Application Modernisation

KFA Management Report - Application Modernisation

The modernisation of applications has been an important topic for many years. The technology which our industry is founded on moves quickly – but the applications that are developed to make use of that technology rarely, if ever, keep pace. Out of date applications can be inflexible, difficult to change and costly to maintain – but are often critical to business.

Risks and Benefits

Risks

Any change to systems brings with it a risk – if the new code malfunctions or introduces errors the business can suffer both financial losses and reputational damage. Mitigating that risk requires diligent testing which itself comes at a financial cost. Errors may not solely be system errors – changes to systems can indirectly cause human error as users get to grips with new interfaces or ways of working.

Equally, inaction and stagnation can pose just as severe a risk to businesses. If your competitor is selling their product through mobile devices and you are not – your business is likely to lose market share rapidly. If you have systems that are not adequately maintained they become difficult to support, costly to enhance and reduce the ability of the business to adapt to changing market conditions, to do business through new channels or to release new products. Knowledge of the business through its systems is lost – either through staff attrition and turnover or simply forgotten through lack of use.

Benefits

An application that has been modernised is more flexible, easier to enhance and responsive to business demands. With mobile device support, staff no longer need to be at their desk to access applications or data – they can be wherever they need to be with access to the information and functions, whether that's checking stock in the warehouse or placing orders when visiting a customer's site.

Staff satisfaction is increased if the interfaces are modern, powerful and assist them in completing their work – and with more and more members of staff in the workforce accustomed to modern digital devices that means that the expectations of those staff are higher than ever before.

Steps towards modernisation

Analyse the existing estate

Countless modernisation projects have failed because of a failure to understand the existing application's functionality or assumptions that have been made about it. Bespoke applications that have been built over time to meet the changing needs of the business can often perform functions far removed from the perceived role of the system. The first step in any modernisation project needs to be to develop a clear and comprehensive understanding of the existing estate and the role of the application(s) within it.

Respect your investment

Too many organisations view IT as an expense and not the investment that it is. Bespoke software is much more expensive than off the shelf packages - but the targeted deployment of such applications creates competitive advantage and a knowledge model held in the business logic of the code. A software product is the embodiment of the intellectual property of the company that created it. It is the bathing baby that should not be discarded when the water drains away.

Identify the problem

All too often modernisation projects are a solution looking for a problem. Before deciding that the modernisation of your application is the solution, first document the problems that you perceive that the existing system is causing the business - or is unable to solve. Only then can it reasonably be determined that the system is in need of modernisation - and more importantly what kind of modernisation is required.

Set clearly defined, tangible, measurable goals

The adage that "if you can't measure it, you can't manage it" has some merit in the context of modernisation projects. Whilst there may be intangible benefits resulting from the modernisation of an application, if you cannot set goals that are clearly defined with tangible results that can be measured then the project is at huge risk. Such projects have a tendency towards mission creep, overspending and running out of time. Where a project is instigated as the solution to a clearly identified problem, rather than putting the cart before the horse, this should be straightforward.

Select a modernisation strategy

Once you know what the problem you wish to solve is, that the existing application is all or part of that problem and that the modernisation of all or part of that application is the solution - then you are in a position to design a modernisation strategy which is appropriate for your needs.

There are many different approaches that can be taken to the modernisation of all or part of an application and some of the most common are listed below as modernisation patterns. The strategy you choose will likely comprise one or more of these patterns and will depend on a number of factors individual to your own circumstances: the specific needs of your project; the problems you are trying to solve; the technologies involved; the budget available; the risk your business is willing to bear…

In our experience, regular, incremental change is the least disruptive way to modernize systems, it's the most manageable way to undertake the task, it reduces risk and should always be preferred if possible.

Modernisation Patterns

There are many different ways to make your applications fit the changing needs of your business. These can be broadly grouped into patterns – these are not always exclusive; some applications may be modernised using combinations of more than one approach.

Retirement

The application's function has become redundant and the system can be simply decommissioned.

Consolidate

The application's function is still required, but can be fulfilled by another application. The original application can then be retired.

Refactor

The design of the existing code is improved incrementally. Monolithic programs are reduced in size and scope to create smaller modules of more cohesive, more easily testable code. Redundant code is removed. This may be required to facilitate a wrappering solution or enable new interfaces to the code to be developed.

Migrate or 'lift and shift'

The software still meets the business needs, but the hardware or OS platform is not fit for purpose. The application code is moved to a new, compatible platform as-is.

Reface

The application remains broadly the same but is accessed via a user interface that has been enhanced, usually with the assistance of tooling (such as IBM Host Access Transformation Services/WebFacing), to have more modern features.

Wrapper

A new interface abstraction layer is added around the application to enable integration with new channels or the creation of a new user interface delivered through the browser, mobile applications – or both. This modernisation technique is frequently combined with refactoring of applications in order to expose data access or business logic as services.

Rewrite

The function that the application performs is required, but either nothing about the current implementation is considered to have continuing value, refactoring will be so fundamental and extensive as to be prohibitively expensive – or refactoring will be so costly that a full rewrite would be a better economic choice. The application is completely recoded from scratch into a new target architecture. Note that a rewrite can still be performed incrementally in many cases – moving functionality from the old application to the rewritten version in phases.

Replace

The function that the application performs is required but can be completely replicated by an off the shelf product. The product is deployed in place of the existing application.

Characteristics of Modern IBM i applications

Historically, the computational cost of activating programs was high. As a result, the best practice was to do this as little as possible, which meant that all of the logic necessary to perform a given task was included in a single program object. Programs with thousands and thousands of lines of code were not uncommon. The database access, business logic and presentation code were tightly coupled into a single executable. Logic would be reused by copying it into other programs meaning that a change to the logic would have to replicated wherever it had been copied.

Thanks to IBM’s policy of maintaining full backwards compatibility in every release of IBM i, many of these programs are still in service today. Whilst this underlines the safety of investment in the platform, the modern IBM i is able to support very different ways of architecting applications in order to meet the needs of the modern business and IT environment.

Built in smaller blocks

Modern applications are comprised of smaller, highly cohesive, loosely coupled functions. A function has a clearly defined task which is reflected in its name. In IBM i applications this means that code is written using the full capabilities of the ILE environment. Procedures are written into service programs, with program objects themselves acting as workflow controllers.

Underpinned by SQL

Modern IBM i applications use SQL to both define and access data. The days of defining databases with DDS are – or at least should be – gone. The modern DB2 database has vast capabilities which can only be utilised when databases are defined with SQL.

Using SQL to access data not only improves performance and adds functionality – it also removes the dreaded “Level Check” error and the need to recompile programs when tables change.

At V7R3, RPG programs can pass SQL results sets between them meaning that the same data access program can easily be consumed both as a native RPG program and a stored procedure.

Loosely coupled

Unlike most languages, RPG has native access to both the database and user interface available without requiring the use of a library or API – JDBC is required for Java to access databases for example. This gives RPG many advantages in performance and function –at the expense of good design. The historic RPG program is likely to tightly couple the business logic to both the database and the presentation layer – be that a 5250 terminal display or a printed report. That valuable business logic cannot then be reused elsewhere without copy and pasting it into the source and cannot be made available to other systems. A modern application design divides the application into different tiers with distinct responsibilities – e.g. presentation, data access, business logic, process control are common tiers or layers. Calls are made between the tiers, but they do not access the resources that the other tiers manage directly – presentation layer code will not directly access the database and vice versa.

Tested automatically

When code is written in smaller, discrete blocks with well-defined inputs and outputs it then becomes much easier to test than the monolithic programs of the past. Unit tests of these functions can be automated which enables suites of regression tests to be performed whenever code is updated to ensure that software continues to function as expected.

Coded verbosely

RPG is now fully free format with the ability to have long variable names, procedure names and database column names – all of which mean that code can be written in syntactically expressive ways which bring it ever closer to natural language and make software more and more self-documenting.

Built automatically

The disadvantage of having smaller, discrete blocks of code is that the build process becomes more complex. In the IBM i sphere, a suite of programs may have dependencies on multiple service programs and these may need to be compiled and linked in a particular order. Assistance is needed to ensure that the software is built in the same way each time, with the same settings and the necessary dependencies fulfilled. Automated builds with tool support help manage this process and ensure that errors do not arise as a result of missing dependencies or incorrect compilation settings.

Founded on open source

IBM has taken huge strategic leaps to enable more and more open source code to run on the IBM i platform. Languages like PHP, Python, Ruby and Node.js all now have native implementations on the server – and it's been made easier and easier to compile and build open source projects to run on IBM i. With the Java 8 JVM that also enables the use of any language that will run in the JVM, technologies like Scala, Kotlin, Clojure et al can be leveraged on the modern IBM i server. In the front end, whether it's a web application using a CSS framework like Bootstrap or LESS, JavaScript frameworks like React.js, AngularJS or JQuery, or a mobile device framework like Ionic – open source software is enabling us to deliver the most modern of applications to business.

Integrated with API interfaces

Our key business functions such as creating orders, generating invoices or registering customer accounts need to be accessed from all sorts of different channels. We'll have the main user interface to the application of course – and then a website, possibly mobile devices; the business may sell its products through third party auction sites or portals like eBay and Amazon – but we don't want to reproduce the business logic to create and fulfil our orders for each channel.

By coding the business logic in a discrete unit that is not coupled to any given interface we can then expose it via an API that these interfaces can consume – so no matter where the order comes from or how the customer registers with us the same logic will always be used and when the time comes to maintain that code it's only done in one place – whether it's called from a local user, initiated via messaging middleware or a web service.

Developed with modern tools

The old tools for IBM i application development are no longer up to the job and IBM have stopped maintaining them. When code is written in smaller blocks, with expressive free-format syntax, a modern development environment is needed to allow developers to see more of the source code at once, to quickly read, understand and move between source files and assist them in being more and more productive. The modern RPG application is developed in Rational Developer for i – which also allows tooling to be added on for many of

the other languages that are now available on the system. A full-stack developer can code both an RPG back-end and the front end of an application in the same toolkit – whether that front end is Java, PHP, AngularJS or pretty much any technology they need to use.

Presented with a modern user interface

Whether delivered via the browser or mobile devices, the workforce is changing and maintaining staff satisfaction and productivity requires a modern interface to our business applications. Whilst the presentation code for this executes locally on the user's device, the IBM i can deliver this code to them and provide the same high-performance, highly scalable, ultra-reliable back-end service as this platform and its predecessors always have.