RAD Applied in the Context of Investment Banking Trading Systems Development

Iosif ZIMAN
Nomura Principal Investments Hong Kong Ltd.
iosif.ziman@nomura.com

RAD as a methodology for implementing information systems has been used in a broad range of domains utilizing technology as an informational backbone but perhaps one of the main areas where this approach has been proven to be a natural fit has been in the investment banking (IB) industry, most notably when applied to trading systems. This paper introduces some of the main tenants of RAD development and focuses on a number of case studies where RAD has proven to be an extremely suitable method for implementing solutions required in the IB industry as well as explaining why RAD may be more successful than other classic development methods when applied to IB related solutions.
Keywords*: RAD, Information Systems, Investment Banking, Trading Systems*

1 Introduction

RAD has its origins in rapid prototyping approaches and was first formalized by James Martin (1991). He believed that it refers to a development life cycle designed for high quality systems with faster development and lower costs than the traditional lifecycle provided. Martin's work followed up on early concepts such as Barry Boehm's spiral model, Tom Gilb's evolutionary life cycle, and Scott Shultz's rapid iterative productive prototyping (RIPP). The prototyping method used in rapid application development allows the developer to rapidly identify the types of data and process models required to meet the application requirements. However, because of the shorter development time, certain compromises in performance and quality are difficult to avoid. By the mid 1990s the definition of RAD came to be used as a cover term to include a number of methods, techniques and tools by a large number of different vendors applying their own interpretation and approach. This rather unstructured ad hoc evolution of RAD means that the rationale behind its use is not always clear. It is perceived as an IS system methodology, a method for developers to change their development processes or as RAD tools to improve development capabilities (Beynon-Davies 1999, Whitten 2007). It could be found that on a number of occasions RAD has been considered one of the delivery methods encompassed in Agile development methodologies. According to circulated literature RAD centers on prototyping and user involvement stages where the analysis, design, build and test phases of the development life cycle are compressed into a sequence of short, iterative development cycles. This had been seen as a remedy to perceived flaws of the traditional lifecycle because the iterative approach encourages effectiveness and self-correcting as each increment is refined and improved. To achieve this, a RAD approach necessitates the collaboration of small and diverse teams of developers, end users and other stakeholders (Martin 1991, Tudhope 2001, Beynon-Davies 1996, Elliott 1997). It is sometimes useful to consider that RAD projects may be distinguished in terms of intensive and non-intensive forms. A non-intensive approach to RAD refers to projects where system development is spread over a number of months involving incremental delivery compared to the intensive RAD where project personnel works somewhat secluded to achieve set objectives with a 3 - 6 week timeframe (Beynon-Davies 1999, 2004).

For a description on the reasons why RAD is well suited to development of IS in investment banks and more specifically trading systems a brief description of the development process including analysis, planning as well as development, testing and integration

is achieved within such organizations. It is important to understand that while the contrary may be desirable in fact an IB trading environment tends to be a hard to capture user functional environment and as such specifications tends to be difficult to pin down and more so have accepted by mail stakeholders, this being often the time one of the main reasons why projects within such organizations may be found to fail repeatedly, despite the need for the functionality that they would offer once delivered. There are multiple reasons why this happens and we can just mention some of them. The trading environment is a highly dynamic one and tends to be populated on the user side by people who have daily responsibilities and for whom system development is not always at the core of their focus at all times. This means that only sporadic attention may be expected from the people best placed to offer requirements, which means that from the very beginning there is a clear case of scope creep. At the same time these same people are in need of new functionality that they demand but yet have a difficult time in focusing on specifying exactly what they require to a level that can then be implemented based on further analysis. At the same time on the analysis and specification definition phase the large number of people that need to be involved for implementing significantly large information systems tends to require a relatively long period of time simply to define requirements, which in fact may have changed before reaching the development phase. As a result often times such institutions are faced with potentially low productivity cycles designed simply to achieve a consensus on requirements. It is important to understand that quite often developments in this area border research and development activities rather than simple problem solving solutions delivery; as a result the time required could be multiplied by orders of magnitude. One general problem is that many users who are involved in the specification phase only really become proficient in providing feedback once they see a first level implementation that is somewhat functional and will then tend to give input on methods of evolving or improving (Gerber, 2004). This is where RAD comes in and offers developers the chance to promote a controlled, structured but flexible development methodology aimed at providing incremental delivery. This generally involves a series of time-boxed mini iterations and a number of software 'release' and test iterations to provide flexibility to meet the recognized volatile needs of the business environment. In general analysts and developers believe this methodology offers all the main benefits of a RAD type approach and is suited to the uncertainty of, and continually changing business requirements. To that extent a structured RAD involves prototyping and iterative delivery while keeping tabs on the problems of lack of rigor, creeping scope and overrun that are perceived as associated with an undisciplined RAD and an iterative development life cycle. The method uses the same main features i.e. workshops, time-boxing, prototyping, intensive user involvement, iterative development and incremental delivery, which they maintain are increasingly used for system functionality development. Analysts and developers believe that a major benefit of an iterative approach to development is that it affords early visibility of the system being developed. As such early validation of the system by the users and the business analysts provides the flexibility to incorporate user feedback and handle any new or changing requirements within the volatile business environment – a key goal of the RAD approach.

A useful example to give at this stage is that of the development process of a proprietary structured equity derivatives system's development cycle. In the case of a given IB it took 4 attempts, of which 3 failed over a period of about 7-8 years, until the 4^{th} attempt has been successfully implemented over a further 7 year period. The reason why the first 3 attempts failed have been varied but it generally had to do with the fact that analysis and development assumed that the needs of end users are already understood and all that is needed is a good quality product that needs to be fully developed and deployed across the

firm. The problems in each cases had to do with the fact that the multitude of, sometimes individual, needs could not be estimated in the analysis and specification phase and invariably the systems developed failed to meet the requirements of users and, having already overrun their budgets, lost sponsorship and failed to be successfully terminated. The reason why a 4th attempt has been successful has been exactly because a RAD type approach has been used in which a lower spec version of the system has been developed and deployed to less demanding users and functions and then gradually enhanced over years of development to include further complex functionalities until finally succeeding in eliminating the legacy system. This was a major triumph for a RAD type approach in developing trading systems.

In general as a systems development approach RAD has both critics and supporters whose opinions, in some cases, are fundamental to individual philosophies and perceptions of this method's rationale. Existing literature presents particular themes of discussion within the RAD arena and a prominent area of debate concerns the scalability of RAD across large and complex environments. While a fair observation is that across the software development industry the lack of provenance is reflected by the limited availability of published material, there is substantial reporting of its application and considerable debate about its appropriateness for different types and sizes of systems development. (Osborn 1995, Beynon-Davies 1999, 2000). Also important to note is that RADs origins as a development process is fair to be placed more within a commercial development arena than an academic one. As such literature considers it more appropriate for small to medium simple, highly interactive development projects rather than for environments that are also computationally complex even if the case mentioned before proves that is not necessarily the case. In general it is observed that RADs success is linked to the project management approach, level of management commitment, degree of end-user involvement and the ability of the team to make fast authoritative decisions (Beynon-Davies 1998). Literature also suggests that RAD projects necessitate cultural and managerial changes because people are required to behave in a different way than in the more structured traditional environments. It is therefore important to note that without radical shifts in organizational attitudes and structures and peoples' mindsets many projects may fail because the change to new methodologies, methods and techniques did not fit within the culture (Hirschberg 1998, McConnell 1996). It can further be observed that the potential of a RAD development and delivery approach to meet information systems requirements in uncertain and volatile business settings of complex system development environments is questioned. Such critics advocate that the need for high levels of user involvement, stakeholder collaboration, lack of project control and rigor are major issues to its success (Ritu 2002, Martin 1991, Osborn 1995, Beynon-Davies 1996, 2000, Elliott 1997, Cross 1998, Boehm 1999, Highsmith 2000). Our personal experience shows that a RAD process fits well IB trading systems development, if mainly because of the difficulty in finding alternative suitable methods of implementing solutions for such systems.

2 RAD – The Search for the Optimal Method

Overtime many software architects have been preoccupied with the search for the optimal development method which would allow optimal balance between all elements contributing to the process, including analysis, know-how, programming skill sets, formal specification and prototyping methods and the many others that need not be enumerated in an exhaustive manner. For RAD one of the possible strategies is to make an investment upfront into specific frameworks, these could be basic fundamental ones, such as using high level languages, to using third party libraries, such as boost, or most of the case in the IB context the use of dedicated APIs provided by internal or third party systems which allow a modularized approach and

RAD type enhancements to existing systems functionalities. Or in some cases even allow implementing core type functionality based on a modular architecture of already existing systems. These systems tend to be preferred in fact in an IB context.

To continue on the IB context what tends to be the preferred RAD type approach is to develop internally in the IB, or indeed adopt external systems, which offer this modular type capability and come equipped with a fully transparent API which offer access to many if not all capabilities of the system. In this way many systems needs can be catered for either by modifying and adapting existing modules using this API or implementing new ones which mostly extend the system, without the need for "reinventing the wheel" and implementing new systems infrastructures thus keeping a good watch on costs and consistency of the systems architecture. In order to achieve this however much care is needed to ensure that generally geographically dispersed teams that need to cater for often diverging needs maintain consistency through communication and use of common tools. This type of architecture is generally achieved only at advanced stages of development in the lives of IB organizations after having gone through multiple painful expansionary and realignment cycles. In Fig. 1 can be seen a high level geographical consistent system deployment. By comparison it is possible that IB will have different systems implemented in different locations which all will require dedicated processes involving usage, deployment and integration, all of which greatly increase the complexity and cost of the operations. Hence the importance of consistency while also catering for diversification is paramount in such organizations.

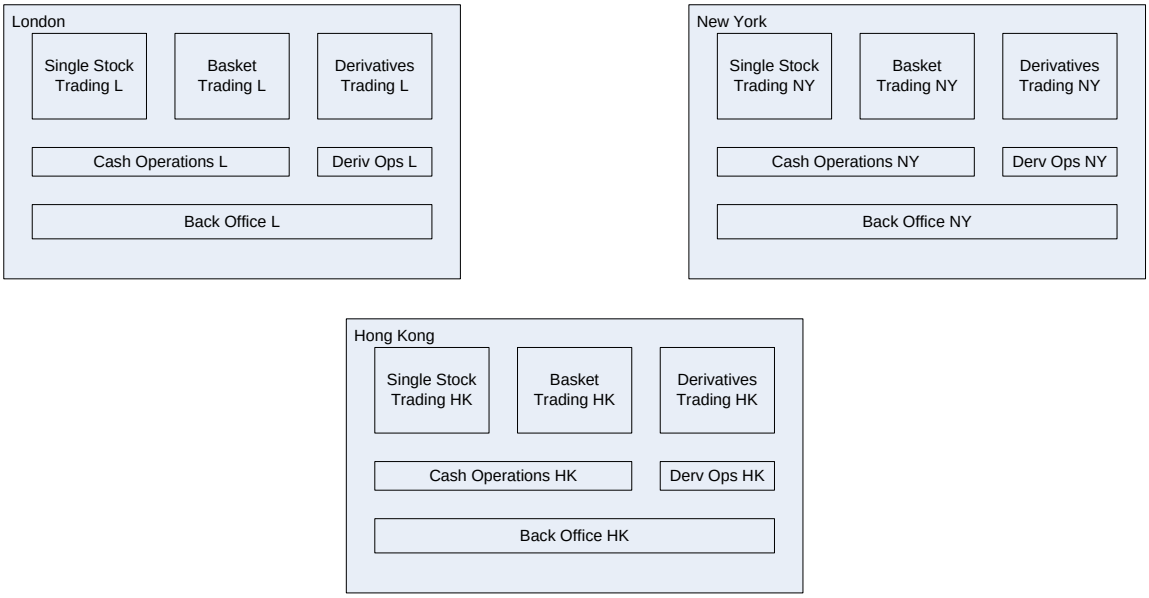

Fig. 1. Geographical System Consistency

In order to correctly place the usefulness of RAD we need to clarify which are the differentiating factors from a methodology perspective that ensure this approach has strong chances for success. Formal requirements are needed to establish a clear definition of tasks as well as being used to communicate systems requirements among the user and developers. These requirements nee to include system functions, performance goals, schedule and cost estimates. Requirements need to include design features, performance goals, and schedule and cost estimates. The use of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) may be an important resource in suggesting what should be done by having a well defined and well understood process. An important goal

of RAD is to keep the time between design and delivery as short as possible. As such the use of cost estimators such as the COCOMO model and PERT charts for ex. to stay on the critical path are highly encouraged (Hirschberg 1998).

One of the important factors to focus and keep in mind is that for a RAD approach to work the firm needs to have a continuous high quality production environment. Generally the best way to approach RAD is with a team of users and developers who communicate effectively and can successfully develop products with well established schedules and within agreed costs. This is one of the most important factors in implementing RAD is the experience of at least part of the personnel involved. In order to ensure that such an approach works at the management level there should be a constant effort to eliminate or reduce tasks that are not necessary, help streamline activities and maintain focus. A well trained and collaborative team is important in RAD and essential for success. The team should have a well formed core which collaborates in setting priorities and in agreeing planning. Also this core is important to me constant for the team across the life of the project, any significant changes being certain to impact the delivery. Multiple aspects of the process need to be taken seriously and considered within the team with increased importance given to quality and configuration management, monitoring and SDLC forms being in place to aid continuous production and reduced maintenance cycles. One of the problems that IB face is in establishing such teams given that experienced resources are often employed in well established activities and sourcing them externally tends to involve considerable time and money. At the same time once resources are secured an important aspect is the retention of the core team and to achieve that motivation is an important consideration.

Rapid application development tends to use relatively small teams of about 4-6 people who develop and test the new application. The team creates a specification for the application followed by simulations of prototypes. A working prototype in a relatively high level language/environment is created prior to coding the initial version. RAD often involves creating many versions of the application, as per the RAD model in Fig 2. Staying on schedule is of great importance. To achieve this users and developers use a process of iterative prototyping in which a structured process is repeated until a usable application is created. The steps involved can be generally stated as:

- Building of a working prototype
- Reviewing the prototype
- Beta testing the prototype
- Meeting between developers and customers to work out kinks in the prototype
- "Timeboxing" needed changes in prototype

The RAD process can vary in the balance between speed and quality depending on the ultimate project requirements. The faster the end product is needed, generally, the greater the sacrifice in product quality. Also, faster application development generally translates into higher development costs.

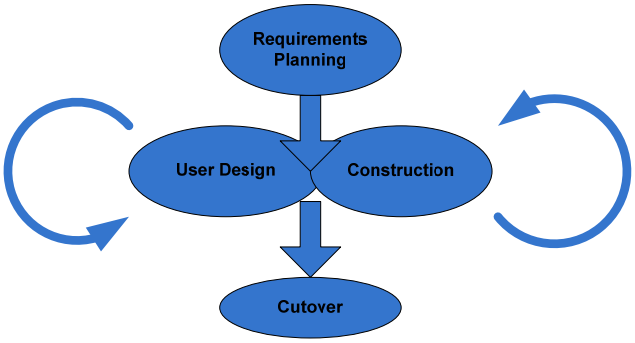

Fig. 2. RAD Model

Thus, it is obvious that use of RAD involves situations when schedule and deployment are overriding factors, especially in comparison to quality, performance, and development cost. An example of such a situation would be a case in which a company desires to be first when introducing a product to the market. Another example would be products that must address rapidly changing environments. In such cases, the situation may change by the time an application is developed using slower, more traditional processes. Such considerations are very important in the context of IB and trading especially given that the environment is a very dynamic one and market conditions vary greatly which means that the need for timely and suitable solutions is paramount.

This is why RAD, which facilitates a greater collaborative atmosphere, as the testing and reviewing process is more fluid and evolutionary, seems to be a natural fit for IB environments. In more conventional processes, user feedback is used only for completely finished products. With RAD, users are involved throughout the prototyping phase, and as adoption is a make or break factor in this case this aspect is also very important.

Important to notice that due to the basic design processes involved, RAD products tend to be more portable, making them easier to scale and redesign. It is easier to quickly solicit and implement feedback and suggestions because of the prototyping model. If time requirements translate into money, RAD can sometimes result in lower costs.

At the same time some pitfalls need to be avoided as the applications tend to be less efficient and have more bugs. They may have reduced features and performance due to personal preferences of users involved. The professional look and feel may be lacking due to the abbreviated time spent on development as well as the potentially smaller user base consulted.

In general for the IB development context RAD tends to be a suitable approach. This is because RAD is most often suited to business and other environments in which change happens rapidly.

3 How to make RAD work in the context of IB Systems Development

It is useful to consider the main types of RAD approaches as well as the critical strategies that may be implemented to optimize the use of RAD within IBs. To start from we can begin from the main RAD forms presented by Barry Boehm (Boehm 1999) at a high level and delving specific aspects when considered for IBs.

One of the most frequently encountered forms of RAD is *dumb* RAD. DRAD occurs in general when a decision-maker sets an arbitrary short deadline for completing a software project. While this does indeed tend to happen quite often it is most destructive and should be avoided at most costs. In the IB environment however such projects do occur and people involved with them tend to have significant problems. This is why a certain level of experience will greatly help in these situations and why saying “no” is so important in this environment.

Another form of RAD involves *generator* RAD in the form of using application generators such as spreadsheets, fourth-generation languages for business or domain-specific languages for finance for ex. This form may be applied with consistent success mainly when portions of the application domain are well bounded and mature such as when using an application API for example. The main problem for GRAD is scalability as often the solutions generated do not manage to go beyond a relatively limited level of efficiency and complexity.

A third form of RAD is *composition* RAD which uses small “tiger teams” to quickly write a small to moderate application in a relatively low timeframe, say 3 to 5 months. Such applications tend to be built using application domain class libraries and large components such as networking packages, GUI builders and frameworks, database management systems and distributed middleware. CRED is more scalable than GRAD

but it is still possible to have limitations mainly in terms of competitive advantage.
Implementing full-scale RAD requires the implementation of several effective methods for reducing cycle time. When software development is considered as a network of tasks along the line of a development timeline we can determine the possible sourcing of savings along the critical path.

- Eliminate tasks
- Reduce time per task
- Avoid single-point task failures
- Reduce backtracking
- Streamlining activity networks
- Increase the effective workweek
- Acquire better people
- Transition to a learning organization

In the end it is the role of management to ensure that the right type of RAD is used for the project at hand. As in the case of DRAD sometimes that means the strength to say "no" to upper management or user community. In general if these strategies are kept in mind and considered RAD is an effective method that can be successfully used in the context of IB systems development.

4 RAD Implementation of a Gamma Hedging Spreadsheet using GRAD

An example of using GRAD is a spreadsheet based solution for an otherwise relatively complex business problem, in this case gamma hedging of index options on major indexes. The solution involved required the availability of pricing information, order/execution information, position information and needed to implement the ability to react to this information in a timely and correct way. A classic approach would have involved development or reuse at a basic level of the modules implementing this functionality, developing a client/server or fat-client approach in a relatively low language programming environment and as a result would have taken a long time and higher costs to implement. However, a spreadsheet based solution (see Fig. 3) with prices sourced directly from Bloomberg or Reuters using third party systems combined with an Excel API for the main order/execution and position keeping system implemented for the IB helped provide a quick solution that relieved workload for users and also relieved developers of the time pressure in developing the long term solution.

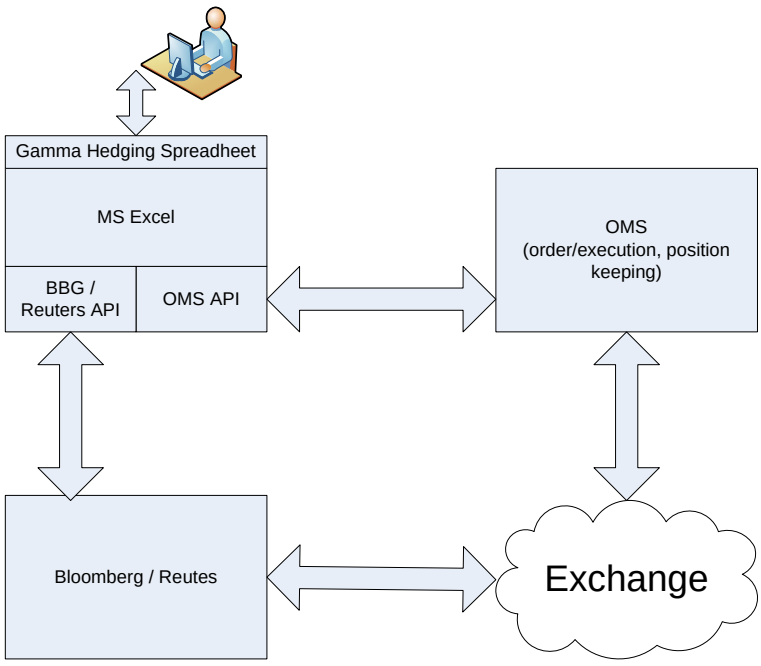

Fig. 3. System diagram for the Gamma Hedging Spreadsheet

5 RAD Implementation of a Basket Trading System

A use case for RAD is the way a Basket Trading System information system development process has been conducted by a medium sized software development company for a partner and beneficiary bank, both based in Japan.

To begin with we will use a brief description of the functionalities a basket trading system needs to implement and use:

Several types of existent events contribute to a basket trading system:

- market information events such as quote data (bid/ask/last/high/low/close),
- trade events (order placement/order cancellations/order amendments/execution fills),
- user driven events (clicking the buy/sell order button,
- changing the parameters for example the fill rate of basket portfolios,
- system events (market status, system health states, network links).

In general the actions taken by the system in response to these events include:

- split baskets in tranches (portions of baskets) before sending to market,
- send/resend baskets/tranches and/or basket/tranches remainders to market,
- cancel and replace bids/offers in the market,
- computing individual and overall exposures,
- update the latest status to the user.

Some of the features required in a basket trading system include:

- the ability to process large amounts of data efficiently without slowing other system components,
- the ability to compute basic Greek values (delta) and/or intrinsic values for large amount of stocks/futures within baskets instantly in real-time,
- accessing and processing 'low-latency' market data from exchange connectivity,
- support high volume trading such as placing tens and thousands of orders in a burst,
- an architecture to support various placement strategies,
- a responsive GUI front end for the traders monitoring and adjusting basket trading strategies, a customizable GUI allowing traders to select what they want to see and control.

The system must include safety features to avoid potential huge losses, this may include:

- the control of limits and order size,
- a panic control to withdraw all active orders in the market, or stop quoting when it detects the possible mispricing of its own baskets/tranches,
- the ability to monitor 'Greeks' and react with auto-hedging actions and warning alerts,
- ability to withdraw and place new orders that comply with exchange regulations and trading rules,
- prevention of market manipulations that may involve regulatory and disciplinary measures against the firm in extreme situations.

In the case of the implementation considered both the software company as well as the beneficiary company found themselves on unfamiliar territory given that they both had no prior experience in building or specifically using similar systems, but the user in this case did have a reasonable set of requirements compiled, based on which to execute this implementation.

In a first phase a classic software development cycle has been considered. The IB has provided a very brief functional spec, on the back of which the software company proceeded to do the development. Given that the functional spec left many things open to interpretation and that the software company did not have too much experience in building such systems, the software company ended up having to make many assumptions along the way. The result was a difficult and rather slow process with many recurrent iterations and refactoring exercises which delayed the delivery process. In the end, after a relatively

long development cycle a prototype of the system has been presented to the IB. At this stage it became clear not only that the prototype satisfied only in part the original functional specification but also that the specifications for the IB have moved on and have now developed and became increasingly more stringent, also based on the increase understanding of the IB personnel, with regards to the systems functional requirements. As a result the prototype was considered less functional than required, was not used for pilot or production, the functional spec was further detailed by the IB, and the software company went back to development based on this new specification.

In the second phased also the classic development approach was considered, this time with more detailed specs and with similarly difficult development and refactoring cycles. The second time round the result was similar in that once the second phase implementation was presented to the IB the product still did not satisfy user requirements for a variety of factors mainly involving a divergence in the fine details of the systems implementation from an end user perspective. As a result the product was again not accepted and increased frustration was observed on the IB side.

In order to reach a compromise and manage to steer the implementation on the right track the IB and the software company decided to change the development method and adopt a RAD type development. The way this was implemented included a number of pragmatic actions:

- The development company was relocated on the IB site
- IB personnel has been given access to all the development, testing and QA phases of the product
- Continuous functional spec revisions have been provided by the IB and incorporated by the development team
- The development team has been given access to most if not all of the internal processes of the IB with regards to the basket execution business
- Intellectual property rights have been shared between IB and the software development company
- Tight development and feedback cycles have been maintained for the duration of the project

The final result has been that at one stage the IB personnel became convinced that the product is an improvement to the processes and systems used previously and from that point on started to execute a portion of the business on the new system, which has become integrated in the IB flows by then. The portion of the business executed using the new basket trading system grew progressively larger until the entire IB basket trading business was moved onto the new platform, which continued to be worked on and developed at a quick pace for a number of months. This way both the IB and software company had a success story on their hands. The IB had a competitive advantage compared to other players in the market, which it maintained and held for a period of almost a year, which is a relatively long time in the industry. The software company had the intellectual property rights for a system that has become desirable by many competing IBs who wished to have it implemented.

The IB ensured return on their investment through the competitive advantage it held for a give time.

The software development company made a significant return in repeat license sales and implementations to a good number of other IBs.

Thus even if the initial costs were not balanced out for either the IB nor the software development company during the course of the project their respective returns on investment were factored at many multiples of the initial investment for their mutual benefit. Had the development method stayed the classic one of specification-implementation with relatively long periods between them as in the first two phases it is more likely that the project would have been abandoned by one party at a given point. Once the RAD method started to be used however both parties have seen a clear improvement in communication

and delivery at all levels which ultimately lead to a mutually successful outcome.

6 RAD inspired Modular Basket Trading System Architecture

An architecture diagram of the resulting basket trading system is presented in Figure 4 and shows the main components which have been used when implementing the system using the RAD method: many of the core components have been reused across the line of systems implemented by the software company (Ziman, 2000).

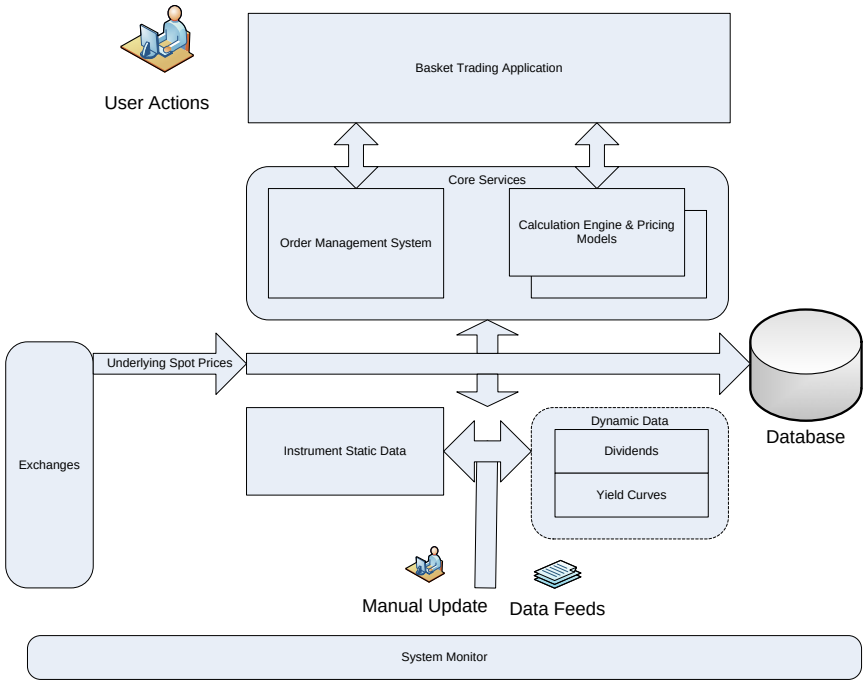

Fig. 4. Basket Trading System Architecture Diagram

In order to support the functionality of the Basket Trading System several components have been developed including:

- Basket Trading GUI – heavy client incorporating all the information across the system allowing tracking of orders execution and status monitoring, performance tracking and so on
- Order Management System – core component responsible for tracking of order/execution and position related information over the life of the orders
- Calculation and Pricing Module – core component required to calculate the theoretical values of non-cash products used for trading (such as futures in the context of basket trading). To facilitate RAD development third party available libraries have been used, these being then replaced with proprietary ones after the first successful implementation phase.
- Exchange interfaces – native exchange communication adaptors implemented for all required exchanges
- Instrument static data interfaces – sourcing static data from sources such as Bloomberg data service and Reuters for equities and futures instruments definition
- Dynamic data interfaces – storing the pricing parameters required to calculate theoretical prices for futures and forwards (dividends, yield curves)
- Market and Data feeds – sourcing real time market data from Bloomberg and Reuters

Several downstream feeds have also been provided for integration with the companies processes:

- *Risk Feed* – provides risk data to the risk department. Generally includes instrument definition, main pricing parameters

for underlying and derivatives as well as calculated Greeks (delta, gamma, vega, theta, rho).

- *Operations Feed* – Provides all the relevant information for the operations department to be able to execute any actions required after trades are executed. This generally includes all information static data as well as some pricing information such as market and/or theoretical prices for derivatives instruments and execution details (size/price) for trades.
- *Finance Feed* – Provides all the relevant information for the finance department to be able to evaluate the prices of the totality of the positions that the issuer holds related to the basket trading business. This generally includes all information static data as well as all pricing information such as market and/or theoretical prices for derivatives instruments and execution details (size/price) for trades.
- *Credit Feed* – The feed generally includes all information required to allow an estimate of the credit liabilities of the issuer.

A dedicated data model has been implemented extending the overall systems data model (which also implemented other functions for single stock trading, derivatives trading and so on) and included information on:

- instrument static data definition (underlying and warrants)
- user related information
- configuration information
- client information
- order/execution information

An overall system monitor has been implemented ensuring:

- System availability - infrastructure is working and has enough resources ex. space in the database is adequate and all processes are working
- All dependencies are accounted for – all feeds are functioning ex real time pricing, instrument data feeds and others
- All parameter calculations are working within parameters
- Order/Execution functions are available and work within accepted latency parameters

7 RAD in the context of Distributed Development for Global Implementation

In the case of large IBs it is often the case that more than one system executing a given set of functionalities exists across the multiple locations where the bank activates. As a result in these cases these systems tend to be purpose built for the given location and its specific set of challenges and requirements. It is more often than not the case that once a use/development cycle passes (usually 3 to 5 years) the IB takes on consolidation work leading to conscious efforts to remove one or more of the existing systems and attempting to standardize on a single one, as presented earlier in the introduction. Generally these cycles can be addressed in a well defined way that can also use RAD type approaches.

A possible way to achieve this is to have consciously assumed large scale projects in a single location and then in fairly short time after the first phase implementation in the chosen location, or even before that, have a 2^{nd} and 3^{rd} project team trained also using a RAD type approach, and have these teams then made responsible for the implementation of the system in their respective location. In general the first core team will continue to drive the main specification/development cycle but the local teams will be well equipped to bring their solid contribution to the project and ensure that no relevant requirements are left out for the location they are responsible for.

The RAD implementation need not necessarily mean that the project needs to be deployed with a distributed geographical infrastructure except in cases where such requirements are mandatory. For example in the case of order/execution systems global integration may be required to share market integration (ex futures trading business) but in the case of back office systems global integration is only required for product sets that require global aggregation.

8 Conclusion

This paper presents a brief description of the RAD methods used in the context of IBs trading systems development, explains in some detail what RAD methods are, what are their main features, traits, characteristics and pitfalls and how may they be used in the context. Care is given to present the main ways in which RAD differs when compared with some other methods and clearly states the advantages that RAD methods present in for IB systems development. The paper shows that RAD methods are in fact very well suited to trading systems development in particular within IBs due to the relative difficulty in determining and capturing correct system functionality details and the fact that sponsorship and adoption are two major requirements for any system offered to users, and mainly the ones involved in demanding areas of the business such as trading.

Considerations and examples are extracted from literature and contrasted with the IB context, showing that even if there seems to be a general consensus that RAD methods tend to be suitable for small to medium sized projects in fact within IBs RAD methods may well suit large sale systems as well due to the close interconnections that form between users and developers, mainly so in the case of internally developed systems. This is also one of the reasons why third party systems are mainly only considered in the case of well established market leaders where internally developed alternatives have no or little potential to provide a competitive advantage. It is clear from the authors experience that trading systems implemented within IB organizations continue to be given great importance and are being considered in many areas of the business as having the potential to facilitate or create businesses. This is especially true for front office type applications where a possible edge compared to competitors is always sought after and tends to be invested in. Due to the time-to-market criticality of such systems RAD methods emerge as a very strong and viable paradigm that IBs tend to adopt. The paper shows that where relevant distributed RAD development should also be considered to ensure consistency on a global scale within IB organizations.

References

[1] A. Ritu, P. Jayesh, T. Mohan and J. Lynch, "Risks of Rapid Application Development," *Communications of the ACM*, Vol. 43, Issue 11, 2002, pp. 177.

[2] P. Beynon-Davies, H. Mackay, R. Slack, D. Tudhope, "Rapid Applications Development: the future for business systems development?," *Proceedings of BIT96 Conference*, November 7th, 1996, Manchester Metropolitan University.

[3] H. Berger, P. Beynon-Davies and P. Cleary, "The utility of a rapid application development(RAD) approach for a large complex information Systems development ," *The 13th European Conference on Information Systems, ECIS 2004*, Turku, Finland, June 14-16, 2004.

[4] P. Beynon-Davies, "Ethnography and Information Systems Development: Ethnography of, for and within IS development," *Information and Software Technology*, Vol. 39, 1997, pp. 531-540.

[5] P. Beynon-Davies, *Rapid Applications Development (RAD),* Briefing Paper, Kane Thompson Centre, University of Glamorgan, 1998.

[6] P. Beynon-Davies, C. Carne, H. Mackay and D. Tudhope, "Rapid application development (RAD): an empirical review," *European Journal of Information Systems,* Vol. 8, 1999, pp. 211-223.

[7] P. Beynon-Davies, H. Mackay, D. Tudhope, "It's lots of bits of paper and ticks and post-it notes and ….. a case study of a RAD project," *Information Systems Journal*, Vol. 10, 2000, pp. 195-216.

[8] B. Boehm, "Making RAD work for your Project," *IEEE Computer,* March, 1999, pp. 113-117.

[9] S. E. Cross, "Toward Disciplined Rapid Application Development," *Software Technical News,* 1998, Available at: http://www.dacs.dtic.mil/awareness/enesletters/technews22-1/disciplined.html

[10] E. Elliott, "Rapid Applications Development (RAD): an odyssey of information systems methods, tools and techniques," *4th Financial IS Conference,* 1997, Sheffield Hallam University, U.K.
[11] A. Gerber, A. Van der Merwe, R. Alberts, "Implications of Rapid Development Methodologies," *CSITEd 2007*, Mauritius, November 2007.
[12] M. Hammersley, P. Atkinson, *Ethnography – Principles in Practice,* Routledge, London, 2000.
[13] J. Highsmith, *Agile Software Development Ecosystems,* Addison-Wesley, London, 2000.
[14] M. A. Hirschberg, "Rapid Application Development (RAD): A Brief Overview," *Software Tech News,* Vol. 2, No.1, 1998, pp. 1-7.
[15] J. Martin, *Rapid Application Development*, MacMillan, New York, 1991.
[16] S. McConnell, *Rapid Development – Taming Wild Software Schedules,* Microsoft Press, Washington, 1996.
[17] C. S. Osborn, "SDLC, JAD and RAD: Finding the Right Hammer," *Centre for Information Management Studies,* Working Paper, 1995, pp. 95-107.
[18] D. Tudhope, P. Beynon-Davies, H. Mackay and R. Slack, "Time and representational devices in Rapid Application Development," *Interacting with Computers,* Vol. 14, No. 4, 2001, pp. 447-466.
[19] J. L. Whitten, L. D. Bentley, K. C. Dittman, *Systems Analysis and Design Methods*. 6th edition, 2004, ISBN 025619906X
[20] I. Ziman, "Basket Trading System Implementation, Dresdner Kleinwort Japan," *Internal Documentation*, 2000.

Iosif ZIMAN is Nomura Principal Investments Hong Kong's head of technology since 2008. He has spent the past 15 years in Asia as a technology professional with a wide range of expertise across trading systems areas including order execution, risk management, operations and control areas across equities and fixed income. Mr. Ziman joined Lehman Brothers Japan in 2004 where he lead equity derivatives trading technology teams most notably implementing the suite of the company's next generation's equity derivatives structured products risk management systems. From 2000 he joined Dresdner Kleinwort Japan where he has been responsible for cash and portfolio trading technology and implemented the firm's warrant market making platform for Japan. Before 2000 he spent 3.5 years with Fusion Systems where he has been the lead for the FOX (Fusion Order eXecution) system which has been implemented by more than 15 major investment banks in Japan and Asia region (including Goldman Sachs, Morgan Stanley, JPMorgan and others) to the extent that in the early 2000's about 30% of the Tokyo Stock Exchange volumes went through the system's various implementations. Mr. Ziman holds a B.S. (1994) and a M.Sc. (1995) in Computer Science from the Technical University of Cluj-Napoca, Romania.