Realising Bespoke Off-the-shelf Software
White Paper

Bespoke software is developed to meet the specific needs of an organisation or an individual. Off-the-shelf software, on the other hand, is ready-made to address the perceived needs of a particular market. Each has its strengths and weaknesses from the perspective of procurement, maintenance, support and use: as a result consumers will often have to accept compromises.

The ideal software model is a hybrid – "bespoke off-the-shelf" (BOTS) – but this seems like a contradiction in terms. How can software be both off-the-shelf and bespoke?

Before considering the essence of BOTS, though, we should understand the broad categories into which organisational software falls, the pros and cons of bespoke software and the dilemma facing providers of off-the-shelf software.

Organisational software

From the perspective of organisations, a software product generally falls into one of three categories: *common*, *operational* and *methodological*.

Common

Most organisations will rely on mass-market products like spreadsheets, word-processing packages and e-mail clients to fulfil common, fundamental activities. Such one-size-fits-all software is typically inexpensive and does not provide the organisation with competitive advantage: it merely supports the administrative needs of an organisation's staff.

Operational

Mid- and large-tier organisations in particular will commonly make use of packaged software, such as ERP and CRM solutions, in support of their operational needs. These solutions tend to be modular or configurable in order to meet a broader spread of client requirements and integration scenarios. They are also regarded as essential components of information management and automation, yet they do not typically confer competitive advantage. Organisations may be prepared to adjust their own working practices in order to accommodate package limitations because there are no cost-effective alternatives.

Methodological

Many organisations will need tailored software solutions either to deliver competitive advantage or to meet highly specific business needs. Niche off-the-shelf package providers may exist whose products can meet these needs – at least partially – or an organisation may choose to commission bespoke software if the perception is that this is the most cost-effective approach. Rarely, though, will an organisation readily compromise its own methodologies in order to accommodate the intransigence of a package provider or the shortcomings of its products. That would be a case of the

tail wagging the dog. Nonetheless, organisations are commonly faced with this situation and may end up accepting that they must find ways of working around such issues.

The pros and cons of bespoke software

The primary benefits of bespoke software are that it meets specific requirements and fits exactly with the way in which an organisation operates. When addressing methodological requirements, bespoke software has the potential to deliver significant competitive advantages.

However, the main drawbacks are those of cost and delivery time. The cost of bespoke software is usually significantly greater than that of a broadly equivalent off-the-shelf solution; and, depending on complexity, the time taken to implement and deliver a bespoke solution can vastly outstrip the time taken to evaluate, procure and integrate an off-the-shelf counterpart. What is more, off-the-shelf products are invariably tried and tested, have committed maintenance and upgrade plans, and are accompanied by both a substantial support function and a community knowledge base.

The off-the-shelf dilemma

Of the many problems facing providers of off-the-shelf software packages, a fundamental one is this: how complex should a solution be?

Relatively simple, generic solutions offering few capabilities are attractive to providers because, in general, they require less maintenance and support. However, when targeting a niche market it is unlikely that such watered-down solutions will gain any traction. This is partly because simple niche solutions add marginal benefit and partly because competing providers can easily offer equivalent functionality.

Relatively advanced solutions typically have a wider gamut of industry-specific capabilities and are functionally rich. They may offer proprietary techniques that create market differentiation for the provider and higher value for clients. These solutions will tend to be predicated on the specifics of a given market, though, and so the potential impact of legislative or practice-led change is far greater: providers will be forced to update their products or face the consequences. Furthermore, in spite of the additional costs incurred in developing and testing these solutions, there is still the risk that potential clients' needs will not be sufficiently well met to merit an investment.

Providers tend to tackle this problem by designing *modular systems* and/or incorporating *parameter-driven behaviour* into their solutions.

Modular systems

A modular system has, at its heart, a mandatory core component to which optional modules may be attached. Each such module is capable of performing a discrete task and the potential client is thus free to buy and configure only those modules that are perceived to be of relevance to his business. This, in turn, means that clients do not waste money on functionality they do not need. Nonetheless, "module" is a synonym for "software library" and, as such, a module's capabilities are still baked in: the client will get the whole module – warts and all – or nothing.

Parameter-driven behaviour

A complementary variant of modular architecture is parameter-driven behaviour. Here, the client provides values for multiple parameters (the administrator might do this during system setup or a user might do this as and when necessary) and subsequently invokes an operation that uses these parameter values to control aspects of its behaviour. This approach is often used by providers who wish to include custom capabilities in their products but there will always be a limit on its flexibility because the operation itself is mandated in software: in other words, it's baked in.

BOTS software

How can an organisation accrue the benefits of both tailored and packaged software in a single solution? Bespoke off-the-shelf software is, first and foremost, off-the-shelf software. Importantly, though, it has all the hooks necessary for clients to define their own operations and to define them *in their entirety*.

On the face of it, this might sound like an easy win for a package provider: conceive some operation, define its scope, define its inputs and outputs, provide some means of invoking the operation and, when the time comes, delegate control to third-party software. However, there are two points of note.

1. An operation scoped by a package provider is not an operation scoped by the client. A true BOTS solution will allow end-users both to scope operations and define these operations' behaviours.

2. This approach is no different to the modular strategy described earlier and it ignores one immutable fact: clients are not software engineers. They have neither the skills nor the desire to implement software. And if they did, the chances are that they would not need another provider to deliver a tailored application.

Instead, what is needed is a tool that affords clients the ability to develop software without the need for traditional software engineering skills and integrates with third-party applications to facilitate the invocation of such software. This is where Absyntax can help.

Introducing Absyntax

Absyntax is an industry-agnostic desktop framework application that bridges the gulf between the need for bespoke software and the ability to deliver it. It helps people without traditional programming skills to develop software that may be executed both stand-alone and via third-party applications, and at the same time sidestep the need to understand the myriad of jargon that accompanies the software industry.

In the context of application integration, Absyntax is particularly suited to handling volatile operations; for example, those that are either sensitive to market change or specific to a particular client and which are thus likely to require more frequent updates. By delegating such operations to Absyntax, the core application will require fewer modifications.

Developing software with Absyntax is a visual process based on a very simple paradigm:

- select blocks of pre-defined functionality;
- configure where necessary;
- combine these functional blocks using connections.

Out of the box, Absyntax defines many granular, reusable functional blocks (called "features") which, when selected and combined in manifold ways, allow users to realise an unlimited variety of operations. In addition, third-party software specialists can extend the framework by developing new features or exposing selected capabilities of their own software libraries in order to make common, industry- or client-specific tasks more readily consumable by Absyntax users.

Package provider options

The ability to extend the framework is particularly relevant to BOTS software because it increases the range of integration options for package providers.

1. Integrate with Absyntax to implement volatile operations that in-house business analysts or other specialists can create and maintain. Such operations would not typically be exposed to clients but their implementation as Absyntax projects will protect the core application from change.

2. Integrate with Absyntax to expose well-defined operations that may be populated and maintained by clients.

3. Offer application programming interfaces (APIs) that facilitate programmatic access to data and core functions; then create and ship Absyntax features whose purpose is to communicate with the packaged solution via these APIs. Clients will then have a focused Absyntax feature toolkit enabling them to scope and define their own operations in true BOTS style. With this option, providers do not need to integrate with Absyntax.

Summary of benefits

Accessible to all

Absyntax enables a wide range of people – not just expensive specialists – to effect software change. This means that the dependency of an individual or an organisation on traditional software engineering skills is reduced. In turn, this allows the creation and modification of software to be realised more quickly and more cheaply.

Third-party integration

Absyntax can be integrated with third-party applications, enabling these applications to invoke Absyntax projects programmatically. This is of particular benefit to package providers whose solutions cannot fully meet all of their various clients' needs. By working in tandem with Absyntax, such solutions can support client-specific functionality while avoiding the need for multiple product versions. Even modular solutions with parameter-driven operations are, ultimately, limited. With Absyntax, though, such limitations can be bypassed, providing clients with the capabilities they would expect from bespoke solutions.

Manage volatile operations

Third-party applications require periodic modification, and not only to introduce enhancements or fix bugs. Changing business requirements, legislative impacts and improved analytical methods are just a few of the reasons why further software development might be needed. Some software-driven operations are simply more volatile than others: Absyntax can be used to handle such operations, affording business analysts and other industry specialists the opportunity to effect changes. This softens the impact of such changes on a package provider's organisation and offers a more responsive, cost-effective service to clients.

Harness APIs

Many software solutions offer an API allowing software developers to tap into the capabilities of these solutions using programmatic means. But why should such capabilities be accessible only to those with software engineering skills? Absyntax makes it possible to expose APIs to a much wider audience.

Visualise your software

Absyntax allows software to be visualised. Discrete, low-level operations can be aggregated for presentational purposes, so you don't need to produce diagrams separately using some other tool.

Extend the framework

Absyntax is extensible. This means that software developers can author their own features, which is of particular benefit when needing to provide higher-level reusable operations to a specific industry. Such custom features typically obviate the need for clients to create equivalent functionality using perhaps dozens of out-of-the-box Absyntax features. Also, developers can integrate existing software components with Absyntax using very little code. This means that Absyntax can harness the power of existing third-party code libraries without the need for major redevelopment.

Conclusion

If you are a provider of off-the-shelf solutions, consider integrating with Absyntax to:

- insulate your solutions from the effects of changes beyond your control;
- provide your clients with the tailored capabilities they demand;
- offer to your clients a quicker, more cost-effective way of implementing changes.

If you are a consumer of off-the-shelf software, frustrated at its shortcomings and the workarounds you have to juggle, consider:

- using Absyntax to extend the software's capabilities;
- appealing to your provider to embrace Absyntax – or finding a provider that already does.

Download the free, fully capable trial version from www.absyntax.com.

Contact us at support@absyntax.com.