IVEY | Publishing

9B13E019

IBM: THE ITERATIVE SOFTWARE DEVELOPMENT METHOD

Rakan Khalid wrote this case under the supervision of Joe Compeau solely to provide material for class discussion. The authors do not intend to illustrate either effective or ineffective handling of a managerial situation. The authors may have disguised certain names and other identifying information to protect confidentiality.

INTRODUCTION

On November 16, 2010, John Lalonde, the head of IBM's Rational Product Development, was sipping his café mocha thinking about his upcoming meeting with his counterpart, Neil Mayor, the head of IBM's WebSphere Product Development. Mayor had requested Lalonde to deliver an additional feature in the Rational product that would then enable WebSphere to deliver a particular feature. Although the Rational and WebSphere products had interdependencies and complemented each other, each product offered a unique functionality to IBM customers and thus each was sold by IBM as a separate product.

The feature requested by Mayor was a customer requirement that had the potential to generate $40 million[1] in revenue for IBM, but the requirement had come very late in the product development cycle. Although the customer requirement was for WebSphere, the interdependency between WebSphere and Rational and the customer's request for a complete end-to-end solution involving the two products meant that the Rational product needed to develop the additional feature in response to the customer requirement received by Lalonde.

Rational's product development team used the agile iterative software development method to develop the product. Unlike the traditional method of developing software products, in which all requirements were hashed out before the product development cycle, the iterative nature of the development process meant that requirements could be received during the product development cycle.

Lalonde faced many questions. Should he undertake Mayor's requirements? If so, could Lalonde deliver the functionality in time without delaying the product release? Did Lalonde have the resources to do so? Would it be risky to change the software code late in the development cycle and potentially break the existing product functionality? Lalonde needed to come up with answers to these questions and ensure that a high-quality product was released on time and within the budget.

[1] *All currency amounts are shown in U.S. dollars unless otherwise noted.*

INTERNATIONAL BUSINESS MACHINES

Founded in 1911 and headquartered in Armonk, New York, International Business Machines (IBM) was one of the world's largest providers of informational technology (IT) solutions and services. In 2010, IBM generated profits of $14.8 billion on $99.8 billion of revenue. IBM generated its revenue through five different business segments (see Exhibit 1):

1. Global Technology Services
2. Global Business Services
3. Software
4. Systems & Technology
5. Global Financing

Up until the late 1980s, IBM's business portfolio had revolved primarily around computer hardware such as the System 390 and AS/400 mainframe servers, ThinkPad and ThinkCenter personal computers, memory, disk drives and printers. The competition in the computer hardware business became intense with the emergence of players such as Dell, Toshiba, Acer, HP and Compaq (now part of HP). IBM soon realized that the hardware business was shifting into a commodity business. It needed a newer business model that would usher it in as the technology titan in the sophisticated 21st-century computing world.

Under chief executive officers Louis V. Gerstner and Samuel Palmisano, IBM had shifted its business mix into higher-value computing businesses that consisted of software and services. IBM exited the low-margin hardware business by divesting its personal computing division to China-based Lenovo Group and by discontinuing the manufacturing of memory, disk drives and printers. IBM focused its investment into building enterprise software such as databases, application servers and infrastructure management software in its research and development labs across the globe. It also acquired several software and services companies to build world-class IT solutions and services capabilities. IBM's strategy to move into higher-value businesses resulted in a larger proportion of its profits being generated from software business segment compared with in the early 2000s. The shift in strategy enabled IBM to remain a relevant force in the IT industry, while some of its competitors, such as HP and Dell, struggled with their lower-margin, hardware-focused business.

In 2009, IBM introduced its five-year roadmap (2010–2015) that could be summarized by the statement "Generating Higher Value at IBM." The roadmap consisted of investing in high-growth software businesses, such as cloud computing and business analytics, and aggressively expanding its revenue from emerging markets. IBM's management hoped that its five-year roadmap would culminate into an Earnings per Share (EPS) of $20, double that of 2010. The software business segment would play an important role to help IBM reach that financial goal as IBM expected the software segment to generate 50 per cent of its pre-tax income by 2015. In 2010, IBM's software business segment generated revenue of $22.5 billion and pre-tax income of $9.1 billion (44 per cent of IBM's total pre-tax income).

THE COMPETITIVE LANDSCAPE

The diverse nature of IBM's business portfolio meant that IBM competed against firms in the broader IT software, hardware and services market. Thus, its software business competed against large firms such as Microsoft, Oracle, SAP and Symantec (see Exhibit 2a). The extremely high gross margin in the software business (approximately 80 per cent) meant that the competition in the software business was fierce, with every player trying to gain as much of the market share as possible. IBM also faced competition from smaller niche players, such as Splunk and Cloudera. In the hardware business, IBM primarily competed

against HP, Dell, Oracle (which now included Sun Microsystems) and Fujitsu (see Exhibit 2b). The services business faced competition from the likes of HP, Accenture, Fujitsu and CSC (see Exhibit 2c).

IBM RATIONAL AND WEBSPHERE

IBM's software business segment was composed of the IBM Software Solutions Group and IBM Software Middleware Group. The various software solutions and products from these groups could integrate with each other.

The solutions group's portfolio consisted of industry-aligned solutions that were pre-packaged using a broad set of IBM software and hardware. IBM had deep domain knowledge across a range of industries and had software solutions built around industry-specific frameworks to solve industry-specific challenges. These solutions fell broadly under the following groups:

1. Business Analytics solutions such as business intelligence, predictive analytics and analytical decision management. These solutions analyzed massive amounts of past data and provided insights to help organizations make more informed and optimized decisions.
2. Collaboration Solutions such as instant messengers, virtual meeting rooms, email systems and document management systems to help customers, partners and clients interact with each other to enhance productivity and efficiency.

The middleware group products provided the software infrastructure that powered an enterprise software system. The IBM middleware group software fell under the following four brands:

1. Information Management: Data Warehouse, Data Security and Data Governance software systems that enabled organizations to deliver proper information throughout the organization's information supply chain.
2. Tivoli: Network Management, Mobile Device Management and Enterprise Asset Management software systems that helped organizations to manage and optimize their IT infrastructure.
3. WebSphere: Application Deployment and Business Process Management Software systems provided organizations with the ability to efficiently run their business applications.
4. Rational: Application Development and Lifecycle Management tools helped organizations to accelerate development, delivery and maintenance of software systems to enhance productivity.

IBM's Rational Application Developer (RAD) and WebSphere Application Server (WAS) platforms had a high degree of interdependencies between them. In fact, IBM released every new major version of these products on the same date. Although the two products complemented each other, integrated tightly and enabled an end-to-end full solution for customers, each product's strategy and development cycle was governed by independent business units that had different budgets, employees and resources.

WAS, the result of IBM's in-house development efforts in application servers, was first released in 1998. IBM subsequently released more than a dozen new versions of it over the past 14 years. RAD came into existence in 2005 after IBM, in 2003, acquired the Massachusetts-based Rational. IBM combined the software platform from Rational and IBM's in-house application development platform to result in RAD. Both RAD and WAS were developed by hundreds of software developers across software development labs in the Americas, Asia and Europe. The complexity of building multimillion-dollar software globally meant that a tight collaborative working environment was essential to the success of the two products and the broader IBM Corporation.

While the WAS product allowed companies to efficiently and securely run their Java-based business applications, the RAD product helped software developers to create and maintain the Java applications that ran on the WAS family of products. Revenue from selling these two products' software licences could range from tens of thousands of dollars to millions of dollars. The sales of software licences in turn drove IBM's services and maintenance line of business that earned IBM additional revenue in the range of tens of thousands of dollars to millions of dollars.

THE SOFTWARE DEVELOPMENT PROCESS

A software development project could span from three to 15 months, depending on the complexity and scope of the project. Large–scale, sophisticated software development teams typically comprised software architects, programmers, testers, product managers and project managers. With the proliferation of the Internet, members of a software team could be distributed across various different locations. Software projects were highly process-driven because a lack of process typically led to delays, poor quality and an increase in project cost.

Software Systems were developed in various phases that included requirement analysis, design, implementation and validation. Although every software development method incorporated these phases, the execution of these phases in the software development cycle varied across the different development methods.

In the requirement analysis phase, software architects and software product managers focused on understanding and gathering customer requirements. Although customers believed that they knew their requirements well, at this stage, requirements were typically ambiguous and incomplete. Using their vast experience, software architects injected clarity into the customer requirements to build a solid definition of the problem that the customer needed to solve.

During the design phase, software architects developed high-level, abstract solution architecture for the problem. The solution architecture defined how the different components of the software system would interact with each other. At this stage, the developers determined the engineering details, such as the different operating systems the software system would support, the type of programming language to use, and the algorithms and data structures to use. Developing a sound design upfront was important, as changes to the design later in the development could be costly and sometimes impossible to implement.

During the implementation phase, the software engineers converted the solution architecture captured in the design phase into a working software system. The focus of this phase was to build, with minimal bugs or defects, a software system that was flexible, scalable and conformed to both the customer's requirements and the software architect's design plan. Any defects in the system needed to be caught as early as possible; when identified too late, defects were both costly and difficult to fix.

The validation phase, more commonly known as the testing phase, ensured that a high-quality software system was delivered to customers. Although software developers made every effort during the implementation phase to ensure that they developed a bug-free software system, it was an impossible task, as software systems were highly complex and developed by hundreds of different software developers. Even the most experienced software developers could not judge the various different scenarios under which a customer would use the software. Software testers were required to run rigorous test scenarios on the system and report any defects found to the development team. The development teams, in turn, fixed the defects and tried to ensure that they didn't introduce any additional defects or

break existing functionality during the code fix. The latter was known as a regression defect. As the test phase neared completion, the software system became stable, robust and high in quality.

After the testing phase was completed, the project team moved the project into "code freeze," the stage at which software developers could not add any new code. Code freeze was important to ensure that no new defects were introduced to a robust system. Projects were also moved into code freeze to help move the project forward to completion and release by rejecting any addition of code.

THE WATERFALL SOFTWARE DEVELOPMENT METHOD

The waterfall software development method was a traditional software development process in which software was developed in a rigid and sequential style. The development process started at the top with the requirement phase and sequentially continued on to the next phases lower in order. The process never moved onto the next phase until the current phase was fully completed. For example, implementation or coding never started until the design was fully completed (see Exhibit 3).

Proponents of this method believed that the waterfall method forced project teams to fully understand a customer's requirements and to build a solid design plan that was set in stone. They believed that this method saved time, money and effort, as late-stage changes were difficult to implement. However, opponents of this method believed that it was impossible to perfect a particular phase before moving onto the next phase. They argued that software architects, in the design phase, might not be able to predict potential future implemental difficulties that could arise in the implementation phase, leading to an inevitable change in design in the later stages. Another issue was that customer and stakeholder requirements continued to evolve, which forced the development team to alter their architecture, design and implementation. Finally, the waterfall method did not allow software testers to test the system until after the implementation was fully completed. As a consequence, defects in the software system were sometimes discovered very late, and late-stage defects were extremely difficult or even impossible to fix due to the complexity in the system.

The waterfall software development method had worked for many years, but lately had fallen out of favour. In recent years, increased business competition and evolving customer demands had resulted in businesses becoming highly dynamic and fast-paced. To support these changing business processes required changes in computer software systems. However, the waterfall method did not accommodate changing requirements during the middle of the development cycle.

THE ITERATIVE SOFTWARE DEVELOPMENT METHOD

An alternate to the waterfall development method was the agile iterative development method. In the agile iterative development method, software was developed in iterations because software requirements evolved as the development cycles progressed (see Exhibit 4). Moreover, as the name suggests, an agile iterative method allowed development teams to easily respond to evolving and changing customer requirements. The focus of this method was to develop high-quality working software in small increments, allowing the project development team to learn from previous iterations. The working software could also be distributed to the various stakeholders to gather feedback and to allow for any corrections or changes to be made in the development cycle as early as possible. In the waterfall method, it would not be possible to incorporate any changes after receiving feedback.

IBM's RAD development team decided to employ the agile iterative software development method to reap the aforementioned benefits. WAS did not provide the RAD team with all the requirements upfront during the 18-month product development cycle that began on June 25, 2009. Both WAS and RAD products were scheduled for release on December 22, 2010.

Each iteration cycle lasted for six weeks:

- Four weeks allotted for the planning, design and implementation phase
- Two weeks allotted for the testing phase

Through the multiple iterations within the 12-month development cycle, the product development team was required to deliver all the features that IBM had committed to develop. The goal of each iteration was to develop a subset of features that were of high quality. Quality was extremely important, and it was key to being able to ship the working software to customers to gather feedback.

During the initial four-week period of the iteration, software architects and developers created the design document that contained the solution architecture for the subset of features under consideration for development. In the latter part of that four-week period, software developers would be busy writing code, and software testers used the design document to develop the test plan and test scenarios. The test plan and scenarios needed to be approved by the various stakeholders to ensure that a comprehensive test plan was developed. At the end of the four weeks, the iteration moved into the test phase. Software testers ran the test scenarios and reported any defects they found to the development team. The defects were fixed within the two-week test phase window. If a defect could not be fixed in the current iteration and if it rendered a particular feature to be non-functional, the code for the entire feature was backed out from the software system. This practice was consistent with the ideology that any feature developed in iteration should be fully functional as if it were ready to be released to a customer. The fix for the defect and the feature that was rendered non-functional would be deferred to the next iteration. The next iteration's planning and design team would take into consideration the time and resources needed to fix the defects and enhancements from the previous iteration.

The iteration would repeat in similar style until all the features requested had been developed and tested for quality. At the end of the final iteration and once the software system was deemed to be both stable and of high quality, the project team would move the development cycle into code freeze.

DECISION

On November 10, 2010, Mayor held a meeting with Lalonde to discuss a new requirement received by a customer. The customer was a retail company and had purchased its first IBM product two years earlier. The customer had also bought software and services from a variety of IBM competitors in the past, including Oracle and HP; thus, its software ecosystem had been developed using a variety of different vendors.

If the feature were to be developed by WAS, it would help the WebSphere business earn an additional $30 million in revenue. A corresponding feature in RAD would allow Rational to earn $10 million in revenue. In total, the software deal would earn IBM an additional $40 million in fiscal 2010. If both features were not delivered, IBM would lose out on $40 million in revenue because the customer would only buy the products if both RAD and WAS developed the required features.

The software development cycles of RAD and WAS were both about to enter into their final iterations. The WAS team had secured approval allowing it to write additional code to develop the new feature during the final iteration. Mayor's research indicated that the WAS group could safely add the new feature without compromising product quality and without delaying the product release date. If Lalonde's team were to add the new feature, Lalonde would need to seek approval to write code during the final iteration. Moreover, Lalonde needed to decide whether it was worth risking the quality of the product and release date by adding new code during the development cycle's final iteration. IBM had commitments and contracts with customers to release the product by the release date. Lalonde also needed to understand the impact his decision would have on IBM's 2015 roadmap. What effect, if any, would it have on future business opportunities with this customer? Could developing this feature help IBM win this customer over from its competitors and retain the customer as a loyal IBM customer?

Another option would be to develop a prototype of the feature requested by the customer that would provide a subset of the functionality. This option would not only minimize the risks of delaying the product release and of inadvertently introducing any potential defects but would also provide the customer with a subset of the functionality that had been requested. This option would also help IBM to minimize the revenue loss that it would have been able to generate if it had been able to deliver the complete feature. However, IBM would need to convince the customer to accept the prototype at its current stage and wait for the full feature to be delivered in the future. If the customer disagreed, IBM would jeopardize losing the $40 million in revenue.

EXHIBIT 1: REVENUE AND PROFIT COMPARISON OF DIFFERENT IBM BUSINESS SEGMENTS

Revenue (in millions)	2000	2010
Global Services *		
Revenue	$33,152	$56,424
Pre-tax Profit/(Loss)	$4,517	$8,137
Software		
Revenue	$12,598	$22,485
Pre-tax Profit /(Loss)	$2,793	$9,097
Systems & Technology		
Revenue	$37,811	$17,973
Pre-tax Profit /(Loss)	$2,702	$1,586
Global Financing		
Revenue	$3,500	$2,238
Pre-tax Profit /(Loss)	$1,176	$1,959
Other		
Revenue	$1,372	$750
Pre-tax Profit /(Loss)	$(297)	$ (18)
Total		
Revenue	$88,430	$99,870
Pre-tax Profit /(Loss)	$11,534	$20,761

** Includes Global Business Services and Global Technology Services*

Source: IBM annual reports

EXHIBIT 2A

Top 5 Enterprise Software Vendors by Total Enterprise Software Revenue in 2010

Vendor	Market Share
Microsoft	22.4%
IBM	10.4%
Oracle	9.8%
SAP	5.3%
Symantec	2.3%
Other Vendors	49.8%
Total	100%

Source: Gartner

EXHIBIT 2B

Top 5 Server Vendors by Revenue in 2010

Vendor	Market Share
HP	31.4%
IBM	30.8%
Dell	14.7%
Oracle	6.3%
Fujitsu	4.4%
Other Vendors	12.3%
Total	100%

Source: Gartner

EXHIBIT 2C

Top 5 Services Vendors by Revenue Estimate in 2009

Vendor	Market Share
IBM	7.3%
HP	4.1%
Accenture	3.2%
Fujitsu	3.1%
CSC	2.5%
Other Vendors	79.8%
Total	100%

Source: Gartner