The Buy versus Build Shift

Erik Dörnenburg, January 2013

This is a draft version of an essay to be published in the book "Right Sourcing" by the Sourcing Initiative.

When a new IT solution is needed in an enterprise, maybe because the business is changing or maybe because an existing manual process should be automated, the people who are in charge of implementing the solution usually quickly get to the question: should we build the solution or should we buy a package? For a long time the accepted wisdom has been to always buy when possible and only build when no suitable packaged solution exists in the market.

In this article I want to explore the reasons behind the existing preference for buying and, more importantly, I want to challenge the accepted wisdom and explain why I believe that the changes that have occurred in software development over the last decade have shifted the answer to the buy-or-build question away from buy and more towards build. I am not going as far as suggesting that build should be the new default but I think building software should be considered more often and it should be considered even when a suitable package exists.

Traditional reasons for buying

In my experience the case for buying software is often not made on the back of arguments that show why buying is a good idea. The usual route seems to be to analyse why building is a bad idea. This in itself says something. It is as if building is the natural answer and only problems with it make it necessary to take a different approach. So, what are these problems that have led people to preferring to buy a package?

Software development is risky

The most common argument against building software is the understanding that building software is risky. There are stories abound of failed software projects and of software projects running late, exceeding their budgets not by a few percentage points but by many times the originally anticipated cost. One way for an organisation to address this risk is to buy the software for a pre-determined price from a vendor. This shifts the risk of budget overruns to the vendor, and if the software has been completed before the purchase is made, as is generally the case with packed software, then there is not even a risk that the software will not be ready on time.

The obvious question is whether there really is something inherently risky in software development that has caused so many projects to be unsuccessful. On the one hand software development is a comparatively young discipline, at least when compared to architecture, in the building houses sense, or to industrialised production of goods. Consequently, there is less experience, practices are not as established, and many ideas and concepts have not been turned into repeatable processes. On the other hand, though, software is flexible and infinitely malleable. With the right techniques it should be possible not only to produce exactly what is needed, but it should also be possible to use parts of the software before the whole is finished, and it should be possible to adapt the software to a changing business environment, all of which are would reduce the risk of a large capital expenditure.

From an organisational perspective the real question is even slightly different: whose risk is actually reduced by buying software? In many organisations the department responsible for building or buying software, the IT department, is separate from the department that uses the software, *the business*. The IT department is often measured more on delivering on time and on budget than on the overall cost or potential realised. So, for the IT department there is little to no incentive to build because buying shifts all the risk in areas that matter to the IT department onto the external software vendor.

For the business, however, the situation is quite different. Not only is it possible that custom software may have been cheaper, therefore providing an earlier return on the investment, but it is also possible, and not uncommon, that the software package does not do exactly what the business needs, leading to decreased productivity and lost opportunities. Of course, these concerns should be addressed in the decision making process. Unfortunately, the process is normally run by the IT department, which is set up to care more about different risks. As a result it is possible that the selection and decision process minimises the risk for the IT department but does not optimise the solution for the organisation as a whole.

A good example of this conflict is the introduction of a CRM solution at Australia's largest telco. A system was bought from one of the market leaders, clearly reducing the risk for the IT department, but as a newspaper reported "dealers at the coalface of the telco's retail chain are still saying the clumsy customer and billing platform at the heart of the company's multi-billion IT transformation is costing them business."[1] Minimal risk for the IT department clearly did not translate into an optimal solution for the business.

If the business may really be better served by building, is it possible to take the risk out of building for the IT department? To start with, major breakthroughs in development methods over the past decade have resulted in much more predicable software delivery. Processes built around the agile value system deliver working software in small iterations, not only releasing functionality incrementally, but also giving the business a chance to provide feedback and, if necessary, allowing the developers to adapt the software. This greatly reduces the risk that the software does not meet the needs of the business. Software development teams have gained the ability to react to changes and develop software iteratively through techniques and practices as those described by the Extreme Programming and Continuous Delivery methodologies.

More intriguingly, it is possible to reduce the risk by shifting the goal. What if the main goal was no longer to deliver a large number of function points in a pre-determined time and budget? What if the core goal was to deliver the most needed functionality at any given point in time as efficiently and effectively as possible? Again, agile practices would clearly help, but how crazy is this idea? In 1982 Tom DeMarco opened his hugely influential book on Software Engineering with the line "you can't control what you can't measure". Interestingly, in an IEEE paper in 2009 he writes: "For the past 40 years [...] we've tortured ourselves over our inability to finish a software project on time and on budget. But as I hinted earlier, this never should have been the supreme goal. The more important goal is transformation, creating software that changes the world or that transforms a company or how it does business."[2] I could not have said this any better.

Software development is inefficient

Another common argument against building software is that it takes a lot of time and effort to implement. A business function that can be described on a couple of pages requires several hundred lines of code, which take a developer weeks to write and test, and yet more time to be deployed into a production environment. Complex systems that can only be described on several hundred pages require years to be implemented. One obvious answer would be to live with the inefficiency but at least to reduce the time by putting a larger team to the task. Unfortunately, it has become more than obvious that the speed of software development does not scale with the number of people on the team. At some point the communication and coordination overhead cancels out any further brainpower.

In the same way that buying a package shifts risk to the vendor, it shifts the problem of efficient delivery to the vendor, too. The buyer simply has to run a tender and procurement process and oversee installation of the software package. Moreover, when multiple parties buy the same software the vendor can spread the cost of development across all buyers, thus reducing the impact of the inefficiencies for each buyer.

Is software development really so tedious and inefficient as proponents of buying suggest? And what could be done to make software development more efficient? In his 1987 paper Fred Brooks explains why "there is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity". He adds, though, that "many encouraging innovations are under way" and that "a disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement."[3] I believe that he was right and in the 25 years since he wrote this paper huge gains in developer productivity have been achieved.

Brooks makes an important distinction between essential complexity and accidental complexity. Essential complexity is the complexity of the business problem the system is implementing. It might seem that there is nothing an IT team can do to reduce essential complexity. They cannot and should not change the business. Interestingly, though, in projects with long cycle times (years) there is a tendency for the business to be somewhat speculative and request all functionality that they can think of because of the worry that if they leave something out of the requirements it will not be delivered for a very long time. With prioritised iterative delivery the business can halt a project when all features that are actually needed have been completed. Of course, this does not really reduce essential complexity but it does reduce the amount of features that are implemented, and based on my experience, quite substantially so.

Returning to Brook's distinction, accidental complexity is the complexity caused by the tools and techniques used to implement the system. This is complexity that would not exist without the implementation. As Brooks predicted no order-of-magnitude reductions in accidental complexity have occurred. For example, there is no widespread use of expert systems or graphical programming environments, and it is not for a lack of trying that such tools do not exist. End-user programming, or *automatic programming* as Brooks called it, is not in widespread use either, maybe with the exception of Excel spreadsheets. However, domain specific languages are a promising current development that has the potential to achieve some breakthroughs in this area.

One area that Brooks underestimated are integrated development environments (IDEs). The capabilities a modern IDE provides in areas such as code navigation and automated refactoring have resulted in impressive productivity gains. Automated refactoring in combination with test-driven development have made it possible for developers to achieve high internal software quality with less effort, thus not only increasing the efficiency with which code is written initially but also making the pace of development more sustainable as the codebase grows.

Developer workstations have become substantially faster, allowing for more complex IDEs and tools. Improved compilers, for example, create more optimised code without developers having to worry about details such as register allocations. The major productivity boost in this area comes with the fast internet connection, though. Rather than experimenting or discussing problems on mailing lists and newsgroups, developers can now access online communities that provide answers to many common development issues within minutes, either because previous discussions of the problem are easily searchable and accessible or because peers answer in real-time using internet relay chat (IRC).

The improvements in hardware obviously do not only affect developer workstations but they also impact production environments. In many cases it is now possible for developers to trade some wastefulness of resources at runtime against increased productivity at development time. For example, instead of spending effort on writing code to manage memory, developers can reduce the accidental complexity associated with manual memory management by employing an automatic garbage collector. There is a line of argument that on modern hardware automatic memory management even increases efficiency due to higher locality of reference and better resulting cache use, but even if automatic memory management is slower, the trade-off is worthwhile in all but a few specialised areas.

Lastly, Brooks and many others in the early nineties envisioned a thriving commercial market for components, creating a middle ground between buying complete packages and writing the entire solution. By and large such markets have not been overly successful. There are exceptions, but in general these markets have been superseded but something even better: open source frameworks. Many developers and development organisations have found that sharing the source code for basic building blocks did not harm their organisations, by helping the competition, but instead helped the organisation by gaining free development capacity from other collaborators. A further benefit of frameworks that are created directly by the practitioners is that their feature set is determined by actual needs and not driven by feature checklists. Innovation extends to the public sites on which the frameworks are maintained, enabling ever more flexible and effective ways to share code.

Software development is not our core competency

Over the last 30-40 years software solutions have reached more and more areas of businesses but it is important to remember that they started as tools providing operational efficiency in certain areas. It is maybe that background that has led many organisations to believing that software development is not part of their core competency. Sometimes people refer back to examples from the early twentieth century when factories had their own electricity generation facilities. The analogy is clear: factories switched to electricity off the grid as soon as that was available because generating electricity was not at the core of their business. Their core competency lay elsewhere, in the production of cars for example. So, why should an enterprise that, say, specialises in managing insurance have competency in software development? In fact, with cloud computing the question that is being asked is whether many companies should have any competence in IT, or whether they should move it all into the cloud, much like the factories moved their energy problem to the grid.

This issue is different from the two issues discussed previously. The assessment of risk and efficiency is changing because of advances in the way software is developed and delivered. The question of competency is mostly affected by how the businesses are changing. Software solutions are no longer just tools that make it possible to run a certain business process more efficiently. Software solutions have become enablers of completely new ways of doing business.

About 25 years ago a newspaper benefitted from electronic desktop publishing, but, still, this only made the process of laying out the pages more efficient. There was little reason to write bespoke publishing software. Today, as much of the business has shifted onto the web, the software behind the newspapers' websites has become strategically important to their business. For example, the audience is no longer just the loyal reader of the paper. It is equally the person who arrives at an article via a search engine. If the website can provide easy access to related content, the visitor will stay longer, see more adverts, and therefore increase revenue for the paper. Providing compelling content is still a task for the editors and writers, but making it available at the right time in the right context is a function of the software.

The newspaper example is just one of many. The trend for software to become a strategic enabler for the business can be found in many domains and verticals. Now, if a piece of software is strategic to a business, the result of choosing from the same set of packages as the competition should be obvious: little or no competitive advantage. As discussed before there may be less risk for the IT department by buying a package, but it certainly will make it much harder for the business to get an edge over the competition who can chose from the same packages. Even worse, though, reacting to change or implementing a brilliant new idea will take much longer if a product and an external vendor are involved. Cycle-time, that the time between having an idea and seeing it running in production, can be reduced by having a well-oiled IT function that uses practices and techniques from Agile methods and Continuous Delivery.

It seems that even if software development was not a core competency for most companies it is rapidly becoming strategically important for many to make software development a core competency. This may require some painful changes and a shift in corporate culture, but ignoring new realities has rarely helped either.

Traditional issues with buying

Motivating the decision to buy software by pointing out the problems with building software can be effective, but it implies an important assumption, namely that there are no issues with buying software or that these issues are smaller than the issues with building software. It should be pretty obvious that there are no approaches without issues, especially not in IT. So, what are the problems that organisations have encountered when buying software products? And how big are these problems?

Software packages deliver features that are not needed

Companies that write and sell software packages face an interesting challenge. Usually each of the organisations they are selling to have somewhat different needs and processes. The overlap can be sizeable but even in the best case there will be features that are only required by a subset of the potential customers. At the same time it is normally too costly and complex for the software vendor to produce individual packages for each of their customers. Their business model is based on the fact that the software can be written once and sold many times. Responding to this challenge most vendors choose to implement all features that all of the potential customers may need.

A variation of this theme is software that is split into modules. Large ERP systems for example have modules such as payroll and logistics. In these cases, however, the same challenge repeats at the module level, not every company's payroll requirements are the same. To make matters worse, the customer now has to choose, install and maintain several pieces of software.

In addition to responding to known needs of their customers the software vendors, especially in competitive markets, exert force on each other. They must implement ever more features to differentiate themselves from the competition. At this point features required across most of the customer base are considered *tickets to the game*, features that just have to be in the product to be considered. The differentiating features are often aimed at smaller segments of the audience or more speculative in nature, which normally leads to a very large feature set, and to users that only need a fraction of that feature set. An extreme case is Microsoft Word. A blog post on Microsoft's developer site reports that out of hundreds of commands in Word "five commands account for around 32% of the total command use in Word 2003"[4]. There are varying estimates for business and enterprise software but often the number of features used by a specific customer or user is estimated to be below 10% of the overall feature set.

When all the features needed by the users at a given customer are available one could consider this a success. The users have their requirements met and the software vendor has a model that works for them. Unfortunately, there is a problem: All those features that are present in the software but that are not needed by a given user add overhead for the customer and their users. First and foremost, the features are visible and at best may just take up space in menus, toolbars, etc but at worst may confuse the users or make use of required features more complicated. This often translates to lost productivity and increased training effort. The additional features are also likely to result in a higher resource usage, such as memory and storage space as well as network utilisation, which translates into higher capital expenditure for hardware and increased operational costs.

Software packages require customisation

One way to address the issues outlined in the previous section is to customise the software so that unneeded features are not visible or not even installed. This way, users are not burdened with these features and the features do not use additional resources. However, any customisation of packaged software brings with it the problems discussed in the context of developing software. Even if it is to a smaller degree, customisation brings risk in that unforeseen issues can cause delays or increase cost, they take time to carry out and test, thus increasing the time to introduce the package, and customisations require competencies that are not core to the business that is acquiring the package.

In most cases customisations are needed beyond removing unneeded features. Business, even those that operate in the same market, providing the same services to their customers, usually employ somewhat different internal processes and practices, and obviously the software package needs to support these. Some packages basically store data and make it accessible at a later stage. In such cases customisation is limited to adjusting the structure of the data and the screens to access it, which is normally relatively painless. More and more often, though, software packages actively support the workflow of an organisation and in such cases customisations quickly become more complex.

Adjusting workflows and processes in a software package exposes the customer of the software package to several equally unappealing options: They can change the actual source code of the package, thus effectively forking the codebase. Apart from the upgrade issues discussed further down, this brings buying software even closer to developing it in the first place, with all the issues discussed before.

Another option is to choose a software package that allows customisation through rules engines and workflow engines. The hope is that following a formal business process modelling exercise, producing a clear description of the business processes in a standard notation such as BPMN, the workflow engine can simply use that model without the need to change any code. In practice, however, it turns out that formally modelling the business processes is actually hard. In fact, many of the problems and risks associated with developing software are not caused by the actual programming but by getting the requirements right, and that is closely tied to understanding the business processes at a detailed level. Furthermore, capturing complex processes in a graphical notation adds its own set of problems. Even with sophisticated tooling navigating and manipulating large diagrams can be more cumbersome that working with a textual representation such as code. Features that facilitate collaboration between several modellers, such as version control involving merges and conflict resolution, are nowhere near as mature in workflow tools than in those tools available to manage source code.

A middle ground between source code and graphical notations are textual configurations and domain specific languages. In the end, they address some of the concerns, but still do not resolve the fundamental issue that any customisation adds risk and complexity.

Now the key question is: How much risk and complexity do customisations add? Unsurprisingly, there is no simple answer. Sometimes customisations are small and negligible in the context of the introduction of the software package. In many cases, though, the customisation can be as big or bigger than entire software development projects. Colleagues of mine have reported instances of customisation code alone exceeding 100,000 lines and I have seen a system in the financial services industry that comprised more than a million lines of XML code describing how to handle so called *corporate actions*. To be fair, this was not based on a software package for corporate actions, if such a thing even exists, but on a more general processing engine package.

An interesting approach to avoid the issue entirely was pointed out by my colleague Martin Fowler. He suggests that "you should buy a package and adjust your business process to fit the way the package works"[5]. As mentioned in Martin's article and earlier in this one, an organisation should not use software packages in areas that are a strategic differentiator for them. In consequence, if packages are used only in areas that are not strategic to them then it does become possible to change the process to match the software, thus avoiding the need for customisation. This is especially true for smaller and medium sized businesses, who in fact may benefit from improved processes enshrined in the software package.

Some companies, of course, choose to use packages in areas that are strategic to them and even then there are examples of companies eschewing customisation. In one case an entire mobile phone network was set up with packages from a large software vendor and the network's CIO "insisted in the set up [...] that there would be no customisation" and "he believes he bought in best practice methods [...] and teams around the business should, and have, aligned their methods with the applications."[6]

Software packages bring their own roadmap

Yet another dimension to be considered is the evolution of the software package. With the exception of cases such as software to handle taxation in a specific year almost all software packages evolve over time as the software vendor responds to feedback from their customers and their evolving needs. New versions may be released as infrequently as once a year or they could come out at short and regular intervals.

Each update brings incremental change and the effort required to carry out the upgrade mounts with each update the customer does not install right away. Therefore the question for most customers is not so much whether to upgrade but when to upgrade. The vendor's product roadmap now needs to be taken into account when planning the internal roadmap and work required to upgrade needs to be scheduled. Depending on how much customisation was required to adapt the package the effort to upgrade can be significant.

One way to reduce the effort is to reduce customisation as described before. I have heard of cases where customers decided against getting access to the source code of a package, even though that was offered for free and would have allowed them to make some necessary customisation more easily. They did so because they knew that by changing the code they would effectively be forking the vendor's codebase and would have to reapply all their changes with each new version. Instead the customers, or more precisely their IT departments, decided to live with workarounds that inconvenienced their users. This is another example of conflicting interests of the IT department and the business leading to a worse outcome for the users. However, in this case it is not just risk but actual effort that the IT department is avoiding, and it can make good business sense to do so.

An extreme solution to the upgrade issues is not to upgrade the software package at all. The customer buys a version of the software that works for them, customises it to fit its needs, and then stays with their customised version indefinitely. Generally, this not an option, though, because updates often include fixes for security problems and with the degree with which software is networked unfixed security problems are not tolerable. If the customer is lucky the package vendor provides security fixes as patches that can be applied independent of upgrades, but even then, at some point the current version of the package will have changed so far from the version the customer is clinging on to, that the patches are no longer directly applicable.

Lastly, over longer periods of time the ideas a package vendor has for its product may diverge from the needs of a given customer. The outcome in this scenario very much depends on the relative size and power of the two companies. If the package vendor is much bigger than the customer then the customer has no choice but to either move into the new direction and adapt to the changes or they have to procure and install a different package and absorb the cost of doing so. If the customer is a very large enterprise and the package vendor is somewhat dependent on them, then the customer can impact the direction into which the product is evolving. They may ruin the chances for the package vendor in the long term, though. In either case, there is significant risk or cost for the customer, something that buying software is meant to reduce in comparison to developing software.

A new issue with buying

The issues discussed above are issues that organisations have had to to deal with when buying software for a long time. Now, advances in IT and corresponding changes in the business have introduced a new issue with buying software.

Software packages struggle with service orientation

Traditionally software packages were mostly applications that solved a specific need in the business, such as the stereotypical example of a payroll system or software to calculate risk and premiums for an insurance. The applications were often stand-alone system with little integration between them. Following a growing need for more detailed reporting, and enabled through better database technology, the 1990s saw the widespread adoption of data warehouses. New, tool-supported extract transform load (ETL) processes took data from several operational systems and placed it into a central data warehouse where it could be aggregated and analysed. Despite the ability to view data across the organisation most IT landscapes were still very much application-oriented.

Reporting was only one step as businesses looked for ever tighter integration of their processes and business units. In areas that relied heavily on batch-processing, often run during the night, or on manual re-keying, sometimes mocked as swivel-chair integration, the expectation was that data should be processed *straight-through*, that is it should flow from one boundary of the IT systems landscape to the other immediately and without manual intervention. This desire led to a widespread move from application-oriented landscapes to service-oriented architectures (SOA). In the most successful cases existing line of business applications were decomposed into a portfolio of services, resulting in tighter integration, more consistency and less duplication. Whether a service-oriented architecture is generally desirable is not the topic of this article. It is clear though that many businesses and enterprises, maybe even a majority of those who rely on IT solutions, are moving towards them. With this change new challenges for software packages arise.

Even without a fully developed SOA a new software package must be integrated into a complex IT landscape, a landscape that may have grown over decades. In such cases configuration and customisation, with their associated effort and risk, are joined by integration work, the effort of which is rarely negligible. Worse, though, integration works usually carries a lot of risk *and* that risk is hard to mitigate because the integration effort involves other software systems.

Today's large ERP packages, for example, have about 3,000 interface points, usually in the form of web service endpoints and operations. These operations are defined by the package vendor, which means that they are unlikely to match exactly the requirements and standards of the systems they are to be integrated with. In fact, they are often thin shims over internal data structures. All this means that the package does not only have to be configured and customised but that specific integrations must be written. Sometimes the impression is created, usually with reference to standards such as SOAP and WS-*, that systems can simply be plugged together by drawing lines in graphical tools. Unfortunately, though, in the real world this is not normally possible. The standards, even if fully interoperable implementations are found, are too low level and do not cover enough of the semantics of the integration. From personal experience I can also attest that even major implementations of seemingly basic standards such as WS-Security are not always complete or fully interoperable.

As an organisation moves from siloed applications to a service portfolio the integration effort and risk in relation to the overall effort and risk increase. Not only do key benefits of buying packages erode, but because the packages by definition have more rigid integration points than custom developed software, they also make this growing area of effort and risk more difficult and more risky.

There are two solutions to deal with the integration problem. One approach is to procure all packages from a single vendor, an approach that is sometimes called *best-of-suite*, I guess not for linguistic accuracy but to contrast it with best-of-breed. In a way this is the logical conclusion of the idea to shift risk from the buyer of software to the vendor at the possible expense of features, fit or upfront cost. Interestingly, many of the large suites sold by software vendors are the result of mergers and acquisitions, which means that the integration work remained necessary but it really did move from the buyer to the vendor. The mobile network example cited above is an example of this strategy.

Another approach to mitigate integration risk, when following a best-of-breed strategy or when integrating bespoke services, is to buy the integration itself in the form of software products such as an enterprise service bus (ESB) or even a complete business process integration suite. The theory is that the integration product removes the burden of having to write integration code and makes integration just a configuration task, which is obviously the same idea as buying and configuring software packages. In practice, however, and certainly not for a lack of trying, such integration products usually do not deliver the benefits expected. As discussed above in the context of graphical modelling tools, the problem is that the tool attacks what is easy, namely the mechanics of the integration, but provides little to no support to tackle what is hard, the semantics of the integration. If two fields in two XML documents can be connected easily in a graphical tool then even in the easiest cases somebody has to find out whether account numbers are zero-padded, whether currency symbols, and which, are included with monetary values, whether a boolean field contains zero and one, or yes and no, or true and false, etc. That combined with the additional effort to master the tool usually by far exceed the effort to connect two custom-developed software components.

Making the shift

In the article so far I have discussed issues with both, building and buying software. As I explained, I believe that the improvements in software development tools, techniques, and practices have reduced the issues with building software significantly while existing issues with buying remain unchanged and the trend towards deeper integration and service-oriented architectures has caused new issues with buying software. If an organisation agrees with this assessment and plans to make the shift towards building more software, what are their options?

One Ring Out

An obvious approach for an organisation to build its own software is to embrace software development as an important competency. The core competencies remain at the centre of the organisation's activities. In the newspaper example used earlier the core competency is the work of the journalists who report and comment on stories. Many other competencies that are important but not a differentiator, invoicing the newspaper subscribers for example or purchasing paper for the printing press, remain further out from the centre. In this picture, with the core competencies at the centre surrounded by the other competencies, software development should be viewed as a ring around the core competencies. It is not at the core, but it is close to it and because of its enabling nature it is deeply intertwined with the core competencies.

The mantra to "focus on your core competencies" is one of the main obstacles for many companies to create that ring of software development competency. However, like the "buy when you can" mantra it is outdated. Many companies that were successful over the last decade were successful because they left this mantra behind and explored the ring, or even further rings, around their core competencies. Google did not just focus on their core competency, arguably search and advertising, and outsource the operation of their server infrastructure, making it a procurement problem for servers and rack space. Instead, Google constructed their own servers and made data centre design one of their competencies, even becoming leaders in the field with high ambient temperature centres[7]. They did not buy, they built. Similarly, Amazon, at the heart a merchant selling books and consumer goods, did not buy an e-commerce package, they did not even outsource server operation. Instead, they built their own software and infrastructure, in the process becoming a provider of cloud computing services. Finally, Apple, now basically a consumer electronics company, is increasingly developing competency in processor and system on a chip (SoC) design. Even though these are components that traditional wisdom suggests one should buy from specialists, Apple decides to grow its competencies and build.

In my experience, the hard problem is to overcome this self-imposed restriction on the existing core competencies. Once it is accepted that software development, the practices, the people, and to a certain extent the culture around it, should become part of an organisation the hardest part of the shift is done.

As with any organisational change, though, it is key to have a sense of urgency to get started and to have some quick wins to make the change stick. If nobody senior truly believes that software development is an essential competency the introduction of a software development group will fail. If everything is pinned on a three-year plan, which will only deliver benefits in the third year, the introduction will fail. The organisational setup should reflect the importance, too. A software development group should not be put under the oversight of a powerless task force. It should neither be put in an existing department that itself is not at the core of the business. There are, for example, many stories of retail companies who created a website team in the marketing department, and then wondered why they struggled to make the shift towards e-commerce. The approach to create a sense of urgency will vary from organisation to organisation, but the establishment of a software development capability should always be well aligned with a core business unit, ideally the teams should be co-located, and the new group should deliver value in small increments as soon as possible.

Avoiding the Alignment Trap

As an aside it is worth mentioning the findings presented in a report titled "Avoiding the Alignment Trap in Information Technology"[8]. Based on data from about 500 companies the authors look at the paths a company can take to create a well-oiled IT function that is aligned with the business. There are clearly two dimensions to the problem, the ability to execute and deliver software on the one hand and the alignment of projects with the business on the other. An organisation that starts with a non-existent or below-average capability has to make a choice on which to focus first.

In the study, companies that did nothing, or not much, made up about three-quarters of the sample. They saw slightly below-average growth and about average levels of IT spent. Companies who focussed on execution, likely improving their competency in software development, were able to achieve revenue growth of 11% above average over three years while at the same time spending 15% less on IT than the other companies in the study. The fact that these companies were able to improve their results, and that even without alignment to business objectives, demonstrates the importance of having competency in IT.

The interesting and maybe surprising finding, though, is that companies that pursued business alignment first, without addressing execution, spent 13% above average and had to contend with 15% below average revenue growth. Based on these results, alignment with the business is not enough. In fact, the study suggests that an effort to align business and IT is harmful unless the organisation has first managed to create an effective IT function. Once the competency is there, though, alignment brings companies into the *magic quadrant*, with costs slightly below the average and revenue growth of 35% above their peers. These are the companies that have managed to create a ring of IT competency around their core competencies.

Buying building services

With the will to create competency in software development, with the understanding that this might shift corporate culture, and with a focus on execution, many organisations will be able to attract people with the necessary skills and organise them into successful teams. Of course, people will bring experience, but another development helps here, too. Many successful companies have realised that being relatively open about their IT competencies and achievements is a huge benefit to their employer brand, their attractiveness as an employer, and by becoming active participants in the wider software development community their teams can learn from other innovators. Even for companies that do not participate directly, this has resulted in an abundance of case-studies and other material to learn from, available through software conferences, books, and online discussions.

That being said, there is a third way: neither buying software nor building it, an organisation can buy services from another company to build the software for them. I should state that the company I have been with for over a decade is such a software consultancy and I am therefore both knowledgeable about the topic and at the same time somewhat biased.

So, instead of creating its own software development competency the organisation finds a partner, a software consultancy that has the necessary competency to build the software. The partner operates on a professional services business model, billing for the effort, for the actual development competency provided. Deals in which a fixed price is agreed for a piece of software are common, too, but that is conceptually closer to actually buying software, and not services. The consultancy can maintain and grow strong development competency when it has multiple clients and it can rotate their consultants through projects at different clients. In each project the consultants will provide some of their past experience while learning something new, which might be useful for the next client. Because the consultants are not fully integrated into the client's organisation their natural tendency is towards execution, focussing on delivery excellence, which can be helpful in avoiding the alignment trap described above. Another benefit of having a complete team from an external partner, over hiring and building a team out of individuals, is the shortening of the *forming and storming* process that occurs with new teams. The price to pay is a dependency on an external partner.

In my experience, the most productive shifts towards software development occur when the organisation that requires the competency works with a partner, but instead of out-sourcing the entire delivery it takes an active part in the development process. Initially, most of the development team is from the external partner, they set up the project using practices based on their experience, bringing a bit of new culture and confidence that the process will work. Over time, the organisation hires or transfers existing employees into the team replacing the external consultants one by one until after a year or two the project is fully under the control of the organisation. This approach may, or may not, be more costly than building a team from scratch, but, again in my experience, it does speed up the creation of a software development group while at the same time reducing risk that the affected projects fail.

Conclusion

Hopefully I have conveyed why I believe that software development has become more attractive, why buying packages is becoming less attractive, and that acquiring competency in software development is within reach of most organisations and can be a huge benefit for them. With all this it is not too much of a stretch to expect that we will see more and more bespoke software development in the future.

References

[1] Mitchell Bingemann, "Dealers still fuming at 'clumsy' Telstra system". The Australian, 2 June 2009. http://www.theaustralian.com.au/news/dealers-still-fuming-at-clumsy-telstra-system/story-e6frgal6-1225719947511

[2] Tom DeMarco, "Software Engineering: An Idea Whose Time Has Come and Gone?". IEEE Software, July/August 2009 http://www.computer.org/portal/web/computingnow/0709/whatsnew/software-r

[3] Frederick Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering". Computer, April 1987 http://www.cs.nott.ac.uk/~cah/G51ISS/Documents/NoSilverBullet.html

[4] Jensen Harris, "No Distaste for Paste (Why the UI, Part 7)". MSDN Blogs, 7 April 2006 http://blogs.msdn.com/b/jensenh/archive/2006/04/07/570798.aspx

[5] Martin Fowler, "Package Customization". 6 July 2011 http://martinfowler.com/bliki/PackageCustomization.html

[6] "Business must align with IT", CIO UK http://www.cio.co.uk/debate/119053/business-must-align-with-it-says-vodafone-qatar-cio/

[7] Steven Levy, "Google Throws Open Doors to Its Top-Secret Data Center". Wired, 17 Oct 2012 http://www.wired.com/wiredenterprise/2012/10/ff-inside-google-data-center/all/

[8] David Schpilberg, Steve Berez, Rudy Puryear and Sachin Shah "Avoiding the Alignment Trap in Information Technology". MIT Sloan Management Review, Oct 2007 http://sloanreview.mit.edu/the-magazine/2007-fall/49102/avoiding-the-alignment-trap-in-it/