
LEGACY RISK

BY DR TOBY SUCHAROV OF ERUDINE

A WHITE PAPER FROM ERUDINE

**3 LEGACY: WHEN RISK LIMITS POTENTIAL
POTENTIAL BENEFITS**

**4-5 COST AND RISK
TECHNOLOGY RISK / TECHNOLOGY RISK MITIGATION / BUSINESS RISK / POLITICAL RISK**

6 THE COMMUNICATION PROBLEM

7 THE MENTAL MODELLING PROBLEM

**8-9 CREATING AGILITY TO REALISE POTENTIAL
INNOVATION AND THE 'IT CALCULATOR' / INVEST IN TRAINING /
MINIMISING RISK AND REALISING POTENTIAL**

**10 FURTHER READING
ON THE WEB**

LEGACY: WHEN RISK LIMITS POTENTIAL

If one were to take a tour of IT departments and try to obtain a consensus on the kinds of legacy systems currently in existence, it is likely that two views would pervade:

1. The stable old 'cash cow' system that is safely chugging away, probably after years of use, and steadily helping to generate high profits.
2. The nightmarish code maintenance hell system, where it cannot match changes in business requirements but is too difficult and too risky to modify considering the revenue it generates.

Both these systems cannot be changed easily. For the first system, the risk of damage during modification is significant enough to cause the potential benefits to be put aside. For the second system, modification is simply not possible because of the incredibly high costs and risks of doing so. As such, legacy systems can be defined as:

"Any system which cannot be modified either because the cost or risk of modification outweighs the potential benefits."

This view has two competing factors that are worth exploring: cost/risk against potential benefits. The important point is that if the cost and risks of modification can be lowered to an acceptable level, then a legacy system is transformed into an agile system and the owner can take advantage of any potential benefits, even ones that offer only small advantage.

POTENTIAL BENEFITS

The potential to leverage services and business information has greatly increased over the last decade. A company's core systems are often its biggest revenue generators and it is these that potentially offer the biggest benefits as a result of change. They are, however, also the systems that have the biggest potential risk associated with modification.

A recent study by the National Computing Centre found that 50% of respondents believed the adoption of new technologies to be too slow, and that legacy systems were a prime restraint on business change.

The benefits of modernisation can be massive: new revenue channels can be exploited, responsiveness to changing business conditions can be quickened, and data can be recycled to enable the use of joined-up information within a business. This is all in addition to a vastly reduced maintenance burden.

The increases in bottom-line benefits offered by agile IT systems are also compelling. Companies that can implement agile systems can raise profits not only by reducing maintenance overheads, but tapping new revenue streams and buffering the effects of entering new markets. For example, the success of Tesco has been largely put down to its effective and agile IT, allowing the supermarket chain to quickly enter new markets as they become liberalised, like finance and mobile phones.

All this prompts the question: if these agile systems offer so many advantages, what on earth can be the reason why a large number of businesses are not implementing them now?

COST AND RISK

Cost and risk are intimately linked. It is usually the case that the extremely high risk of complex IT projects is what drives their costs so high. Risk is a calculation of the expected and potential negative impact induced by alteration of a legacy system. Small levels of risk and low cost of change are what define an agile system.

TECHNOLOGY RISK

When undertaking a modernisation project to make a complex IT system more agile, there are several technological risks that will arise:

- Risk that the final system will not behave as expected
- Risk that the project will take longer and be more costly than expected
- Risk that the new system will be no more maintainable or agile than its predecessor

One important factor here is systems that generate higher revenue have a larger barrier to modification because system failure is more catastrophic. A relatively simple system made up of only 10 lines of code but which processes millions or billions of pounds in transactions may well be a legacy system simply because total system failure carries such a high financial cost.

TECHNOLOGY RISK MITIGATION

In the last 20 years IT has placed much emphasis on risk mitigation and a number of tools and best practices have been fashioned as a result. These are all geared towards making changes to code a simpler task by writing cleaner code, testing changes and maintaining documents that describe how the system works. All these methods will pay off in the short and medium term:

Refactoring. In the last 10 years the concept of refactoring code has taken off. It is used to tidy up or redesign the structure of code while keeping the functionality the same. Refactoring enables code to be reduced in both size and complexity, making future modifications easier.

Unit testing is a procedure to validate individual modules or units of code and check they are working correctly. With unit-tested code, modification is inherently less risky because the tests are more likely catch any unforeseen problems that occur from modifying the code.

Integration testing achieves the same ends as unit testing but tests the code from end to end. For the duration of the system's lifecycle this kind of testing ensures all the units of code are integrating together smoothly.

Code reviews are low-level tasks where developers review each others' code before it is put forward for testing. It ensures code quality and accuracy is maintained and removes risks incurred when modifying code.

System requirements capture and management. Requirements documents not only detail how the system should behave but also the critical factors that dictate when to enable the various aspects of behaviour. The documents provide support to people working with the system, from the original inception to code maintenance later in the system's life.

Neutral-platform systems, programmed in neutral languages such as Java, can be ported across different hardware. This allows the hardware to be updated independently of the software. Systems can be placed on mainframes and then moved onto commodity hardware when the mainframes' lifecycle expires and commodity hardware power increases to the point where it can sufficiently handle system processing power. Trying to move systems off mainframes is one of the main reasons for undertaking modernisation projects and neutral platforms remove this dependency.

Any system that utilises all these best practices and techniques will be an agile system - technological, political and financial risk will be small. Such systems will strongly support developers in changing the code and mitigate much of the risk of modifying systems.

BUSINESS RISK

Legacy systems are largely victims of their own success. If they were not so capable at generating income they probably would have been decommissioned long ago. The fact they occupy a vital place in many business operations makes them indispensable and remarkably resistant to change.

Most businesses are prepared to accept the high upkeep of legacy systems as long as they continue to meet requirements and/or generate revenue. IT departments are therefore unwilling to engage in modernisation because of the perception either that business processes would be severely interrupted or the flow of income would be stemmed.

Any successful modernisation would therefore need to be as non-invasive as possible and limit any anxiety that a core legacy system would be damaged in some way. The replacement system would not only need to be more flexible and easier to maintain, but perform precisely the same functions as its predecessor.

POLITICAL RISK

Modernisation of legacy systems also introduces risks to reputation as well as direct monetary risk. If the project stakeholders are not the same people who will benefit from successful modernisation then there is a chance that, on the political level, the modernisation tasks will never be undertaken. In these cases there needs to be buy-in from all stakeholders throughout the company. But this can be a delicate issue because company policy might well act as a barrier to modernisation.

For example, it is common for project managers to have contracts that annually reward successful management with bonuses; these tie-in project managers to a year-long financial cycle. Managers may well resist urges to start legacy modernisation or invest in using innovative technologies when nearing the end of their annual cycle. From the project managers' point of view, it may be more attractive to invest in modernisation and innovation at the start of the annual cycle and achieve a Return on Investment (ROI) before its end.

Such timescales are driven by policy to make project managers deliver. However, such timescales might not be in the best interest of modernisation tasks and innovation. In these situations, the idea of breaking down legacy systems into chunks and modernising them a piece at a time becomes attractive.

'Picking the low hanging fruit' is a phrase which is firmly established in business jargon, which simply means to perform the easiest task or attain the most achievable goal. If a system can be broken down into logical subcomponents and tackled in this way then it is likely that legacy modernisation will run more smoothly. But if a system is broken into illogical subcomponents to satisfy someone's bonus timescale then this componentisation may complicate the problem.

THE COMMUNICATION PROBLEM

Failure during IT projects is very often due to a communication collapse or, more commonly, a communication disconnection. This could be a disconnection between development teams or it could be between a development team and a commissioning body. The possibility of disconnection increases as more disparate parties become involved with a project.

IT has focused a lot of energy and time in looking at how to increase or solve the communication issues between these parties.

At present there is a clear segregation between commissioning bodies (who make the request for system functionality), domain experts (who hold the knowledge of system functionality), and developers (who write code to deliver functionality). While there are a plethora of tools and processes to address this divide, none of them fully succeed. To manage a clean interface between these parties, an interface is needed that allows developers to concentrate on development and the commissioning body to only concern itself with the domain.

But the difficulty in creating such an interface lays in the unrelated languages each group (developers, commissioners etc.) use to communicate. A commissioning body cannot talk in terms of logic, a development team cannot talk in the language of the domain, and there is no Rosetta Stone to help them communicate together. The only (unsatisfactory) option these groups have is to learn as much as they can about their counterparts' jobs.

Commissioners and developers communicate through different mediums. Developers want to use logic as their language transport mechanism; commissioners want to talk in use-cases and examples.

What frequently happens is that developers need to become domain experts or representatives from a commissioning body need to become developers to bridge the gap; this is a clear sign of an inefficient interface. The development world is bleeding into the domain world and vice versa. Even in the unlikely event that teams learn each others' language, communication is still difficult because not only were they speaking different languages they were using entirely different methods of communication.

Most physical communication between commissioning bodies and development houses is through requirements documents. If a requirements document was a full representation of code then the communication gap would be narrowed significantly, as there would be an accurate common interface between developers and commissioners.

But the act of writing requirements documents itself is an unnatural task for commissioners. Generating such formal documents is equivalent to asking domain experts to program.

The simplest solution is a technology that can automatically capture requirements and minimise the involvement of human beings. A technology that can extract all requirements from a legacy system and its users, and prove that requirements have been met by the replacement system, would circumvent most of the communication problems introduced by traditional requirements documents.

THE MENTAL MODELLING PROBLEM

Another problematic interface is that between computer and developer. When developers write code it is not enough for a developer to understand the piece of code they happen to be working on, there also needs to be a holistic understanding of the entirety of the code to ensure changes will not break other parts of the code.

When undertaking legacy modernisation using traditional methods, there is a risk that the limitations of developers may result in errors being carried over from the old system, or new ones written into the replacement. It is likely, because of the difficulty in understanding the entirety of the code, that at least some bugs will remain undetected.

Extreme Programming (XP) has tried to simplify the computer-developer interface by replacing the holistic view of how the code works with unit tests. Done well, this approach is a very effective and powerful tool for simplifying this interface and simplifying the task of programming at the unit level. However, writing unit tests is a difficult skill in itself and considering that, under the XP method, all areas of the code should be tested it would seem prudent that this task become a prime candidate for partial automation.

Automating developers' ability to understand the entirety of a system vastly diminishes the risks presented by the limitations of the human mind. The ideal would be a technology that can encompass all the functionality of the legacy system, perform tests as modernisation takes place, and visualise this clearly to users.

Such a technology would avoid any gaps in understanding that inevitably occur when a system has been maintained for some time by several different people, each with slight variations in their knowledge of how the system works.

And because the new system would hold all knowledge of functionality, with clear visualisation, it becomes much safer to change. The trepidation that comes with modifying legacy systems is no longer there as the new system clearly shows how new functionality will fit and resolves any clashes with existing system behaviour.

CREATING AGILITY TO REALISE POTENTIAL

INNOVATION AND THE 'IT CALCULATOR'

Legacy systems are largely victims of their own success. If they were not so capable at generating income they probably would have been decommissioned long ago. The fact they occupy a vital place in many business operations makes them indispensable and remarkably resistant to change.

Most businesses are prepared to accept the high upkeep of legacy systems as long as they continue to meet requirements and/or generate revenue. IT departments are therefore unwilling to engage in modernisation because of the perception either that business processes would be severely interrupted or the flow of income would be stemmed.

Any successful modernisation would therefore need to be as non-invasive as possible and limit any anxiety that a core legacy system would be damaged in some way. The replacement system would not only need to be more flexible and easier to maintain, but perform precisely the same functions as its predecessor.

INVEST IN TRAINING

Currently, IT has not been simplified to a level where it can be performed by laymen, and IT will probably never reach this stage. However, it is becoming simpler as innovative tools and processes remove the communication barriers and cognitive load from developers, but until organisations widely adopt these new technologies, they need to strongly invest in training.

Training needs to last beyond the traditional six months or year where developers are normally let loose on system code and additional training methods need to be added to this process. Much like apprenticeships of old, there needs to be a mentoring of developers. This should continue not just for a year or two but as an ongoing process - even for master developers.

Only companies that invest in training will be capable of delivering agile systems, but such agility is currently beyond the average industry developers who make up the bulk of the IT workforce.

Right now, there are not enough IT candidates to deliver systems that will drive businesses forward as far or as fast as should be possible. And training is not universally embraced as the solution to this shortfall.

For highly skilled jobs it is common for workers to go through a lengthy apprenticeship stage. For example, printers used to work up through the apprentice stages, to journeymen, and then to masters. This process would take many years and would result in highly professional and capable staff. The same process is used today with solicitors, builders and doctors.

There is no real apprenticeship within IT. By far the most common route to development is straight from a degree course and into a development house. There may be some cursory training, there might even be mentoring, but it is common to see developers performing very advanced roles after only a year or two of experience.

Another interesting point about printing apprenticeships is that they no longer exist; IT and word processors have automated the highly-skilled printing process almost completely. A current typesetter lacks all the experience and wisdom that his predecessors had because software now handles the majority of the tasks and there is no longer as great a need for all that knowledge.

The relevance to IT is that when technology does not raise standards far enough, professionals need extensive training and experience to completely fulfil their tasks. When technology is capable of sufficiently raising the bar, this training is not as necessary.

Presently, the route to agile systems can be found in training developers to ever higher standards, combined with the adoption of new technologies that simplify the legacy modernisation process.

Once organisations have mitigated the risks inherent in legacy modernisation through training and innovation, they will be able to realise their true potential with agile IT systems.

MINIMISING RISK AND REALISING POTENTIAL

Risk in all its forms is ever-present in business IT. The challenge with legacy modernisation is to minimise the risks and maximise the potential benefits.

The risks of legacy modernisation are well-known but can be overcome with careful planning and application of effective technologies, training and testing. Acknowledging the risk is the first step, it should not prevent a move towards agile systems that allow potential benefits to be realised.

There is an apparent perception among businesses that the technology or processes do not exist to allow legacy systems to be safely changed. But such innovations are available and need not be the sole preserve of small, swiftly-changing companies. Large multinationals could open up the possibility of millions of pounds in extra revenue if they could instil the appropriate business culture and adopt the correct technologies.

Companies are currently hemorrhaging money into developing and maintaining legacy systems; this will turn into massive profits for the businesses that can stem this flow using tools that simplify the entire legacy modernisation process and create truly agile systems.

FURTHER READING

The Innovator's Solution: Creating and Sustaining Successful Growth. M. Christensen, Michael E. Raynor, Harvard, ISBN-13: 978-1578518524.

Crossing the Chasm: Marketing and Selling Technology Products to Mainstream Customers. Geoffrey A. Moore, Capstone Publishing, ISBN-13: 978-1841120638.

Experimentation Matters: Unlocking the Potential of New Technologies for Innovation. Stefan H. Thomke, Harvard, 2003. ISBN-13: 978-1578517503.

Modernizing Legacy Systems: Software Technologies, Engineering Processes and Business Practices. Robert Seacord, Addison Wesley, 2003. ISBN-13: 978-0321118844.

ON THE WEB

DARWIN: On Incremental Migration of Legacy Systems. Michael Brodie, Michael Stonebraker.
<http://db.cs.berkeley.edu/papers/S2K-93-25.pdf>

Chickens and Turkeys Migrate, but not Necessarily in IT. Ben Wilson.
<http://www.anubex.com/migrationscentre!chickensandturkeys.asp>

IT Needs a Culture Change. Lou Washington. <http://www.ciupdate.com/career/article.php/3652426>

WWW.ERUDINE.COM

Central House Beckwith Knowle Otley Road Harrogate HG3 1UG United Kingdom

Tel: +44 (0) 1423 522 336 / Fax: +44 (0) 1423 701 542

technical@erudine.com / www.erudine.com

Design: www.solutiongroup.co.uk