

Some Thoughts on the Implementation of Trading Systems

Christoph Bennemann and Christoph Schneggenburger

d-fine GmbH, Opernplatz 2, D-60322 Frankfurt, www.d-fine.de

e-mail: christoph.bennemann@d-fine.de

Introduction

Mathematical models are the driving force behind the seemingly never-ending evolution of financial markets, and accordingly, mathematical models are the subject of many articles in this field, the majority of them dealing with pricing and risk measurement. Few of these articles give readers an insight into the intricacies of integrating such models and the corresponding financial products into today's technical infrastructure, or even into building this technical infrastructure in the first place. And this despite the fact that without this technical infrastructure the entire evolution would come to a grinding halt.

In today's front office, trading systems constitute the backbone of the technical infrastructure, supporting traders and the processing of their deals. It is striking that although there is certainly no lack of publications on trading and risk management related subjects, few of them discuss implementing the corresponding software packages¹. This is even more surprising considering the amount of money many organizations spend on this task. The goal of this article is to at least partially bridge this gap.

But before we carry on with the article, a short definition of the term "trading system" shall be given: In this article, a trading system is a piece of software that:

- is used to capture deals of a trading desk or department,
- therefore allowing traders to keep track of their position, both in terms of absolute numbers as well as derived numbers, i.e. Greeks or P&L,
- allows risk management (and often senior management along) to moni-

tor the risk of a desk or a department, usually also yielding a break down of risk numbers to different asset classes, risk types or locations,

- quite often is also used to assure compliance of the trading operation with regulatory or internal rules, i.e. market conformity,
- and is connected to various back office and accounting systems, if it does not serve as a back office or accounting system in its own right, i.e. in the latter case implementing a sub-ledger.

We acknowledge that in some respects this is a very narrow definition for trading system, and stress that it is for the purpose of this article only.

The role of a trading system in the sense of the article is depicted in diagram 1, which shows how such software integrates into today's IT-landscape. A trading system is usually linked to many applications both inside and outside of the organization. It is apparent that a seamless integration from front to back, for which many market participants nowadays use the term straight through processing (STP), requires much more than just clever technology. In our experience, organizational issues such as well designed workflows are the key to achieving STP.

Today, most trading systems come in the form of packaged software; few organizations have the resources and the will to develop such systems in-house from scratch. This is even more so, since the products offered by the major vendors tend to be very flexible, allowing users to customize them according to their specific requirements. This article only covers the implementation of a packaged software trading system, excluding the more exotic case of an in-house developed system. The

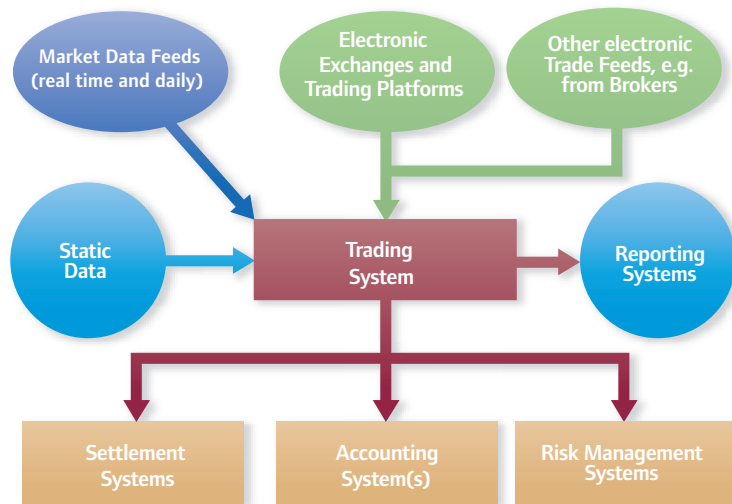


Figure 1 High level diagram of today's front-to-back office system architecture

market for trading systems has entered a mature state, with only a few vendors dominating it ¹.

In the remainder of the article a general approach to trading system implementation will be described, and selected aspects of a trading system implementation will be elucidated. Pure information technology related topics, i.e. implementation of a transaction-based interface, itself a formidable task, will not be covered.

General Approach to Trading System Implementation

(a) Initial Situation

In general there are two different cases of trading system implementations:

- A new trading system must be implemented from scratch, often replacing one or more existing systems. In recent years many organizations have chosen to consolidate their existing (best of breed) FO-system landscape due to the cost pressure, often ending up with a single system covering all asset classes.
- Much more common is the situation where functionality must be added to an existing trading system, e.g. the system must be extended to cover new asset classes.

In most cases the second task is a subset of the first; therefore, the implementation of a trading system from scratch will serve as basis for the article. Nevertheless, most of it will also apply to extending an existing system.

The first question one must address when implementing a trading system is how to organize the implementation project. The problem has two dimensions, the first one being the break down of the project in different tasks, sub-tasks, and milestones, the second is the organization of

TABLE 1 LIST OF TASKS IN EACH PROJECT PHASE

Phase	Main Tasks
Analyze	Identification of financial instruments, workflows and trade life cycle events to be covered by the system High level system architecture, including all interfaces to be build by the project Preliminary report list High level plan for data migration and testing to be carried out by the project Implementation of technical environment to be used for development Project plan, including detailed planning for design and possibly build phase
Design	Basic parameterization of system including data specification: <ul style="list-style-type: none"> ● Organizational structure and user access rights ● Portfolio/book structure ● Instrument capture ● Static data, e.g. counterparties, settlement instructions ● Specification of workflows and trade life cycle events Valuation parameter set-up, which comprises <ul style="list-style-type: none"> ● Modelling of Interest rate curves, spread curves, volatility surfaces, etc. ● Specification of P&L and risk measures Interface specification Detailed system architecture Data migration specification Final report list
Build	Building of reports, interfaces and data migration tools Implementation of technical architecture, in particular set-up of production and test servers Preparation of test phase Preparation and start of user training Data migration test System test Integration test User acceptance test (Parallel phase) Roll-Out Final data migration

the project team. The first one clearly drives to a large part the second. In line with literature, we will use the term “software process model” for it. In this part of the article we will look at software process models at project and task level before we discuss a very critical aspect of any project, which is the testing phase.

(b) Software Process Model on Project Level

There are many books and articles available on software engineering (see for example²), and in theory the software process models described there should also be applicable to the implementation of a trading system. However, the bulk of available literature deals with software development from scratch while we, as outlined in the beginning, are implementing an already existing software package. The implementation of such a complex commercial off-the-shelf software package must inevitably focus on setting up and configuring the software, e.g. setup of user access rights, as well as integrating the software with other applications. Therefore, in general, one cannot naively apply common methods of software engineering, although its software process models yield valuable insights.

From a management's point of view, we found that often it is best to divide the project into five phases: Analyze, design, build, test, and roll-out. At first sight this seems similar to the classical waterfall model from software engineering; however, the reason for this is to impose an overall structure on the project, rather than selecting a particular software process model. Such a structure is helpful for planning and managing the project, and also to communicate project goals and the schedule to users. And indeed, for individual tasks like reports or interfaces, different software process models from software engineering can be used. And it is not uncommon to use more than one process model in a given project.

Table 1 gives an overview of the different project phases. A more detailed discussion of some of the key issues will be given afterwards. Ideally, all tasks of a given phase are completed when the phase actually finishes. However, in particular with large projects this is difficult to accomplish and some overlap between different phases must be accepted.

The durations of the different phases usually depend on the project at hand. In trading system implementations project size is largely determined by the number of financial products, the scope of front-to-back functionality and the number of interfaces. Unfortunately, as already experienced by the early pioneers of software engineering, the duration of a software project can grow exponentially with its size. The reason for this is that task dependencies and therefore complexity do not grow linearly with increasing project size [3].

Often an implementation project is preceded by a system selection which will include a top-level requirements analysis. This can serve as input for the more detailed analysis project phase.

Since configuring the software is a key project task, sufficient time must be allocated to the design phase. Note in particular that most tasks of the design phase depend on how the software is configured, e.g. in order to build an interface one must know what data populates a particular data field. The high dependency between the parameterization tasks and almost all other tasks complicates and, therefore, lengthens this phase. And when later in the project the need for a change in the initial configuration is recognized, this dependency assures that a number of other project deliveries will be affected.

It is also a good idea that one person or a small team is responsible for the parameterization throughout the project. All requests for changes

in the parameterization must be assessed by this team which is in turn responsible for communicating changes to the rest of the project. This approach also assures a high degree of uniformity of the parameterization, e.g. by defining and enforcing a general naming policy.

(c) Phased Approaches

Many organizations try to address the problem of a growing project complexity by reorganising the project into a series of smaller ones, often along asset classes or their overall system architecture. This approach avoids the typical "big bang" crisis, and usually reduces project risk significantly. However, it will hardly reduce overall project complexity and duration. The reason for this is that reorganising tasks will in general not eliminate dependencies between the tasks, but at best facilitate managing these dependencies. It is nevertheless a highly recommendable approach, and Figure 2 shows an example, depicting a possible high level project plan.

Depending on the specific situation, in particular the availability of resources, different phases can also run in parallel, as shown in the example. If, for instance, has been specified, one can use this as an input to start the implementation of credit products.

(d) Software Process Models for Individual Tasks

As outlined in the beginning, a software process model can be used to break down a task, e.g. building of an interface into different sub-tasks and steps, making it easier to control and manage. Software process models are a topic of intensive discussion in software engineering, with many different models being advocated by academics and practitioners, where

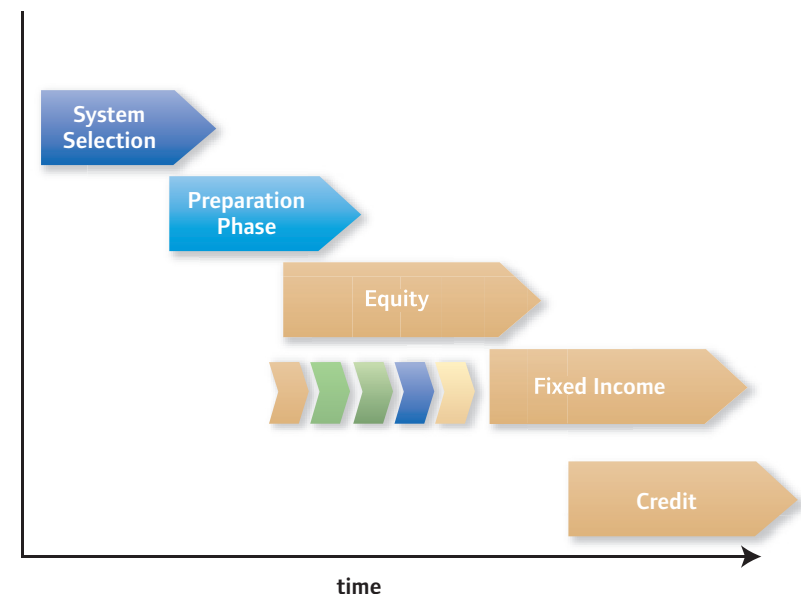


Figure 2 Example of a phased approach, where the project is broken up into a series of smaller ones; each delivering functionality for a different asset class. Each phase consists of different sub-phases (analyze, design, etc.) which is shown for the Equity phase.

TABLE 2 DIFFERENT TYPES OF TESTING A TRADING SYSTEM IMPLEMENTATION PROJECT SHOULD BE CONSIDERING

Developer Test Tests on the level of individual software functions or modules, carried out by developers. Sometimes also called unit test. These tests are rarely properly documented. For this reason it is usually difficult for project management to assess the readiness of a module.

Model Test Intensive test of all mathematical models implemented by the trading system, covering not only valuation but also derived financial quantities like Greeks. Often neglected because such tests are very tedious and customers like to believe that other customers of the software have already conducted such tests. However, in particular if the system is used by Front Office for deal analysis, such a test should be seriously considered.

A model test can be conducted by an independent team in parallel to the remainder of the project.

System Test Once all modules are finished and the trading system is properly parameterized, all workflows as well as reports of the system should be tested prior to hooking it up to other systems. Main objective of such a test is to verify that the parameterization, in particular the user rights structure, supports the workflows designed by the project team.

Integration Test The next step after a system test is to hook up the trading system to other systems and to start testing the interfaces to these systems both individually and collectively. The intention is to verify that cross system workflows and data flows function properly. Depending on the specific situation this can comprise everything from a field by field reconciliation between two systems to a verification that data can be processed in downstream systems and produce the expected results.

often the real differences between models are difficult to spot for the non-initiated.

Whereas we already stated that for the overall project several software process models can be applied, the question remains which process model to use for each individual task. In our view, there are two choices:

- The first one is to use some form of the waterfall model, allowing for some degree of iteration. This is to start a task with an analysis of the detailed requirements, afterwards write a specification, and then perform a build, e.g. writing an interface, before testing starts. This approach should be used if it is necessary to sign off the results of each step, in particular the specification, or if more than one team is going to work on the task, e.g. one team doing the design and the other the build.
- The second choice is to use a prototype-based approach where only a basic analysis of the requirements is done at the beginning. Afterwards design and build are carried out in a single step and in close interaction with users. This approach should always be chosen if it proves difficult

Data Migration Test Data migration can be one of the most complex tasks in a trading system implementation. Especially the initial upload of instrument static and trade (respectively position) data often proves to be difficult. The first step to test the upload is always to do a field by field reconciliation between source and target systems.

And since false instrument or trade data can have very nasty consequences, users frequently demand additional tests like a reconciliation of present values or Greeks.

User Acceptance Test Ultimately, it is the user who decides whether a trading system implementation will be considered a success or not. In a user acceptance test, cases are provided by future users of the system, albeit often with support from the project team, and they should also be the ones executing the tests. And not surprisingly scope and depth of a user acceptance test depends mainly on the willingness to devise such test cases.

Parallel Phase In highly critical environments, e.g. where a trading system must support many thousand trades per day, almost no amount of systematic testing will give users the confidence that the new system will fully support operations. One problem here is that in such an environment it is often almost impossible to identify all relevant tests cases. A parallel run, where trades are entered into both old and new system in parallel, can be a solution to this problem. However, a parallel run causes lots of extra effort on the user side, and therefore again will depend on the willingness of the users to engage in such an endeavor.

Another critical point here is to what extent downstream systems can be run parallel. Often it proves not feasible to set up and maintain a second settlement or accounting system over the course of a week or more.

to get detailed requirements from users or if writing a specification is uneconomical. Reports are a good example where prototyping is usually favored over a waterfall type approach.

But as often in life, nothing is black and white, and sometimes it can make sense to apply a mixed approach. In this case, a high level specification can be prepared in the beginning to reach consent on the key features. Afterwards, prototyping is applied to finish the task. In our experience this is often a sensible approach for data migration, where a field by field specification often adds little value to the project, but a high level specification describing data sources and reconciliation is needed as a starting point.

(e) Testing

Given that many trading systems are used to manage positions in the billions and to transfer large amounts of money, it seems that only fools would roll out a new trading system without extensive prior testing. However, ambitious project schedules, often in combination with an

equally ambitious project management, frequently force projects to cut back on testing. And indeed, if a project is already running late (and most projects do at some point) but project management still wants to deliver on time, reducing the scope of testing (e.g. the number of test cases) is the only option available. In our opinion this is a dangerous choice associated with many risks.

When it comes to testing, terminology can vary considerably among practitioners, and sometimes the same word can have very different meanings. Bearing this in mind, Table 2 lists a number of test phases that we feel any trading system implementation project should at least consider conducting.

Clearly, such an amount of testing requires proper up-front planning and preparation, both on a high level, e.g. in the form of a testing strategy, and on a detailed level, i.e. in the form of test cases. However, the effort for test preparation can be reduced considerably if corresponding work is organized properly. It is usually worthwhile to prepare a (or utilize an existing) test handbook in order to define a general approach, the corresponding terminology and to supply templates for documenting test objects, test cases and test results. Test handbook and templates can readily be applied by all team members responsible for test specification. Also, test specifications can be recycled for tests further up in the hierarchy, e.g. module test cases can be linked to chains in order to produce test cases for integration and user acceptance tests.

Selected Topics

In the following sections we will discuss a few selected topics in more detail. Our selection is by no means complete, but should cover some of the more critical aspects of trading system implementation.

(a) Customization

In our experience implementing trading systems can take a considerable amount of time, often exceeding one year, and once implementation starts, many trading system customers express their surprise about the sheer amount of required customization. They expect that the out-of-box functionality of the system they bought should be sufficient to support their current workflows, and all that is required are a few interfaces to some of their other systems. This is closely linked to the common misunderstanding that workflows and therefore requirements do not fundamentally differ among customers. While this certainly holds at a higher level, e.g. OTC-settlement is done very similar everywhere, a lot of differences emerge at a more detailed level. For instance what information is required to settle a trade? Who is allowed to amend a trade once it is booked? Or, how is P&L calculated, which is often very specific to an organization. Identifying these details, adapting the trading system accordingly and building the required interfaces is what requires so much time.

And many customers also fall prey to the common mistake of customising a system until it reproduces the current workflows and reports, rather than using the opposite approach that is to amend the current

workflows and reports to make them easier to implement in the new software. This usually proves to be the much simpler and more cost effective road, in spite of the initial resistance by users, which comes with any change. The identification of the key requirements is the starting point for such an approach, so that no effort is wasted on a nice-to-have functionality, e.g. reproducing the current format of a given report although a very similar report could be created using standard functionality. Asking for the added value of a specific feature can yield surprising insights.

Some vendors nowadays advocate fast implementation methodologies to speed up the delivery of the system. Such approaches usually consist of a standard parameterization of the system in combination with a bulk upload of data provided by the customer. So far we have failed to come across any such methodology delivering its initial promises, although there is some merit in this approach. Such approaches are usually advocated by vendors offering particular complex systems, which allow for a high degree of customization. Usually this flexibility, which allows users to tailor a system exactly to their specific needs, is one of the main reasons such systems are selected. Trying to impose a standard setup on such users is clearly a contradiction, and therefore can complicate things. On the other hand, having a starting point can be a huge advantage. In the end everything will depend on the specific requirements and the flexibility of the methodology. In our opinion, there is clearly a need for standardized implementation methodologies, however, they should be light-weight rather than rigid and built around standardized and proven software process models.

(b) Priorities

In any system implementation it is important to get priorities right from the beginning; that is to distinguish between critical and less critical issues. Unfortunately, prioritization from a project perspective will not always coincide with the priorities of the future users of the system leading in turn to conflicts with these future users. The main reason for this is that in technology many critical aspects remain hidden from the user's eye and therefore are not properly appreciated by them. Just think of a car where the engine and the electronics are much less visible to the user than the bodywork or the interior, and therefore the latter play a much more important role when it comes to buying a new car.

Usually data integration, namely the various interfaces to other systems that must be built, is among the most critical project tasks and therefore must have top priority on the project agenda. Without the required static data, either uploaded daily or at least once in the beginning with subsequent manual maintenance, no deal can be captured in the system, be it because the deal counterparty has not been set up or, worse, the traded instrument is missing. If the deal is not forwarded to other downstream system it settlement will most likely fail. Building such interfaces is often a tedious and lengthy task, in particular if information on up- and downstream systems is missing, which often is the case for in-house developed software.

Closely linked to data integration and therefore of equal importance

is the basic parameterization of the system. As already mentioned above, this comprises mainly the specification how various data items, instruments, counterparties, and so on, will be mapped to the data model of the trading system. This is by no means a trivial task, because usually the data model will not fit exactly the requirements and therefore needs amendment. Also part of basic parameterization is the specification of the various workflows that must be supported by the system, at the heart of this task is the mapping of the user's organizational structure to the system and the specification of the pertinent user right structure. Often additional data requirements, e.g. internal counterparty reference number required for settlement, are identified in the course of this task.

What makes things worse is that due to the high number of dependencies, interfaces and basic parameterization are often very difficult to modify once the system has been moved to production. If, for instance, the capture of counterparties is modified, all interfaces and all reports using counterparty data must be modified as well. And this applies to all data items, and often works the other way around as well, namely the modification of an interface will trigger a modification in the setup of certain data items. It is therefore usually much easier to add another report to the system or setup an additional interest rate curve than to modify an existing interface or change the way data is captured in the system.

In order to avoid such dependency problems in the course of the project, we strongly recommend to honor the details of the software process model outlined in Table 1. A signed-off list of financial instruments will serve as input for a proper workflow analysis. Both results are necessary input for basic parameterization, which is in turn required for interface design.

Unfortunately, data integration and basic parameterization are largely hidden from the eyes of the future users or, in the sense of the example used above, are located below the hood of the system. The configuration of the valuation engine and the creation of reports will sit much higher on their agenda, and they will push to make this a top priority of the project as well. While indeed the configuration and testing of the valuation engine is among the high priority tasks of a trading system implementation project, it usually does not deserve higher priority than data integration and basic parameterization, even if this is difficult to accept for the future users. But without the motor the car will not drive².

Another highly critical task, closely linked to basic parameterization and data integration, is data migration, in particular the initial instrument and trade migration. The task itself can hardly be started before basic parameterization finishes. But we found that often projects underestimate the effort required for this task. It is not sufficient to identify the source systems for the required data and to write the upload scripts, one must also address issues such as data quality, data cleansing, and how the uploaded data will be reconciled. For the latter it might be desirable to not only reconcile data on a field by field basis but also to compare financial quantities derived from that data, e.g. present value, between the old and the new system.

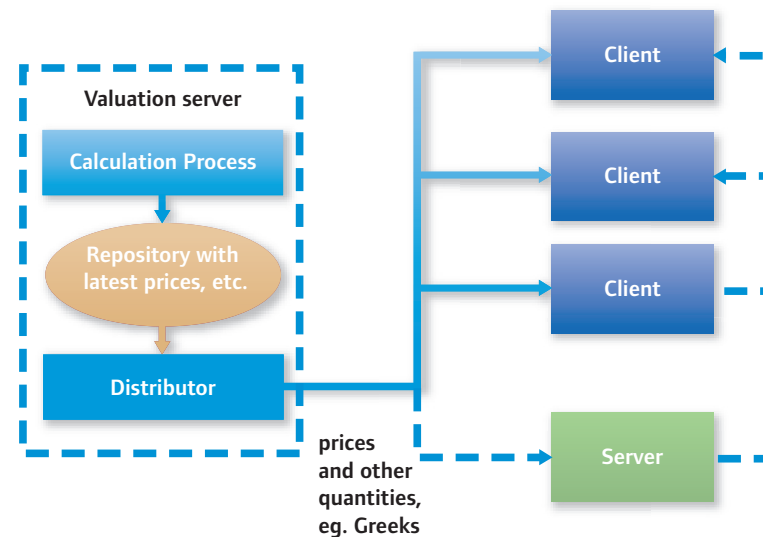


Figure 3 A valuation server consists of at least two processes running parallel, one continuously calculating new prices (as well as other financial quantities) and the other distributing these prices to clients as well as the central server of the trading system (depending on specific implementation).

(c) Model Support

Although not often mentioned during the sales process, most trading systems will not offer the latest models for some financial products. Currently many systems will be missing support for the latest kind of structured equity product, and also credit derivatives tends to be a weak area, although most top tier systems do at least offer a version of the hazard rate model. The reason for this is that software suppliers are often missing the capacity to incorporate every new hot product into their system and therefore choose only to do so if sufficient demand exists. Furthermore, they frequently do not have access to the latest models, only known to few market participants. Only when such models move into the public domain can they start incorporating them into their software.

This frequently poses a problem for organizations with an active business in such an area. They are then confronted with the choice of either managing these products outside the trading system while only capturing a place holder in the trading system, or integrating their in-house models with the trading system. The first approach imposes serious limits to any form of integrated risk reporting, which needs to be done outside the trading system. The second can become extremely complicated, when also the integration of Greeks and similar quantities computed by the model is desired, since many trading system expect to calculate such quantities numerically by shifting the underlying parameters.

Worst of all is that frequently such in-house models are extremely slow, and therefore hardly suitable for real-time reporting. One possible solution, currently discussed by many organizations, is to perform such computational intensive calculations on a separate server, i.e. dubbed valuation server, and feed the results to the clients. The basic idea behind

this idea is sketched in Figure 3. The approach can also be extended to the models of the trading system, e.g. to support real-time monitoring of large portfolios. Needless to say that such a solution can prove very difficult to implement, because a number of conceptual as well as technical issues, e.g. timing, must be resolved.

(d) Metrics

Estimating the duration of a task belongs to the most difficult parts of project planning. Two techniques can be of great use, breaking down the task into sub-tasks and using metrics. Here a metric is a quantitative measure for what needs to be accomplished in a task, e.g. the number of instruments which must be mapped in an interface, and can be used to come up with an estimate of the duration of a task. If in our example we have to map 40 instruments and it takes on average half day to map an instrument, we estimate the task to require 20 days. While this technique is not perfect, it certainly helps in reducing the margin of error. Metrics can also be applied to measure progress in a more objective manner, helping to avoid the common 80 per cent finished problem.

Typical metrics in a trading system implementation are: The number of instruments, the number of reports, the number of processes and workflows, the number of interfaces as well as the number of fields in an interface. Often it is sensible to introduce additional sub groups, e.g. complex vs. simple instruments, to make estimates more reliable.

Conclusions

Implementing a trading system is a challenging task, often taking much longer than initially anticipated. A structured project approach, such as the one outlined in this article, will help the project succeed. However, we certainly do not claim that it will assure smooth delivery, because many pitfalls await those who endeavor on such a project. We hope that the techniques and ideas discussed in this article will help others avoiding at least some of these pitfalls.

The authors wish to express their gratitude to some of their colleagues who contributed to this article, namely Egbert Schark, John Meskat, and Sven Kircher. We are also grateful to Claudia Sievers for proofreading this article.

REFERENCES AND FOOTNOTES

- [1] "Risk Technology Survey 2004", *Risk Magazine* (December 2004)
- [2] Ian Sommerville, *Software Engineering* (7th edition), Addison-Wesley (2005)
- [3] Frederick P. Brooks, Jr., *The Mythical Man-Month* (Anniversary Edition), Addison-Wesley (1995)

(Footnotes)

¹ This also applies to risk management software as most of this article is applicable to its implementation.

² But remember that nobody buys an ugly car.