1 Introduction

Purpose-built trading platforms have become indispensable tools for exotic trading in most modern banks. One reason for their success and prevalence is that exotics and structured products business is highly lucrative to a financial institution. One prerequisite for success in exotic business is the ability to price the products. For pricing model development and hedging or risk analysis, the financial institution can resort to its mathematicians and the quantitative models they develop.

Another vitally important prerequisite is the capability to assess and manage the associated risks. The risks arising from market movements are related to pricing and are managed through sensitivity analyses. However, in order to manage credit and operational risks, especially when the business volume grows, banks need robust trading platforms supporting and augmenting the mathematical effort.

A carefully designed and deployed trading platform can easily recuperate its development costs within a few short months of going live, and turn into an important profit centre for banks. Once they started appreciating the necessity and profitability of an in-house trading system, banks and other financial institutions began to invest resources into developing in-house trading systems.

Resource allocation attracts professionals with highly specialized skills to move into the field. However, their skills, at times, work against them. They think of quantitative development as a purely technical endeavour. Although, at its heart, trading platform development is a technical challenge, it is the myriad of business processes around it that will eventually make or break its viability. The proverbial devil is really in the details.

1.1 WHAT IS A TRADING PLATFORM?

The goal of this book is to help design a robust, durable and reliable trading platform. The first question then is: What exactly is a trading platform (or system)? For our purpose in this book, I define it as a program or a collection of programs that meets the following broad requirements.

1.1.1 Model archival

One of the problems that the quantitative analysis effort in any modern bank faces is the archival and reuse of their pricing models. A trading platform acts as a repository for the quantitative intelligence generated within the bank. A model developed for a particular product is all too often so specialized that it is difficult to deploy it to tackle another product. This lack of reusability results in duplicated effort. A well-designed trading platform can help by imposing the right quantum of structure and standards to encourage generic programming and to enforce reusability. Furthermore, it will hold all the pricing models with standardized interfaces in one place where they can be found, examined and reused.

1.1.2 Incremental deployability

A trading platform should provide means by which the new quantitative models developed or implemented in the bank can be easily integrated and deployed into the revenue generating work streams of the bank. In other words, even after the trading platform is commissioned, as new and innovative products are developed by the mathematicians and structurers, we should be in a position to augment our system to make use of the innovation. This requirement of incremental deployability drives the whole architecture and design of the trading platform, as you will see in later chapters. It also imposes the necessary rigidity in the form in which quantitative analysts deliver their pricing models, facilitating the first requirement.

1.1.3 Live data feeds

A trading platform should talk to the external world to obtain market data (either as live feeds or as snapshots at regular intervals) and archive snapshots of market conditions for bulk processing. Market data feeds reach the trading platform from several data providers who use dissimilar, and often incompatible, technologies and interfaces. For this reason, the market data handling may become another auxiliary project in its own right, providing a uniform interface to the trading platform regardless of the origin of the data provided. This market data project often has its own database back-end for data persistence.

1.1.4 Trade persistence

A trading platform should be able to save trade data into a database and manage all the associated inception and life-cycle events. The database layer of a trading platform has many stringent requirements on security, performance and record-keeping integrity. It should be capable of handling the inception events such as cancellation, cancel and amend, reissue, novation (changing the ownership) etc., consistent with the policies of the financial institution. A cancellation, for instance, is never a database operation of deleting a record, for a complete deletion would erase the necessary audit trails. In addition, the database layer may reside in geographically dispersed locations, and the trading platform may be called upon to provide data replication services that can be challenging when the trade volumes grow.

1.1.5 Regular processing

A trading platform should facilitate and mediate all aspects of regular downstream processing, such as risk management, trade transformation, settlements, etc. This requirement, although summarized in one bullet point, is very vast in its scope, as you will appreciate by the time you finish this book.

It is instructive to benchmark a vended system (such as the trading system currently in use in your bank) against these requirements. All vended systems do an admirable job in meeting the last two requirements. However, they fail miserably in the first two. In fact, the pricing models implemented in vended solutions are the principal intellectual properties that the vendors jealously guard. They have no incentive in facilitating easy deployment of in-house custom models using their trading systems.

The pricing tool that comes with this book is near the other extreme: it implements the first two requirements wonderfully. Using the pricing tool, you can define and deploy new models without even recompiling the core program. However, it has no market data feeds, not even a clear notion of a market, and it does not provide a database layer. The only data persistence it offers comes in the form of XML files that store pricing scenarios.

Most in-house trading platforms find their place in between these two extremes. They do not try to handle every aspect of trade life-cycle management, but they are far more than the prototyping or proof-of-concept pricing programs that the mathematical brains of the bank generate. Depending on the implementation strategy of a particular bank, the in-house trading platform may end up focusing on different aspects of the requirements listed above. In all cases, however, an in-house trading platform attempts to bridge the gap between quantitative pricing models and a deployed and supported system out of which the bank can generate profit. Its emergence has engendered the relatively new domain of quantitative development. Trading platforms are the domain of the quantitative developers (computer scientists who deploy the models) in the bank, who are distinct from the so-called ‘quants’ (mathematicians who develop pricing models).

1.2 QUANTS AND QUANTITATIVE DEVELOPERS

The term ‘quant’ is short for quantitative analyst. They are mathematicians who develop pricing models and keep abreast with the cutting edge research on stochastic calculus and quantitative finance. They transform the academic knowledge they assimilate into programs that can generate revenue and profit. Quants usually have an advanced degree in mathematics, physics or other quantitative fields. Ideally, quants have a sound knowledge of markets, business and products, as well as computer science.

The input to quants' work is research and the academia. Their outputs are pricing models, often delivered as programs or functions in a library, aptly called the quant library. They may also deliver standalone pricing programs, usually as spreadsheets and add-ins.

Quantitative developers are computing professionals who make the output from the quants widely usable, often through the in-house trading platform. Their functional duties fall in between the quant output and the programs used at trading desks. Ideally, quantitative developers would have the same skill set as the quants (mathematical aptitude, product/business knowledge and computer science), but with less emphasis on mathematics and much more focus on computing and software engineering. How separated quants and quantitative developers are in terms of job functions and organizational hierarchies depends on the human resourcing strategies of the bank.

In our discussions in this book, we will assume a clean separation between the duties of quants and quantitative developers. The developers start their work from the quant library, the fruits of the quant's labour. The end point of their work is an in-house trading platform and other maintainable quantitative tools.

1.3 NEED FOR SPEED

In today's financial markets, opportunities are transient. The skyrocketing commodity prices of early 2008, for instance, generated a strong demand for cost-effective hedging. Banks that could roll out structured products to meet customized hedging requirements reaped handsome benefits from this demand. Hedging essentially caps the customer's upside exposure while subjecting them to unlimited downside risks. Because of the dizzying fall in commodity prices (especially in energy) that soon followed during the second half of 2008, the banks that could provide the hedging structures again made even more profit. Rolling out such hedging solutions on short notice to benefit from the market fluctuations of this kind necessarily calls for the agility and flexibility that only an in-house trading system can provide.

Due to such enticing profit potential, an increasing amount of assets under management gets earmarked for exotics trading. In fact, this influx of institutional investments was at least partly to blame for the wild fluctuations in commodity prices. However, the influx also underscores the importance of exotic and structured products. Coupled with this emphasis on exotics trading is the increasing sophistication of the clients who demand customized structures reflecting their risk appetite and market views.

The net result of the changing financial market attitude is the need for more mathematical modelling and speedier deployment than ever before. The need for speed, in fact, goes beyond exotics business. In spot FX trading, the time scale of profit making opportunities is measured in seconds or milliseconds. The so-called

P1: JYS
c01 JWBK444-Thulasidas May 14, 2010 18:29 Printer: Yet to come

high-frequency trading tries to capture such small, but prolific, market opportunities. High-frequency trading modes are too fast for human intervention and rely heavily on machine intelligence and algorithmic approaches. The management of high-volume tick data and the real-time decision making requirements hold a challenging fascination for quantitative developers in this field as well.

1.4 IMPLEMENTATION OPTIONS

Once we appreciate the need for a trading platform through which new products can be launched rapidly, we have a few options to choose from.

1.4.1 Outsource to vendor

We can request our vendor to custom-design and integrate the new products into our existing systems, which we are already familiar with. This approach has the attraction that the new product will be native to the existing system, assuring perfect integration, especially in the back-end. In the front-end also, the familiar look and feel of the interface will enhance usability and productivity.

Vendor development, however, tends to be heavy and slow. When we come up with an innovative product, we do not want to wait for months before we can launch it. An innovative product stays innovative only for a brief span of time.

Another disadvantage of this implementation option is that the cost of custom design can be prohibitive, especially if we demand exclusivity on the product developed. The perceived profit attraction of the innovative product can soon evaporate because of the development cost involved. The nonexclusive mode is clearly not attractive because we will see our innovation in the hands of our competitors.

Vendors may be reluctant to accept our quantitative intelligence because of intellectual property considerations. This resistance makes our in-house mathematical talent superfluous. Due to these reasons, vendor development is not the preferred route for deploying new models and products.

1.4.2 Use vendor API

One way to overcome some of the disadvantages of asking the vendor to write code for us is to use their application programming interface (API) ourselves. In principle, this approach should work well. After all, we will have full control over the intellectual property associated with our new product ideas and pricing models. Furthermore, vendor API provides the same level of integration to the downstream processing systems as the vended trading system.

In practice, this approach also suffers from many disadvantages. The vendor provided APIs tend to be incomprehensible and inflexible, which has to be expected because vendors of trading systems have no incentive in encouraging in-house development.

In addition to the shortcomings of the API, we end up battling the process issues related to the release cycles of the systems as well. The vended systems are deployed by the IT team, not by quantitative developers, and the deployment involves the vendors heavily. Thus, deploying new products through the API may still be delayed by the scheduling priorities of other teams over which the product innovators of the front office have no control.

Since the vendor API is usually complicated, it is only one or two key developers in the quantitative development team who turn out to be familiar with it. This concentration of a crucial skill results in significant key person risk to the financial institution.

Finally, vendor APIs are not cheap – after all, it is not in the vendors' interest to help us be totally self-reliant. (However, they do tout the existence of the API as a key selling point.)

Despite these disadvantages, in-house development using vendor APIs is the chosen route for a large number of mid-tier players in the financial industry. If the level of innovation in the bank is low, this implementation option may prove to be the most cost-effective one.

1.4.3 Develop in-house

For larger players in the financial markets with their armies of quants churning out new pricing models and structures, in-house development may be the only viable option. If we choose to go with the in-house approach, we (quantitative developers) can control the release schedule, resulting in a near-ideal response to the front-office demands. A well-designed in-house system can be flexible and extensible.

In-house development is rapid and responsive, although it might prove to be more error-prone than using the inflexible vendor APIs. In addition, supporting such a trading platform may turn out to be costly because of the nature of in-house development, which puts the quantitative development teams and the front-office stakeholders together. The impact of support duties on high-quality (and therefore expensive) quantitative developers can be mitigated either through sound design or through a planned support strategy. For instance, if planned in advance, the less expensive IT teams can take over a large part of daily and routine support load.

Another potential issue with the in-house trading system is a less than ideal integration with the existing settlement and risk management systems. Again, a sound understanding of the downstream systems and processes (of the kind this book is sure to inculcate) and a good design and implementation plan can help avoid nasty surprises during the integration phase.

P1: JYS
c01 JWBK444-Thulasidas May 14, 2010 18:29 Printer: Yet to come

In-house development also results in obvious key person risk on the chief software architect and the lead developers. This risk has to be managed through sound documentation requirements.

Most of the investment banks (of the pre-2008 financial meltdown era) had well-developed in-house trading platforms. This fact alone is a testimonial to the profit potential of an in-house trading platform. In fact, part of the blame for the financial meltdown can be placed on such systems, which enabled the giants to roll out a slew of new products with such rapidity that the regulatory bodies could not keep up with them.

1.4.4 Replace vended systems

One of the main drawbacks of an in-house trading platform is the ponderous and weak integration with the existing risk management and settlement systems. This drawback plagues both the input side (access to live market data, trade approval signalling, etc.) and the downstream output part (sensitivity report transmission, accounting entries, settlement triggers, etc.) In order to address this weakness, some financial institutions decide to expand the scope of their in-house platform essentially to act as the backbone of their entire trading activity, not merely for their exotics business. In this mode, the in-house platform becomes the master and other trading systems (including the vended ones) become subordinate to it.

Although risky because of its scope and ambition, such a system can enjoy all the benefits of the previous choices listed above, while subjecting the financial institution to an even higher level of key person (or key team) risk. An all-encompassing in-house system soon becomes a viable trading platform in its own right, independent of the financial institution that originally embarks on it. Once the development team recognizes this potentially lucrative independence, the risks become more significant.

1.5 CURRENT TRENDS

Any of the preceding options can be outsourced to IT consultancy firms if the bank wants to minimize the cost of maintaining a dedicated quantitative development team. Most of the advantages and disadvantages of the options apply in the outsourced approach as well. However, outsourcing brings in a few drawbacks of its own. One is the viability of the IT firm, which becomes a critical consideration. Another is the implications in the intellectual property of the bank's products, practices and processes. There may be additional regulatory issues related to the security of the information accessible to the information technology (IT) firm.

Some traditional banks do take the first approach and entrust an established vendor to deploy their product ideas. However, due to time-to-market and profitability considerations, this solution is rapidly falling out of favour. The second approach of exploiting vended APIs is still prevalent, but the lack of flexibility implicit in it will soon make it less than ideal. This approach ties the bank to a particular vendor, with all the disadvantages of the dependency involved.

The last two options are essentially variations on the same theme. The only difference is the scope of the in-house trading platform. Of all the different options listed above, the most popular seems to be the third one – tasking a team of quantitative developers to design and implement an in-house trading platform, plugging the gaps in vended systems, while not attempting to replace them. They also integrate any new products and pricing models coming out of the quant team.

1.6 TECHNICAL AND BUSINESS ASPECTS OF PLATFORM DESIGN

The technical demands of a trading platform are shared by most large-scale software projects. Among them, those important to quantitative development include maintainability, scalability, modularity, robustness, reliability, security and, of course, performance. While these features may look like the standard principles of software engineering, our own special requirements add a twist to their flavours.

The requirement of maintainability has a special significance in a financial institution, where those who design successful trading systems are in demand because in-house trading systems are a relatively recent phenomenon. They tend to move on to greener pastures, rolling out new trading platforms for other financial institutions. Faced with constant disruption, the maintenance of a trading system is always a challenge. The only way to achieve it is through rigorous documentation. Formal documentation requirements are sure to follow as regulators (both internal and external) start reviewing and auditing custom-built trading platforms, which are becoming more and more commonplace. In the absence of policy-driven guidelines, we have to rely on our own discipline to spend time on self-documenting coding practices and conscientious efforts on documentation and mentoring to ensure continuity and maintainability of our in-house trading system.

Modularity is a fundamental design principle that will aid in achieving scalability as well as maintainability. When modular components are used, the purpose of each one of them is usually singular and easily understood, which helps future developers maintain the whole system efficiently. A modular design is essential for a trading platform for an organizational reason as well. In a bank, quantitative analysts are usually organized under asset class banners. Since the trading platform is the conduit of their output, it has to optimally deploy dissimilar pricing tools, emanating from diverse groups. It therefore has to sport a uniform delivery framework and a predefined structure in which the quants can provide their pricing models and tools. Modularity thus becomes an automatic prerequisite. Furthermore, modular components can be replaced with improved versions with more functionality, thereby implementing some amount of scalability.

Scalability, while a desirable trait in any program, is an absolutely vital feature in an in-house trading platform. At the prototyping stage, a trading platform may prove its worth by booking and managing a handful of trades. However, once deployed, the number of trades soon explodes into hundreds of thousands. Unless designed with this exponential scaling in mind, a trading platform will stay as an expensive proof-of-concept exercise. The lack of scalability is a show-stopper.

The real scalability we require of a trading platform, however, is of a more stringent and demanding kind – in addition to scaling performance and capacity, we also demand extensibility. Indeed, the primary reason for embarking on a trading platform project is to deploy and monetize from our in-house quantitative pricing capabilities. Without the ability to plug in new pricing models and products, a trading platform has no *raison d'être*. The functionality of the program needs to be extended with no change (nor rebuilding) of the platform. The software accompanying this book implements such a paradigm, where new products or new models for existing products can be defined and implemented dynamically while keeping the core program untouched.

Security requirements of a trading system also are much more stringent than a standard piece of software. Quantitative developers will need to worry about end users with different levels of credentials and access control. They also have to implement secure and indelible audit trails on all crucial operations. Most of these requirements come as a surprise to quants and quantitative developers, and are often implemented as an afterthought. Understanding, at the design stage, the whole slew of features and operations that a trading platform will need to support will avert much pain and suffering down the road.

Reliability and robustness, again common enough notions in software design, take on special dimensions in our quantitative finance context. In our world, we have to worry not only about the system robustness and reliability but also about transactional integrity. For instance, booking a trade may involve a series of actions:

1. Trade is priced.
2. Confirmation is sought for booking.
3. Trade is written to the database.
4. Trade is placed in the appropriate processing queue.
5. Booking confirmation is given.

Given that these transactions involve multiple computers, usually in a client–server configuration, each of these steps and the associated communication links is a potential point of failure. How we recover from a failure and how far we need to roll back depends crucially on the robustness choices made at the outset, while designing the architecture of the trading platform.

When we dump the performance requirement into the mix, we often end up with conflicting demands on our trading platform. A system that gives us a high

P1: JYS
c01 JWBK444-Thulasidas May 14, 2010 18:29 Printer: Yet to come

level of robustness or flexibility in terms of pricing model deployment may have to sacrifice a bit of its performance.

In assessing the computational efficiency of a pricing model, every iota of performance improvement is significant. For instance, one millisecond saved in pricing a trade may sound like a ridiculously insignificant improvement to a front-office quant. However, the same model will be used to evaluate, say, the Vega scenario of a portfolio over a matrix of eight spot values and eight ATM volatilities. With a volatility pillar count of ten and a portfolio containing 500 trades, the one millisecond time saved soon balloons to over ten minutes. Imagine that, if you waste ten milliseconds on a trade using a suboptimal pricing routine, you may be holding up a downstream report by two hours every day!

The scope and impact of our trading platform may be bigger than we think when we design it. We can fully appreciate the ramifications of our design decisions only if we can see all the associated processes and scenarios in which the final program will be used. We have no alternative but to understand the whole host of business processes that are involved in trading.

1.7 IMPORTANCE OF PROCESSES

A financial institution operates through a system of checks and balances. The rationale behind the system and its processes is not immediately obvious to all involved. To the front-office professionals, for instance, many of these formal processes, implementing the so-called maker–checker paradigm, may look inane and pointlessly bureaucratic.

As a result of this plethora of processes that weigh it down, a financial institution moves like a giant juggernaut, slow and full of deliberate inertia, but purposeful and with a stunningly low error rate.

The front-office quants and quantitative developers need to understand that if the objectives of various business units seem to be in conflict with one another, it is no accident. They are, in fact, designed that way. It is through these incessant conflicts, which may have evolved into a proposer–objector scenario, that a large number of policy decisions are implemented.

The basic job descriptions of risk controllers (namely to minimize risk) and traders (to make profit) are already in conflict. All the processes that we put around these functions will have to reflect and percolate this basic conflict.

Big Picture 1.1: Conflicts by Design

The conflict-driven implementation of various policies goes well beyond the workflow of a trade or the interactions between the traders and the risk managers. Even in employee compensation schemes, we can see traces of this philosophy.

P1: JYS
c01 JWBK444-Thulasidas May 14, 2010 18:29 Printer: Yet to come

For a trader, for instance, performance is quantified in terms of the profit (and, to a lesser degree, its volatility) generated by him. This scheme seems to align the trader's interests with those of the bank, thus generating a positive feedback loop. As any electrical engineer will tell you, positive feedback leads to instability, while negative feedback (conflict-driven modes) leads to stable configurations. Thus, we have rogue traders engaging in huge unauthorized trades resulting in enormous damages or actual collapses like the Barings Bank in 1995.

We can find other instances of reinforcing feedback generating explosive situations in upper management of large corporates. The high-level managers, being board members in multiple companies, end up supporting each other's insane salary expectations. If the shareholders, on the other hand, decided the salary packages, their own self-interest of minimizing expenses and increasing the dividend (and the implicit conflict) would have generated a more moderate equilibrium.

Conflict is, in reality, a good tool for policy implementation. It is in the absence of conflict that we see spectacular failures. However, engineering a self-regulating conflict by itself does not guarantee a perfect outcome. Looking at the performance-based incentive schemes again, the incentives may be drawn from a set amount (a bonus pool, for instance). While this mode of compensation can encourage healthy compensation, the inherent conflict in it can also create unhealthy politics – you can benefit not only from your own stellar performance but also from others' poor shows.

The front-office traders taking risk and the middle-office teams managing it form a negative feedback system that is supposed to bring about a predefined equilibrium. Once we understand the philosophy behind risk taking and control processes, we can see the conflict between them, often resulting in ugly office politics, as a good thing.

Once we understand that the conflict between risk taking and control processes is designed to implement the risk appetite of the bank, we can incorporate them in the design of our trading platform in a balanced fashion, rather than satisfying only one side of the risk–reward equation. With the understanding of what needs to go into the thinking process behind the system design, we can start looking at possible strategies of rolling out a trading platform.

It is in the context of a purpose-built system that this book finds its relevance. A robust trading platform design calls for more than computer science knowledge on the part of the quantitative developers and the ability to specify requirements on the part of the rest of the bank. It demands a thorough and reciprocal understanding of the functions, roles and working paradigms of all the stakeholders in the life cycle and risk management of trades.

P1: JYS
c01 JWBK444-Thulasidas May 14, 2010 18:29 Printer: Yet to come

1.8 OBJECTIVES AND ORGANIZATION

The objective of this book is to lay down the requirements and the ideal design principles of a trading platform, especially one that is built in-house for rapid deployment of quant pricing models. Given the interdisciplinary nature of the beast that is an in-house trading platform, we will organize this book in a top-down fashion, emphasizing the big picture whenever possible. In order to encourage an appreciation for the interconnections among trade life-cycle events and the rationale behind them from this Big Picture perspective, we will pepper the discourse of the book with a series of columns, aptly called 'Big Pictures.'

A short summary of what to expect in each of the following chapters may be of interest. After this introductory chapter, we will delve into the subject matter, starting from an overview of banking in Chapter 2. Here, we will look at the organizational structure of a bank from the perspective of a quantitative developer. While there are aspects of the bank that call for in-depth treatment, they are not of direct relevance to us. We will look at only those departments and business units that affect the flow of a trade (particularly of an exotic or structured product). This chapter is meant to provide a backdrop to those of us who venture into the exciting world of quantitative finance from engineering or scientific fields. For such professionals, the main difficulty is in the finance jargon built around the organizational structure of the bank.

In Chapter 3, we make use of this overview of the banking organization and look at the life cycle of a trade. From its inception to its eventual settlement, a trade goes through a complex web of interrelated operations and processes. Each of these operations will have to be modelled and executed on the trading platform. A thorough understanding of the business processes, therefore, is the key to deploying a trading platform successfully.

In Chapter 4, we bring the two preceding chapters together and examine the trade perspectives held dear in various business units. In order to understand, anticipate and fulfil the requirements coming out of these business units, we have to understand their work paradigms and trade perspectives.

Chapter 5 represents our switch to the technical side of quantitative development. While the previous chapters deal with the business aspects in an attempt to bring a typical quantitative professional up to speed in business knowledge, this chapter does the opposite – it gives an overview of computing to business users. After the initial introductory sections, however, it goes deeper into the language and database choices open to us while designing a trading platform, and their merits and demerits.

Chapter 6 is where we put all these things together and come up with the architecture of a typical trading platform. In this pivotal chapter, we will discuss the considerations and components that need to go into the architecture. We will then complete the picture with an example trading platform design. After

P1: JYS
c01 JWBK444-Thulasidas May 14, 2010 18:29 Printer: Yet to come

describing this fully functional trading platform design, we will present another architecture that has a more ambitious scope.

Once we have looked at sample architectures, we go over some of the recurring programming patterns in Chapter 7. Patterns are a useful and effective means of implementing our design in an object-oriented framework. Since patterns and related topics are discussed in detail in specialized books on programming, we will look at only a few patterns that are commonly used in trading platforms.

Chapter 8 is a description of the design philosophy and the functionality of the program accompanying this book. Although the program, the pricing tool, is no trading platform, it does have certain traits that are desirable in a full-fledged system; in particular, its extensibility and dynamic GUI generations are strong candidates for inclusion in any trading platform. These aspects will be highlighted in this chapter, along with a full description of its other features.

Chapter 9 discusses the technical aspects of the pricing tool. Much more documentation is available to the interested reader on the CD. The discussion in this chapter also includes the gaps between the pricing tool and a real trading platform. Once these gaps are addressed (which is a considerable effort), the pricing tool can indeed become a mini trading platform.

We will sum up our discussion in the final chapter, where we will suggest further reading material and provide some food for thought on the real Big Picture, which the reader may find both interesting and inspiring.

At the end of each chapter, after summarizing the points, we will pose some questions that reiterate the main points of the chapter. Most of these questions are descriptive, rather than mathematical or technical problems. This quiz will serve to benchmark your takeaways from reading the chapter. I would encourage you to attempt the quiz, if not on paper, at least by mentally checking the points you have retained. If you find it hard to answer the questions, it may be a good idea to refer back to the text of the chapter.

QUIZ

1. Why do modern banks prefer an in-house trading system to vended solutions?
2. What are the four high-level requirements of a trading platform?
3. Describe the main development options in deploying a trading platform.
4. How do seemingly conflicting mandates of various business units (like ‘increase profit’ for the front office and ‘minimize risks’ for the risk controllers) bring about stability?