of customized propeller blades is received. This may refer to a previous "model"; if not, a geometric description of the blade is enclosed. This data is then used to make a new model, which is produced out of wood in a five-axis milling machine. Due to the fact that metal shrinks as it cools down, the model must be made larger than the blade. The model is then placed in a box with a sand-fixture solution to make a mould. Finally, liquid bronze heated to approximately 1200 degrees Celsius is poured into this cavity. The capacity of the foundry is limited by the area that can be reached by the cranes, as well as by the total amount of metal that can be heated at any given time.

In practice, capacity is defined by a combination table that shows which sets of blade types can be produced at any time. After casting, the root part of the blade (the part that is to be connected to the center part of the propeller axis) is machined. The blade is measured and factors such as pitch and thickness are controlled according to an elaborate ISO standard. Excess material is then ground off. Finally, the blade is sent to the customer, along with documentation on metal quality and geometry.

No ERP or off-the-shelf package can support these specialized functions. Therefore, the company has developed its own proprietary systems: a planning system, a system to ensure that blades conform to the ISO standard, a system to produce efficient models, etc. The planning system employs a cutting-stock approach to the problem, using heuristic methods to optimize the plan by finding the best combinations to use on any given day (Nonås and Olsen, 2005). This feature alone has led to orders being packed 20 percent more efficiently than with the previous system. The systems are programmed in Microsoft Access and Visual Basic.

Previously, a manual routine based on a spreadsheet was used to calculate the amount of grinding needed to have the blade conform with the ISO standard. This process required an experienced operator and could take many hours; sometimes it was impossible to find a solution and a new blade had to be cast. Now, a program automates the complete process, from generating the measuring program and retrieving the results from the measuring machine to using brute force techniques in order to find the minimum amount of grinding necessary to get a blade that confirms with the ISO standard (Olsen, 2009).

Since casting is an inexact business, as much as five to eight millimeters of additional metal on each side of the blade for the casting has been acceptable; this is to handle the shrinkage and to achieve the allowance necessary to produce a ISO blade. Therefore, the standard method has been to inflate the nominal geometry by a certain percentage. The disadvantage of this approach is that the shrinkage is not constant across the blade; it may be great along the edges and less in the thicker part of the blade.

By carefully analyzing data from the program described above, the company was able to find the exact shrinkage for each part of the blade, for each type of blade. A new system was developed with the idea of implementing a "shrink-to-fit" policy; in other words, a model that

offered a casting that would fit ISO requirements directly. This system generates a model based on the nominal data (as received from the customer) and data on how different types of blades will shrink in different parts. In contrast to the "blowing up" strategy, this implies that a new geometry will be generated for the model. This is not a simple task, however. The system must generate new spline curves, smooth the curves, add additional reference points, etc. Nevertheless, this system does enable the company to operate with an allowance of between one and two millimeters. Metal is saved, while the costly process of removing excess material is reduced from days to hours.

Parameter	Blade weight 500 kg			Blade weight 1000 kg			Blade weight 2000 kg		
	Before	After	Diff.	Before	After	Diff.	Before	After	Diff.
Allowance [mm]	6	2	**4**	6	2	**4**	6	2	**4**
Hours grinding	27	12	**15**	38	17	**21**	54	30	**24**
Energy usage [kw]	118	53	**65**	167	76	**91**	240	134	**106**
Costs [$]	2500	1100	**1400**	3500	1500	**2000**	5000	2750	**2250**
Savings	**56 %**			**57 %**			**45 %**		

Figure 5. "Shrink-to-fit" savings for different parameters.

This is a good example of IT influencing business strategy. Figure 5 shows the savings from the "shrink to fit" process, with data from before and after the implementation of the system.

The proprietary systems are seamlessly integrated with the administrative part, such as the accounting system. Whenever a blade is produced, the proprietary system will store all details on the order directly in the accounting system's database. This system will then take care of all subsequent processes, such as invoicing and accounting.

5 Discussion

In a paper and a book entitled "IT doesn't Matter" (Carr, 2003, 2004), Nicholas Carr argued that IT has become a commodity. It is something that you need, just like electricity, but it will not provide any strategic advantage. Carr's claims have met a lot of opposition from the IT community. Perhaps the problem was that he did not qualify his argumentation. Today, we have the choice to either view IT as a commodity or try to gain strategic advantages out of the technology. Taking a "commodity" view, it may be a good idea to go for standard solutions: either an ERP system or off-the-shelf packages. The goal then is to reduce risk and costs, as Carr advocates. The requirement is that the system suits the company, or that the company changes to suit the system without any loss. However, in order to gain strategic advantages from IT, a proprietary solution should be considered.

This paper has presented a few examples of proprietary solutions. It is not possible to generalize from these, as the solution that each case company found would only suit that company's needs. What is clear, however, is that a proprietary solution can be achieved with limited costs and effort. In one of the cases (the propeller company), this solution was developed using a standard programming language, while the other cases used macro programming to increase the functionality of the tool. If successful, a proprietary solution can give a company a strategic advantage. While Carr acknowledges this, he says it will only apply until the competition follows suit. This is true, but a company should always aim to stay ahead of its competition.

The business areas of the three case firms are publishing, furniture, and propeller manufacturing, respectively. The competition in these areas should be expected to have high competence, and all companies will be quick to take advantage of new technologies within

their own field. However, the competition may not be able to follow up as fast in the IT area, where it may lack competence. It should be noted that we are referring to small and medium-sized firms here. Imagine playing a soccer game and, with the scores tied at half time, telling the other team that you want to play the second half on a tennis court, hoping that tennis is outside their area of expertise. This would not be allowed in sport, but business is not a sport. Therefore, strategic advantages can be achieved by moving the "game" to the IT field before the competition does so. While the competition does have the option to buy the expertise it needs, the main point may be to recognize the possibilities in the first place; that is, to understand that a proprietary system may offer a strategic advantage.

The propeller company has gained clear strategic advantages by developing customized software. By doing so, it has significantly reduced its costs, at the same time as being able to take more complex orders than before. The furniture manufacturer has gained an advantage since customers who use its system can instantly receive a large set of important data on their order, such as prices, weight, and fire value. With the next version of the system, this will be extended to include other products than furniture. The customer is also offered a discount; that is, a part of the savings that the manufacturer achieves by receiving exact specifications, without having to do any more work than before. The publisher has managed to reduce costs at the same time as being able to present higher quality work.

As Carr states, the competition may follow. However, any advantage can be described as strategic if it enables a company to maintain its lead. If any of these companies stop investing in better systems, they may lose their advantage. However, having an advantage today offers improved profits, higher quality, and new types of orders, all of which may provide the incentives and resources that are needed for continued improvement.

6 Conclusion

Most small and medium-sized niche companies need to be in control of their IT systems if they are to stay flexible, be efficient, and exploit the new possibilities offered by IT. In some cases, this can be achieved by employing standard software only, such as by adjusting the packages to the company's needs. Alternatively, in-house development of core functions may be a solution. We have seen that this is feasible for many companies. A company may take advantage of open-source code or use application generators that simplify system development. The most important point is that only the functions that express the company's unique idiosyncrasies need to be programmed; for other areas, the company may use off-the-shelf packages. With existing standards, these different programs may be integrated and viewed as one system.

These days, a company must be dynamic if it is to survive; this is especially the case for small companies. A company achieves flexibility by being in control of its IT. With control, a company can ensure that all processes run efficiently, supported by IT, and that these move along with the company. This provides a strategic advantage.

References

Asimov, I. (1958) *The feeling of power*, Worlds of Science Fiction, February, Quinn Publishing Company (also at http://www.themathlab.com/writings/short%20stories/feeling.htm).

Akkermans, H.A., Bogerd, P., Yücesan, E., van Wassenhove, L.N. (2003) The impact on ERP on supply chain management: Exploratory findings from an European Delphi study, *European Journal of Operational Research*, 146, 284–301.

Beck, K., Andres, C. (2005) *Extreme Programming Explained*, 2nd ed., Addison-Wesley.

Braa, K., Vidgen, R. (1999) Interpretation, intervention, and reduction in the organizational laboratory: a framework for in-context information systems research, *Accounting, Management & Information Technology*, 9, 25–47.

Carr, N.G. (2003) IT Doesn't Matter, *Harvard Business Review*, May, 41–49.

Carr, N.G. (2004) *Does IT matter? Information technology and the corrosion of competitive advantage*, Harvard Business School Press, Boston.

Cockburn, A. (2002) *Agile Software Development*, Addison-Wesley.

Davenport, T.H. (1998) Putting the enterprise into the enterprise system. *Harvard Business Review*, July-August, 121–131.

Dörner, C., Draxler, S., Pipek, V., Wulf, V. (2009) End Users at the Bazaar: Designing Next-Generation Enterprise Resource planning System, *IEEE Software*, September/October.

Hasan, M., Tring, N. T., Chan, F., Chan, H. K., Chung, S. H. (2011) Implementation of ERP of the Australian manufacturing companies, Industrial Management & Data Systems, 111(1), 132–145.

Hirchheim, R.A., Klein, H. K., Lyytinen, K. (1996) Exploring the intellectual structures of information systems development: a social action theoretical analysis. *Accounting, Management & Information Technology*, 6(1–2), 1–64.

Johansson, B., Sudzina, F. (2008) ERP systems and open source: an initial review and some implications for SMEs, *Journal of Enterprise Information Management*, Emerald Group Publishing Limited, 21(6), 649–658.

Lippert, M., Becker-Pechau, P., Breitling, H., Koch, J., Kornstädt, A., Roock, S., Schmolitzky, A., Wolf, H., Züllighoven, H. (2003) Developing Complex Projects Using XP with Extensions, *Computer*, June.

Malhotram R., Temponi, C. (2010) Critical decisions for ERP integration: Small business issues, International Journal of Information Management, 30(1), 28–37.

McGovern, T. & Hicks, C. (2004) How political processes shaped the IT adopted by a small make-to-order company: a case study in the isulated wire and cable industry, *Information & Mangement*, 42, 243–257.

Muffatto, Moreno (2006) *Open Source: A Multidisciplinary Approach*, Imperial College Press.

Nonås, S.L., Olsen, K.A. (2005) Optimal and heuristic solutions for a scheduling problem arising in a foundry, *Computers & Operations Research*, 32(9), 2351–2382.

Olsen, K.A. (2009). In house programming is not passé – automating originality, *IEEE Computer*, April, 114–116.

Olson, D.L. (2004) *Managerial Issues of Enterprise Resource Planning Systems*, McGraw Hill, USA.

Parry, G. (2005) Counting the cost, IEE Manufacturing Engineering, February/March, 22–25.

Whyte, W. F. (Ed). (1991) *Participatory action research*, Sage, Newbury Park, CA.

Williams, L., Cockburn, A. (2003) Agile Software Development: It's about feedback and Change, *Computer*, June, 36

Womack, J.P., Jones, D. (2005) *Lean Solutions*, Free Press, New York.

Yen, H. R., Sheu, C. (2004) Aligning ERP implementation with competitive priorities of manufacturing firms: An exploratory study, *International Journal Production Economics 92*, 207–220.