Kari Kostiainen

Development of Trading Algorithm Backtest Environment

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

14 April 2016

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Kari Kostiainen Development of Trading Algorithm Backtest Environment 57 pages + 2 appendices (5 pages) 14 April 2016
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor	Ville Jääskeläinen, Principal Lecturer

The algorithmic trading in the financial markets has constantly gained popularity during recent years. The main contributors to the algorithmic trading are institutional investors such as investment companies and banks. However, the practice of algorithmic trading is available for retail investors as well.

An essential part of the trading algorithm development is an evaluation of the financial results of an algorithm. Backtesting is a commonly used evaluation method. In the backtesting an algorithm is executed with a historical market data and evaluation of the financial performance during the time period is produced.

The goal of the thesis was to develop an algorithmic trading backtesting environment suitable for a private investor. In the beginning of the project no historical market data was available. Therefore the practical part of the project was divided into two parts, the data collection part and the backtester application development part.

In the first part a system collecting market data was designed and implemented. The system is Excel based and fetches the data from an existing WinTrade application. After the system was completed it was used to collect market data for backtesting purposes. The system collected real time data from all stocks listed in the Helsinki stock exchange for a five month time period.

The second part focused on the design and the development of the Windows based backtester application. The application executes the selected algorithm with the data provided by the data collection system and produces financial analysis from the algorithm behavior. The application and developed example algorithms were tested with the collected historical data.

As a result of the thesis it is now possible for a private investor to develop and verify stock trading algorithms without commercial trading platforms. Due to the modular design the backtester application can be expanded by developing new features to it.

Keywords	algorithmic trading, backtesting

Tekijä Työn nimi Sivumäärä Päivämäärä	Kari Kostiainen Taustatestausympäristön kehittäminen algoritmiseen kaupankäyntiin 57 sivua + 2 liitettä (5 sivua) 14.4.2016
Tutkinto	Master of Engineering
Koulutusohjelma	Information Technology
Työn ohjaaja	Ville Jääskeläinen, yliopettaja

Algoritminen kaupankäynti finanssimarkkinoilla on lisääntynyt jatkuvasti viime vuosina. Pääsääntöisesti algoritmista kaupankäyntiä harjoittavat institutionaaliset sijoittajat kuten sijoitusyhtiöt ja rahoituslaitokset. Kuitenkin yksityisen sijoittajan on myös mahdollista toteuttaa algoritmista kaupankäyntiä.

Kaupankäyntialgoritmin kehittämisessä oleellinen osa on algoritmin taloudellisten tunnuslukujen arviointi. Yleisesti käytetty menetelmä on taustatestaus. Taustatestauksessa algoritmia suoritetaan käyttämällä toteutunutta historiallista kurssitietoa, jotta nähdään miten algoritmi olisi käyttäytynyt kyseisenä ajanjaksona.

Tämän lopputyön tavoitteena oli kehittää yksityiselle sijoittajalle sopiva kaupankäyntialgoritmien taustatestausympäristö. Ympäristön kehittäminen alkoi tilanteesta, jossa historiallista kurssitietoa ei ollut käytettävissä. Tästä syystä lopputyön käytännön osuus jakautui kahteen osaan.

Työn ensimmäisessä osassa suunniteltiin ja toteutettiin kurssitiedon keruujärjestelmä. Järjestelmä on Excel-pohjainen ja kerää kurssitiedon WinTrade-sovelluksesta. Toteutettua järjestelmää hyödynnettiin taustatestaustiedon keräämiseen. Järjestelmä keräsi Helsingin pörssin kaikkien osakkeiden reaaliaikaiset tiedot viiden kuukauden ajalta.

Työn toinen osa keskittyi taustatestaussovelluksen suunnitteluun ja toteutukseen. Sovellus suorittaa valittua algoritmia kurssitiedon keruujärjestelmän tuottamalla tiedolla ja laskee taloudellisen analyysin algoritmin toiminnasta. Sovellus ja kehitetyt esimerkkialgoritmit testattiin kerätyllä historiallisella kurssitiedolla.

Lopputyön tuloksena osakemarkkinoiden kaupankäyntialgoritmien kehittäminen ja testaaminen mahdollistuu myös yksityiselle sijoittajalle, ilman maksullisia kaupankäyntialustoja. Modulaarisen rakenteen ansiosta taustatestausjärjestelmää on myös mahdollista laajentaa kehittämällä siihen uusia ominaisuuksia.

Avainsanat	algoritminen kaupankäynti, taustatestaus

Contents

Abstract

Table of Contents

1 Introduction

Financial markets have been in constant development in recent decades due to advances in computer science. Computational power is harnessed in different stages during a trading process. The situation can be inspected from a retail investor, a broker company and a stock exchange point of view. Retail investors typically follow price and other information of financial instruments via computer systems, whether it be a desktop, a laptop or a handheld device. Placing transactions to brokers and further to stock exchanges takes place via same computer systems.

The broker companies utilize fully digitalized systems as well. They receive real time finance data from stock exchanges and provide it to their customers through a digitalized channel, for example a web page. In addition, broker dealers facilitate systems able to receive and process financial transactions from the customers. Certainly there is also man-power involved in supporting these systems, but when inspecting the flow of an individual transaction from the customer to the stock exchange, the chain is fully automated.

In many cases the decision making whether to sell or buy a financial instrument is still executed by a human, even if from the technology point of view it is no longer necessary. When computer software makes a sell or a buy decision the technique is called algorithmic trading. Institutional large scale investors have used algorithmic trading already for quite some time. However, among retail investors algorithmic trading is still a newer concept. Algorithmic trading offers interesting possibilities for private investors as there is a chance that the whole trading process can be fully automated with minimal human intervention.

Before setting up a computer system executing autonomous live trading with real capital, one must be extraordinary cautious and ensure that the algorithm performs as expected. One way of trying to ensure this is to execute a process called backtesting. In backtesting the algorithm is run against a historical market data and the outcome is carefully evaluated. (Chan 2013: 21)

Electronic trading platforms are available for the retail traders with various functionality. Main functionalities typically include the following: Collecting information of financial instruments, either real time or delayed. Providing analyzing tools to reveal trading opportunities. Charting support for data visualization to improve data readability. Broker company connection for the buy and sell transaction delivery and inquiry of the transaction state. Some of the trading platforms offer also support to develop custom trading algorithms and running backtest for them. Even these modern platforms offer lots of functionality in one package, there are also downsides. One major issue is a poor connectivity to the Finnish financial markets, as in many cases a broker company supported by the platform is fixed. Another issue is that some platforms require purchasing of a license, which can be substantial for a retail investor.

This proof of concept type of study concentrates on the development of the algorithmic trading backtesting environment. The testing environment is suitable to be utilized by a retail trader. Due to limitations and restrictions in the available trading platforms, a decision to develop a backtesting system from the clean table was made. The research question of the study can be elaborated as following:

How to design and implement a low cost backtesting environment where financial performance of different stock trading algorithms can be evaluated?

The thesis can be roughly divided into two sections, a background and a practical section. In the background section the basics of investment financials are introduced with special attention on the algorithmic trading. The backtesting theory, requirements and techniques are explained as well. In the practical section a process of building a retail investor backtesting environment is explained. The first step in the process is to collect historical market data. The second step involves development of a Windows application which is able to run backtesting for candidate trading algorithms. The application runs the algorithms with the collected market data and produces information about the financial performance.

The thesis is divided into the following chapters. Chapter 1 gives a brief overview of the project. Chapter 2 gives an overall picture of the algorithmic trading and its various nuances. Chapter 3 introduces the key financial concepts required to follow through the project. Chapter 4 concentrates on the financial algorithm testing theory, being also the last chapter in the background section. Chapter 5 is the first chapter in the practical part.

It describes the process of stock data collection in detail, including data collection planning, used tools and required SW development as well as the outcome of the whole process. Chapter 6 concentrates on the development of the backtest application. It lays out requirements for the backtester, reveals design and details of implemented software and also presents the final outcome. Chapter 7 discusses the project and its results, drawing conclusions and suggesting future improvements. Chapter 8 is a summary.

2 Algorithmic Trading

This chapter gives an overview of the algorithmic trading. It first discusses the electronic trading, as the algorithmic trading belongs to that domain of trading. There are several closely related terms related to the algorithmic trading, therefore a clarification of the terms is necessary. The chapter also gives a description of the high frequency trading. Finally a closer look is taken into the existing trading platforms supporting the algorithmic trading.

2.1 Electronic Trading Types

Nuti et al. (2011: 62) describes an electronic trading as any method of exchanging financial instruments such as stocks, bonds, foreign exchange and derivatives through a computer system. The electronic trading aims to bring together buyers and sellers through an electronic media in order to create an exchange.

Automated trading systems typically refers to trade execution programs that automatically submit trades to an exchange (Nuti et al. 2011: 62). Actual decision to buy or sell has already been made by some other means before passing the trades to the automated trading system for the delivery.

Algorithmic trading systems contain analysis and decision making parts. The purpose of these parts is to analyze market data and generate trading signals from it. Typically algorithmic trading requires high liquidity markets, meaning the market has willing buyers and sellers at all times during the opening hours. (Nuti et al. 2011: 62)

In the literature the algorithmic trading has multiple naming conventions depending on the author. For example, in addition to algorithmic trading Davey (2014: 4) uses terms such as *mechanical* trading and *rule-based* trading. *Black box* trading can also refer to algorithmic trading (Aldridge 2010: 58). Nuti et al. (2011: 62) mentions term *systematic* trading. In order to avoid confusion term algorithmic trading is used throughout this thesis.

2.2 Properties of Algorithmic Trading

According to Leshik and Cralle (2011: 11) the word algorithm itself describes a plan with number of steps to achieve a defined task. Basic algorithm is deterministic i.e. it results the same output values from the same input values every time. Sallberg (2010: 3) describes algorithmic trading simply as using computers to generate trading orders that are placed on the marketplace.

The same definition applies also to a closely related term - high frequency trading. Algorithmic and high frequency trading relations can be clarified by placing them into a same domain with the traditional investing. This is illustrated in Figure 1.

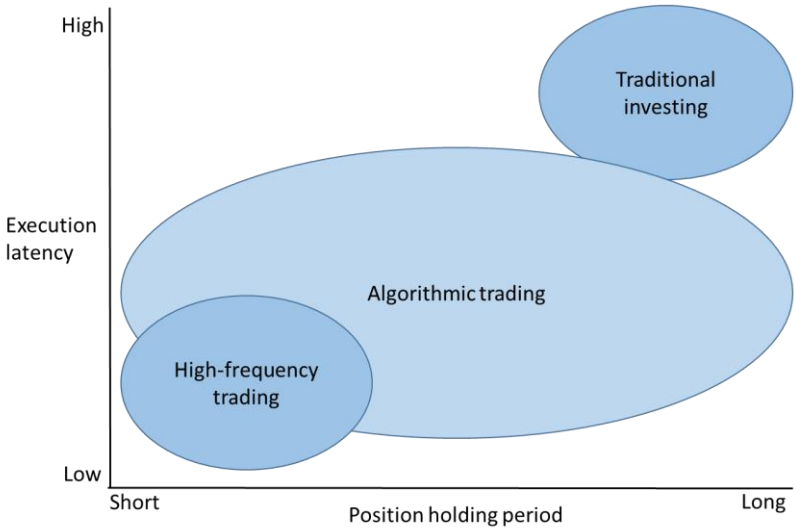


Figure 1. Relation of different trading styles (Aldridge 2010: 17)

In Figure 1 the position holding period describes how long time financial instruments are owned by the investor. The execution latency describes how quickly a trade is finalized from the trading decision. In the traditional long term investing a human is a deciding entity for buy and sell transactions. Therefore the position holding time is typically long, from days to years. However, day traders make an exception to this rule. Execution latency is typically high, at least when using traditional methods of delivering the trade orders i.e. without computer systems.

Algorithmic trading can be used for various purposes. One purpose is to carry out trade execution whereas trading decision itself is still human originated. In this case algorithmic

trading tries to minimize transaction costs. This is necessary for institutional investors that are often dealing in large quantities. Another purpose of the algorithmic trading is profit generation. These proprietary algorithmic trading systems seek to maximize profits against some amount of financial risk. Nuti et al. (2011: 63). In this thesis the focus is on the latter type of algorithmic trading.

High frequency trading is always fully automated meaning no human intervention is necessary. All decisions are performed by computer software. As a result decisions are made very quickly, which can lead to very short position holding times - typically from seconds to minutes. (Aldridge 2010: 15) Execution latencies are very short - even below one millisecond in the fastest systems.

2.3 Algorithm Categories

Trading algorithms can be divided into categories according to their intended goals. Sellberg (2010: 3) identifies two categories. *Alpha-preserving* algorithms aim to minimize market impact caused by a buy or a sell transaction. A decision to perform the transaction has been made by some other means, not by the algorithm itself. However, the algorithm takes care of the details how and when the transaction is executed. For example, a large order may be divided into smaller chunks executed over time. As for a retail investor *alpha-preserving* algorithms are not very interesting. This is because typically the traded volume is low compared to market volatility, therefore having virtually no market impact.

Alpha-creating algorithms consist of a diverse set of algorithms that by themselves try to create new value for the investment. Treleaven (2013: 78) also discusses this type of algorithms but refers to them as *Alpha model*. The term alpha can be expressed as a mathematical model designed to predict the future behavior of the financial instrument. Alpha-creating algorithms offer interesting possibilities also for a retail investor. Algorithms developed during the thesis project are alpha-creating algorithms.

Treleaven (2013: 79) continues that essentially two strategic approaches apply for the alpha model development. In *theory-driven* approach the developer selects a hypothesis for the way financial instruments are likely to behave. The hypothesis is then confirmed by modelling. In *empirical* approach the developer harnesses data mining techniques to reveal patterns in the underlying data. This approach can be applied without any assumptions of the financial instrument behavior.

Two commonly used Alpha strategies include the following: The *momentum strategy* is a trend following trading strategy. It seeks to capitalize on continuance of existing trends in the market. An assumption is that large increases in the price of a financial instrument is followed by continuing increases. The same applies vice versa for decreasing prices, a large decrease is followed by more declining. The *mean reversion strategy* relies on mean or average price levels. An assumption is that diverged prices eventually move towards long time average values. In pair trading strategy this can be applied to two financial instruments. If the instrument's price levels have been correlating historically and then separated from each other, assumption is that they will converge in the future. (Treleaven 2013: 81)

2.4 High Frequency Trading

In this chapter high frequency trading (HFT) is looked more closely into in order to draw a difference between it and the algorithmic trading type handled in the thesis. According to an asset manager survey presented by Aldridge (2010: 21) HFT can be identified from the key characteristics listed below.

Tick-by-tick data processing. This means that the decision making algorithm analyses incoming data in accuracy of one tick. Frequency of the ticks can be high, depending of the available market data system. For example, a new tick may be available in matter of milliseconds.

High capital turnover. Since HFT systems are generating trades during small fraction of time, price changes of an instrument are typically not very substantial. Therefore, a high capital investment is required in order to gain profits from the small price variations.

Intra-day entry and exit of positions. A HFT system may have really short holding times, even less than a second in some cases. But even with holding times such as few hours, the system can still be characterized as HFT system. Short holding times offer two benefits. Firstly, considerable savings of overnight position carrying costs in markets where such costs are applied. Secondly, they reduce a risk exposure resulting from the passive overnight positions, as some event may influence price drop in the night time which realizes in the morning when the market opens.

Algorithmic trading. This is a necessary component of HFT trading platforms. Otherwise it is not possible to evaluate every tick of data separated even just by milliseconds, process market information and make trading decisions in a consistent continuous manner. The algorithms, on the other hand, perform fast, efficient and emotionless decisions.

The type of algorithmic trading approached in this thesis shares some common properties with the HFT. Market data processing in accuracy of one tick is one common factor. Short position holding times and exiting the position at the end of the trading day is another. Ultimately the algorithmic trading in the thesis is still not HFT. Differentiating factors are the lack of high capital as well as longer time periods between the ticks. Also strategic approaches experienced with are not typical HFT strategies.

2.5 Trading Platforms

According to Thulasidas (2010: 1) institutional investors employing a great number of manpower often develop their algorithmic trading software in-house. The main benefit of such an approach is that the software is fully customizable. Retail investors typically do not have vast resources or the required skillset to develop their own trading software. Therefore, various trading platforms are targeted especially for the retail investors. All of them have some benefits and disadvantages. This chapter briefly discusses a couple of the most popular commercial trading platforms suitable for stock trading.

2.5.1 MetaTrader

Even the MetaTrader trading platform is mainly meant for the Forex trading, it also supports stock trading from the fifth installment. MetaTrader is developed by MetaQuotes Software and can be downloaded for free. The platform provides tools and resources allowing traders to analyse price, place and manage trades, and employ automated trading techniques. (Folger 2016)

MetaEditor is a proprietary code editor included in the platform. It supports creating, editing and compiling program source codes written in MetaQuotes Language (MQL). Customers are able to create three types of programs with the MQL. *Expert Advisors*. This type of programs are used for the automation of trading processes. The platform supports also testing and optimization of these trading algorithms. *Custom Indicators.* These

work as technical indicators which are able to analyse price activity but not able to perform actual trades. *Scripts*. Script programs are intended to perform a single execution of some action and can fulfill both analytical and trading functions. (Folger 2016)

2.5.2 NinjaTraider

NinjaTrader is a trading platform developed by NinjaTrader, LCC. The platform is intended to traders interested in the stock, futures and forex markets. The basic features of the platform can be used free of charge. Those include advanced charting, market analytics, automated strategy development, backtesting and optimization, and trade simulation. However, for live trading a license must be purchased. NinjaTrader supports variety of technologies for brokerage connection, enabling trading operations with several broker companies. Multiple market data vendors are also supported. The platform supports algorithm trading development with the C# language. (Folger 2016)

2.5.3 TradeStation

TradeStation is another full blown trading platform with a long history and multiple features. The platform provides extensive functionality for receiving real time data, displaying charts, entering and managing orders, and controlling market positions. Although the platform contains many pre-defined indicators, strategy components and analysis tools, it offers possibility to customize all of these and in addition to create new ones. This is achieved through its programming language, EasyLanguage. The EasyLanguage is an object-oriented programming language designed especially for the trading algorithm programming. (Reinkensmeyer 2015)

2.5.4 Evaluation

Modern trading platforms offer a plethora of functionalities. Some of them have been in development well over two decades, enabling to bloat them with many features. A rich set of features can be considered as their main benefit. However, the commercial trading platforms contain also downsides.

Firstly, some of the trading platforms are tied with certain brokerage companies and data vendors. Internationally small markets such as Helsinki stock exchange and related brokerage companies are not necessarily supported. Secondly, the pricing can be an issue. Some platforms offer certain basic functionality free of charge. However, in order to access historical market data or execute actual live trading, either a license or fees are to be paid. A full trading platform license costs can form a substantial part of typical retail trader budget. Thirdly, even the platforms offer customizable options and programming environment, they still place restrictions and limitations what can be done programmatically.

3 Financial Theory

This chapter introduces the applicable financial concepts while developing a trading algorithm backtest software. Financial science includes a multitude of information, terminology and concepts, but naturally only matters relevant for the thesis are introduced here. First different market order types are introduced, then the focus is on the tick data and finally some popular methods for the data analysis are explained.

3.1 Market Order Types

Financial markets support a variation of order types to meet different transaction needs. Order types include at least market order, limit order, fill-or-kill order and stop-loss order. (Glen 1994: 24)

Market order specifies quantity of financial instrument to be bought or sold immediately at the best available market price (Gregoriu 2008: 294). This order type is suitable when the price of the instrument is developing rapidly and therefore trade needs to be executed fast. An example of this can be buying of stocks just when their price starts to rise rapidly. However, for a large order in relation to the volatility, the order may be executed in parts and each part may be traded with different price.

Limit order specifies, in addition to the quantity of the financial instrument, the price at which they are bought or sold. In order to execute the order the following conditions must be realized. In case of a buy limit orders current market price needs to develop equal or lower than the specified price. Contradictory with sell limit orders current market price needs to develop equal or higher than the specified price. (Ponomareva et al. 2012: 351)

Fill-or-kill order requires that the transaction is executed immediately or the order is cancelled. The order can be filled also partially, in that case just part of the shares are traded. The purpose of this order type is to ensure that a position is entered with the desired price. Typically it is used when acquiring a large number of shares where even a slight price deviation could result in a significant difference in capital. In case the partial order fill is not acceptable, order type can be specified as *fill-and-kill*. (Aldridge 2010: 69)

Stop-loss order instructs the trading system to hold the financial instrument until its price decreases into or under a given price limit. In that case the financial instrument is sold

immediately. (Aldridge 2010: 70) Stop-loss limit is useful when an investor does not tolerate price fluctuations under a certain price limit. In other words, the stop-loss can be used to save the situation if the instrument's price suddenly falls.

3.2 Tick Data

Tick data describes price information of a financial instrument at its finest level. Tick data is time series data consisting of sequential ticks. By comparing two sequential ticks one can observe a minimum change in a price the instrument can have. This chapter familiarizes the reader with different tick data attributes.

3.2.1 Content

Each tick may typically have the following properties: financial instrument identification code, timestamp, bid and ask price, available bid and ask volume, last trade price and size. (Aldridge 2010: 116) In addition other properties may be available.

A timestamp records the date and time at which the quote originated. It may be the time at which either the exchange or the broker dealer released the quote. In HFT systems the timestamp may have an accuracy of one millisecond, as the quote travel time from the exchange or the broker to the final trading system takes some time. In lower frequency trading systems the timestamp has typically the accuracy of one second. (Aldridge 2010: 116)

A tick may have multiple fields describing price information. The bid price is the highest price available in the market while buying the instrument. Whereas the ask price is the lowest price available in the market for selling the instrument. Both bid and ask prices are provided by other market participants through limit orders. The last trade price shows the price at which the last trade for the instrument was executed. The last trade price can differ from the bid and ask prices. (Aldridge 2010: 117)

Volume related information such as available bid and ask volumes describe the amount of the instrument currently in the market. They indicate volatility during one tick and can naturally vary from tick to tick. The volume information can be used to estimate the suitable market order type and required time to fully execute an order.

3.2.2 Frequency and Quantity

Dacorogna (2001: 6) claims that the number of ticks in a liquid market during one single day is equivalent to the number of daily data samples within 30 years. New ticks are generated at an exchange with high frequency. This causes that the tick data tends to be voluminous by nature. For example, one instrument may generate tens of thousands of ticks throughout a trading day, even with a moderate tick accuracy of one second. In case the tick data is targeted for the HFT the number of ticks can be even one thousand fold. Automated trading systems typically follow a multitude of financial instruments at once. As a consequence this causes that the incoming data amount requiring analyzing and storing can be substantial.

3.2.3 Quality

In automated trading systems where tick data is the sole source of information for decision making, the quality of the data raises into an important position. Therefore, the trader must be aware that the real world tick data may contain several potential quality issues.

Dacorogna (2001: 85) discusses possible pitfalls with the tick data. Data errors can be roughly divided into human or computer originated errors. Human errors can further be divided into unintentional errors, such as typing errors and intentional errors such as insertion of dummy ticks just for technical testing. An example of the computer generated error is a failure to change significant decimal digits of a quote. To illustrate this a bid price of 1.3498 is followed by true quote 1.3505, but the published bad quote is 1.3405.

Naturally a tick data provider has a significant effect to the tick data quality. According to Aldridge (2010: 117) centralized exchanges generally provide accurate data on bids, asks, volume of trades and reasonable timestamps. However, in decentralized markets, such as foreign exchange, no market-wide quotes are available. There each dealer provides their own tick data to their end users. Therefore, while inspecting a specific quote at certain point of time, information may vary between dealers.

The tick data may contain delays. I.e. the tick is delivered to the end user's trading system seconds or even minutes after it is originally formed. At the moment when the tick is received and processed the real situation of the instrument may have already changed drastically.

Due to quality issues some automated trading systems perform tick data filtering. The filtering inspects every incoming tick and aims to clean out or correct erroneous ticks. When filtering price information, the filter can be based for example on moving averages. Delayed ticks can be identified and filtered by comparing the timestamp to current time, assuming the tick timestamp is set at the originating end. (Dacorogna 2001: 82)

3.3 Analysis Methods

Financial data typically includes certain variation. For example an instrument price is seldom totally constant, instead it contains some random looking variation from tick to tick. Sometimes the variation of the consecutive ticks takes place into a same direction, which might give an impression of a price change even the price might soon return back to the normal level. Therefore, before making any assumptions based on the data, it must be somehow processed to smoothen out the unwanted variations. In this chapter such methods are introduced. Furthermore, methods called trend forecasting and risk analysis are presented.

Moving averages are perhaps the most commonly used methods to smoothen out incoming data. They are applied widely in different fields of engineering. Common for all moving averages is that they can be though as windows with certain length sliding over the data set. When the average is calculated the window is moved onwards by one step causing the last sample to drop out and a new sample to step in. Then the calculation is repeated. Hence, moving averages are also known as sliding windows. There are different kinds of moving averaging techniques. They are mostly separated by the way how the sample points are emphasized, or weighted, in the calculation.

3.3.1 Simple Moving Average

This method takes a number of consecutive observations called points and calculates their average value. Then this group of numbers is moved forward, dropping the last value and adding a new value to the group. Then the average is calculated again with the new values. (Humphris 2014: 203) A simple moving average formula is presented by equation 3.1.

$$SMA = \frac{1}{k}\sum_{n=0}^{k-1} x_{t-n} \tag{3.1}$$

In the formula k describes number of last observations, x observation value and t latest observation time. The number of observations determines the smoothness of the moving average. With low number of observations the average will follow closely the original values. Whereas with large number of observations the short variations, in relation to the observation period length, are cleaned out efficiently.

From the performance point of view simple moving average calculation can involve a lot of iterations. However, when the average value is once calculated, consecutive calculations can be shortened dramatically. In that case new average value must be calculated at the same time while removing the oldest data point and inserting a new data point. The optimized calculation of the simple moving average is presented in equation 3.2.

$$SMA_{new} = SMA_{prev} - \frac{1}{k}x_{k-1} + \frac{1}{k}x_t \tag{3.2}$$

In the formula SMA_{prev} represents previous simple moving average, x_{k-1} value of the removed oldest data point and x_t value of the incoming new data point.

3.3.2 Weighted Moving Average

Problem with the simple moving average is that it gives equal importance to each sample point. In many applications this is not desired behaviour as the latest data points are often considered to be more important. In finance, for example, the latest sample points can be thought to reflect direction of the price movement more effectively than the past samples. The weighted moving average is a solution to this problem. When utilizing the weighted moving average one must first determine weight factors $\{w_0, w_1,\ldots, w_{k-1}\}$ for each of the sample points. When summing up the weight factors the result shall equal to one. Then the weight factors are applied to the formula calculating the weighted moving average. (Chiulli 1999: 240) The formula is presented in equation 3.3.

$$\sum_{n=0}^{k-1} w_n = 1 \qquad WMA = \frac{1}{k}\sum_{n=0}^{k-1} w_n x_{t-n} \tag{3.3}$$

Weighted moving average calculation cannot be as easily optimized as the simple moving average calculation. This is due to the fact that each data point must be multiplied by a potentially unique weight factor. When removing the last data point and inserting a new one, all data points shift their position and thus a complete re-calculation is required.

3.3.3 Simple Linear Regression

While the moving averages are used to clean out the data from irregularities, sometimes it is necessary to see the direction of a change from a given data set. In finance interesting information is typically the direction of a price change. One method to accomplish this is fitting a straight line through the given set of data points so that the distance from each data point to the line is minimized. This is called a simple linear regression.

Yan (2009: 4) describes three goals for the regression analysis. Firstly, establish a casual relationship between dependent variable *y* and independent variables $x_1, x_2, \ldots, x_n$. Secondly, predict *y* based on a set of values $x_1, x_2, \ldots, x_n$. Thirdly, screen variables $x_1, x_2, \ldots, x_n$ to identify which of them are more important than others to explain the dependent variable *y*. From the algorithmic trading point of view the second goal seems most interesting. Given that the y would be the price of a financial instrument and x variables would present points in time. Yan (2009: 9) continues by explaining a typical model for the simple linear regression, presented in equation 3.4.

$$y = \beta_0 + \beta_1 x + \varepsilon \tag{3.4}$$

In the model *y* is the dependent variable, β_0 is the *y* intercept, β_1 is the slope of the simple linear regression line, *x* is the independent variable and ε is the random error. Challenge arises from determining the values β_0 and β_1. There are several methods to accomplish this, but perhaps most commonly used method is called the least squares estimation. According to Yan (2009: 10) the goal of the estimation is to "find the estimates β_0 and β_1 such that the squared distance from actual response y_i and predicted response $\hat{y} = \beta_0 + \beta_1 x_i$ reaches the minimum among all possible choices of regression coefficients β_0 and β_1". In other words, the aim is to find a straight line that is closest to all data points. The least squares method for the simple linear regression is expressed in equation 3.5.

$$(\beta_0, \beta_1) = arg\,min \sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_i)]^2 \quad (3.5)$$

3.3.4 Sharpe Ratio

Profitability and risk of trading strategies can be estimated by the Sharpe ratio, a risk adjusted return metric proposed by William Sharpe. The Sharpe ratio indicates return per unit of a risk. For example Sharpe ratio of 2 means that the average return on the strategy exceeds twice the standard deviation of the strategy returns. The Sharpe ratio also hints distribution of returns. (Leshik and Cralle 2011: 46) Negative Sharpe ratio indicates the strategy is generating loss and the positive value indicates profit. As a rough estimation of the Sharpe ratio values 1 and above can be considered good, 2 and above really good and 3 and above exceptional. The Sharpe ratio can be calculated with equation 3.6.

$$SR = \frac{E(R) - R_f}{\sigma} \quad (3.6)$$

In the equation $\mathrm{E(R)}$ denotes the expected return on an investment, R_f is the risk-free rate of return and σ is the standard deviation of the returns. (Gregoriu 2008: 303) When applied to the algorithmic trading $\mathrm{E(R)}$ equals average profit of the observation period. The risk free rate is a theoretical concept in a sense that no investment is completely without of a risk. The standard deviation describes the risk part in the equation as more wide spread the returns are, more risky the investment. Formula for the standard deviation is given in equation 3.7.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2} \quad (3.7)$$

In the equation μ is average value of samples, x_i value of one sample and N number of samples. Sharpe ratio comes in handy when comparing investments alternatives. In the context of the algorithmic trading this means comparison of different trading algorithms, or strategies. For example, two trading strategies may generate almost equal returns but

clearly contradicting Sharpe ratios. The strategy with higher Sharpe ratio can be considered as the better option due to better risk profile, even its returns would be somewhat lower.

4 Trading Algorithm Testing

Software testing is an important, sometimes even underestimated, part of the software development. Algorithmic trading software makes no exception. In addition the fact that capital is at stake emphasizes the correct system behavior and therefore testing importance. In this chapter testing methods for a trading system development are discussed.

Naturally as the question is about software all traditional software testing methods apply also to the algorithmic trading system development. Such testing methods are for example unit testing, module testing, functional testing and system testing. Complexity and modularity of the developed system many times dictates the number of needed testing stages. However, all these testing steps can only verify that the trading software works as it was specified. They are not capable of proofing that the system is performing well financially and generating positive return of the investment.

Several methods exist when trying to verify the profitability of a trading system. Davey (2014: 47), as well as other authors, identifies four methods: historical backtesting, out-of-sample testing, walk-forward analysis and real-time analysis.

4.1 Historical Backtesting

In the historical backtesting the trading algorithm under evaluation is fed with a real market data from the past. The process tries to reveal how the algorithm would have performed if it would have been running on the live market during the given time period. This information can then be reflected to the future with a hope that the algorithm will perform in the similar manner. (Chan 2013: 20)

Treleaven (2013: 79) states that “While backtesting does not allow one to predict how a strategy will perform under future conditions, its primary benefit lies in understanding the vulnerabilities of a strategy through a simulated encounter with real-world conditions of the past.”

Historical backtesting may contain algorithm optimization with in-sample data. In such a case the algorithm will contain so called free parameters whose pre-determined values

can be changed. (Davey 2014: 48) An example of such parameters can be the moving average window length. Amount of the free parameters can vary greatly between algorithms.

The backtesting may have pitfalls. In that case difference between the backtest results and the real market performance can be huge, typically in an unfavorable way. Hence special attention is to be paid to avoid the pitfalls. Chan (2013: 22) describes most importantly two pitfalls: a look-ahead bias and a data snooping bias.

The look-ahead bias means that the backtest program is using future information to determine current trading signals. Such an error could happen for example when a trading algorithm would use day's high or low price to determine entry signal for the same day. Problem is that day's high and low prices are not known before the trading day has ended. Another example would be to use indexing that points to the future data. Essentially, look-ahead bias can be thought as a programming error. One way to avoid look-ahead bias completely is to use the same program to run the backtesting and the live trading. In that case a difference between the two is in a way how the market data is fed to the program. During the backtesting historical data is fed to the program, whereas during the live trading it is the current market data.

The data snooping bias, also known as curve fitting or over optimization, occurs when a trading algorithm has too many free parameters that are fitted to match with historical market patterns. It is unlikely that exactly such patterns occur in the future, thus making backtesting results look too promising. The data snooping bias can be avoided by implementing simpler trading algorithms without excessive amount of free parameters. In addition linear trading models avoid the problem better than non-linear trading models. However, typically trading algorithms still contain the data snooping bias to some extent. Therefore, testing methods to avoid the data snooping bias are required.

4.2 Out-of-Sample Testing

Davey (2014: 49) explains out-of-sample testing as period in the backtesting where the algorithm is fed with data not used in the algorithm optimization. Testing with out-of-sample data tends to produce results describing real life market performance better than testing executed with in-sample data. In case the optimization is not used at all then backtesting is naturally out-of-sample testing. However, this is seldom the case as while

performing the backtesting there is a great temptation to fiddle with parameter values to see if the results would be better, and that is counted as an optimization. On the other hand, some very simple algorithms can be without free parameters, eliminating possibility for the optimization completely.

4.3 Walk-Forward Analysis

The walk-forward analysis measures the financial performance of a trading system solely on the basis of out-of-sample data, but it also includes an advantage of the optimization. The out-of-sample data offers far more reliable estimation of the real market performance than the in-sample-data. Basically the walk-forward analysis comprises of two alternating parts. The first part is in-sample-testing used to produce the optimized parameters. The second part is out-of-sample testing where an algorithm with the optimized parameters is fed by unforeseen market data. (Pardo 2008: 237) The walk-forward analysis is illustrated in Table 1.

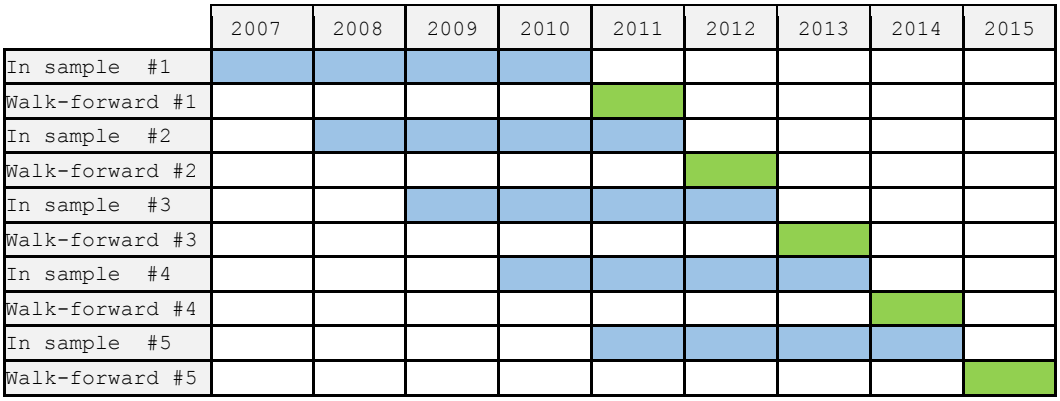


Table 1. Walk-forward analysis example

In Table 1 in-sample period is four years and walk-forward period is one year. The length of the periods can be freely chosen. First the in-sample period is executed in the backtester software, producing optimal parameters for the first four years. Then using the same parameters a one year out-of-sample period is executed and financial performance is measured. After that the process repeats itself with newer data, until the desired number of walk-forward periods are executed. Finally when aggregating the walk-forward periods they form a continuous time period that reflects the real life market performance much better than the in-sample periods. (Davey 2014: 51)

4.4 Real Time Analysis

In the real time analysis the trading algorithm receives data directly from the live markets. The system may either use real money when the transactions take place in reality or the transaction can be simulated when no real money is involved. For reasons mentioned in the paragraph below the real time analysis is typically executed as a last testing phase before going to live. It is not suitable for the early stages in the algorithm development, with possibly multiple trial and error type of adjustments. (Davey 2014: 52)

Benefits of the real time analysis include the following. Potential biases in the algorithm are effectively eliminated. The look-ahead bias is eliminated because the data has never seen before. The data snooping bias is eliminated for the same reason. This analysis method is also usable when no pre-gathered collection of the market data is available. Greatest disadvantage of the analysis is naturally the execution speed, as it follows the market speed. For example, testing of one month time period consumes one month also in reality.

5 Data Collection

This chapter presents background for the data collection process as well as its design and implementation. Execution of the backtesting requires availability of real life historical market data. Such data can be accessible through different media, for example web sites or even in financial newspapers. Financial instrument daily closing prices are freely available for anyone. However, when working with relatively high frequency algorithmic trading aiming for intraday transactions, daily prices are not enough.

Backtesting of fast paced algorithmic trading requires fine granularity data. The best alternative is to have real time tick data. The problem is that such detail and high voluminous data is not publicly available. Basically there are two alternatives to acquire the data: either purchase a historical data set or implement a data collection system. In this thesis the latter alternative was chosen, the data collection.

5.1 Nordnet

As the data collection is implemented on top of services provided by Nordnet, the company deserves its own introduction. In this chapter a short overview of the company and its services is given. The focus will be especially in trading applications included in the company's portfolio of available tools for the end users.

5.1.1 Company Overview

Nordnet is a Nordic online broker company. The company operates in Finland, Sweden, Norway and Denmark. Their tradable financial instrument selection includes at least equities, equity rights, warrants, options and mutual funds. Trading through Nordnet requires an active trading account. An account holder is able to perform trades through company's web site. Moreover, the company offers a selection of trading applications for easing up the trader's daily activities and allowing customized trading experience.

5.1.2 Trading Applications

The trading applications include mobile application for iOS, Android and Windows Phone mobile operating systems. The full featured trading applications include Webtrader, Win-Trade and Infront. With the applications it is possible to view financial data and perform trading operations. The mobile application is targeted for an audience with a need to stay alert of the markets and perform transactions on the move. Webtrader runs on top of a web browser. It offers more advanced views than the regular trading website of Nordnet. Webtrader and is practical for investors who need advanced trading application features to be available from any computer, without installing any application. Wintrade is a full blown Windows trading application offering advanced market information for stock markets and is targeted for active traders. Infront is also an installable trading application. Compared to Wintrade benefit is that it offers a wider selection of financial instruments. (Nordnet 2016)

5.1.3 Selected Trading Application

When planning real time data collection for stock markets, a choice needs to be made which application will be utilized. In this competition between applications, Wintrade was chosen due to following reasons. Firstly, Wintrade offers an option to continuously export data to an Excel spreadsheet. Secondly, it offers real time data from stock markets. Thirdly, a monthly fee of a Wintrade account is considerably more affordable than Infront account.

5.1.4 nExt API

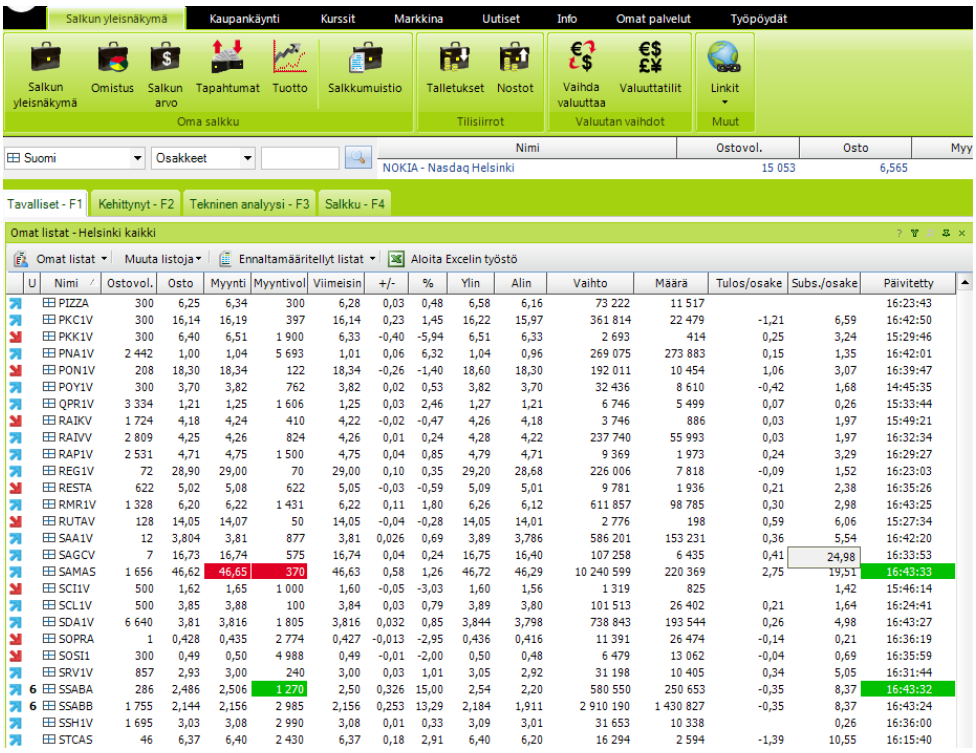
Nordnet also offers Nordnet External Application Programming Interface (nExt API) making possible to attach the trader's own software systems with Nordnet's services though the internet. nExt API would be a perfect match from the algorithmic trading point of view since it would enable autonomous software based trading. Unfortunately nExt API is not yet available in Finland when writing the thesis. Otherwise it probably would have been selected for the data collection as it would offer a clear advantage. That being later on the backtesting software could have been further developed towards live algorithmic trading utilizing the nEXT API.

5.2 WinTrade

This chapter takes a closer look into the WinTrade application. The application contains lots of functionality feasible for an active trader. However, here the focus will be mainly on available tick data in the application.

5.2.1 Overview

Wintrade is Windows based trading application provided by Nordnet and it offers multiple features for stock investors. The trader is able to view stock market information and trade stocks in several markets. At the time being the markets are Finland, Sweden, Norway, Denmark, Germany, Canada and USA. The application offers graphs with different styles for stock information illustration. Price information accuracy is presented in Nordic OMX Equity Level 2, consisting of top five levels of aggregated bid / ask depth for each stock. In addition the application provides financial news feeds from multiple news providers, different kind of lists including daily top lists of stocks prices, price alerts, various stock sorting options, etc. The application has multiple customization options how the information is presented. One of such options is to define and configure several desktops inside the application. Useful feature is also an ability to configure custom stock lists. (Nordnet 2016) Typical Wintrade screen is presented in Figure 2.



U	Nimi	Ostovol.	Osto	Myynti	Myyntivol	Viimeisin	+/-	%	Ylin	Alin	Vaihto	Määrä	Tulos/osake	Subs./osake	Päivitetty
	PIZZA	300	6,25	6,34	300	6,28	0,03	0,48	6,58	6,16	73 222	11 517			16:23:43
	PKC1V	300	16,14	16,19	397	16,14	0,23	1,45	16,22	15,97	361 814	22 479	-1,21	6,59	16:42:50
	PKK1V	300	6,40	6,51	1 900	6,33	-0,40	-5,94	6,51	6,33	2 693	414	0,25	3,24	15:29:46
	PNA1V	2 442	1,00	1,04	5 693	1,01	0,06	6,32	1,04	0,96	269 075	273 883	0,15	1,35	16:42:01
	PON1V	208	18,30	18,34	122	18,34	-0,26	-1,40	18,60	18,30	192 011	10 454	1,06	3,07	16:39:47
	POY1V	300	3,70	3,82	762	3,82	0,02	0,53	3,82	3,70	32 436	8 610	-0,42	1,68	14:45:35
	QPR1V	3 334	1,21	1,25	1 606	1,25	0,03	2,46	1,27	1,21	6 746	5 499	0,07	0,26	15:33:44
	RAIKV	1 724	4,18	4,24	410	4,22	-0,02	-0,47	4,26	4,18	3 746	886	0,03	1,97	15:49:21
	RAIVV	2 809	4,25	4,26	824	4,26	0,01	0,24	4,28	4,22	237 740	55 993	0,03	1,97	16:32:34
	RAP1V	2 531	4,71	4,75	1 500	4,75	0,04	0,85	4,79	4,71	9 369	1 973	0,24	3,29	16:29:27
	REG1V	72	28,90	29,00	70	29,00	0,10	0,35	29,20	28,68	226 006	7 818	-0,09	1,52	16:23:03
	RESTA	622	5,02	5,08	622	5,05	-0,03	-0,59	5,09	5,01	9 781	1 936	0,21	2,38	16:35:26
	RMR1V	1 328	6,20	6,22	1 431	6,22	0,11	1,80	6,26	6,12	611 857	98 785	0,30	2,98	16:43:25
	RUTAV	128	14,05	14,07	50	14,05	-0,04	-0,28	14,05	14,01	2 776	198	0,59	6,06	15:27:34
	SAA1V	12	3,804	3,81	877	3,81	0,026	0,69	3,89	3,786	586 201	153 231	0,36	5,54	16:42:20
	SAGCV	7	16,73	16,74	575	16,74	0,04	0,24	16,75	16,40	107 258	6 435	0,41	24,98	16:33:53
	SAMAS	1 656	46,62	46,65	370	46,63	0,58	1,26	46,72	46,29	10 240 599	220 369	2,75	19,51	16:43:33
	SCI1V	500	1,62	1,65	1 000	1,60	-0,05	-3,03	1,60	1,56	1 319	825		1,42	15:46:14
	SCL1V	500	3,85	3,88	100	3,84	0,03	0,79	3,89	3,80	101 513	26 402	0,21	1,64	16:24:41
	SDA1V	6 640	3,81	3,816	1 805	3,816	0,032	0,85	3,844	3,798	738 843	193 544	0,26	4,98	16:43:27
	SOPRA	1	0,428	0,435	2 774	0,427	-0,013	-2,95	0,436	0,416	11 391	26 474	-0,14	0,21	16:36:19
	SOSI1	300	0,49	0,50	4 988	0,49	-0,01	-2,00	0,50	0,48	6 479	13 062	-0,04	0,69	16:35:59
	SRV1V	857	2,93	3,00	240	3,00	0,03	1,01	3,05	2,92	31 198	10 405	0,34	5,05	16:31:44
6	SSABA	286	2,486	2,506	1 270	2,50	0,326	15,00	2,54	2,20	580 550	250 653	-0,35	8,37	16:43:32
6	SSABB	1 755	2,144	2,156	2 985	2,156	0,253	13,29	2,184	1,911	2 910 190	1 430 827	-0,35	8,37	16:43:24
	SSH1V	1 695	3,03	3,08	2 990	3,08	0,01	0,33	3,09	3,01	31 653	10 338		0,26	16:36:00
	STCAS	46	6,37	6,40	2 430	6,37	0,18	2,91	6,40	6,20	16 294	2 594	-1,39	10,55	16:15:40

Figure 2. Screenshot of Wintrade application displaying real time stock data

A real time stock market service comes along with the Wintrade service. In practice the real time service means that Wintrade displays stock information as soon as possible after the tick is formed in the exchange. For less liquid stocks information can be updated only a couple of times per hour, whereas for most liquid stocks that are constantly traded information may update several times per second.

5.2.2 Data Export Functionality

From the data collection point of view the most interesting feature in Wintrade is continuous data export into an Excel spreadsheet. Wintrade copies all stock information displayed in its main window once per second to the selected spreadsheet. This process always overwrites the previous data. Some data granularity is lost in the process, as Wintrade is able to display market data in real time but the data copying operates only once per second. The data export feature itself is straightforward, but its true value is revealed after the trader starts to implement own Excel macros processing the data.

5.3 Data Collection Design

This chapter can be seen as the start of the practical part in the thesis. As the background and starting point of the data collection is introduced, it is time to dive into the details of the developed data collection system. First an overview of the data collection is presented and then the data collection requirements are discussed.

5.3.1 Overview

The collected stock data will be later on processed by the backtester application. Therefore the data shall be easily readable by a computer program but also a human friendly presentation is required. Overview of the data collection system is presented in Figure 3.

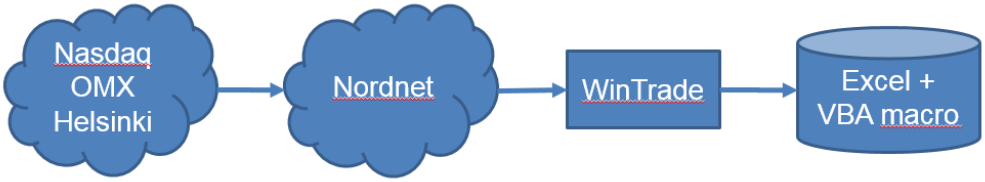


Figure 3. Data collection setup overview

Starting from the left hand side, tick data is originated from the Helsinki stock exchange, also known as Nasdaq OMX Helsinki. For time being this market has 134 publicly listed stocks available for trading. The plan is to gather tick data of all available stocks. Buying and selling shares takes place in computer systems of the exchange. Nordnet operates as a broker company in the middle between the exchange and the retail client. From the tick data collection point of view the broker company's sole purpose is to deliver high quality tick data to the retail client. WinTrade application is running on the retail client's computer and feeding the tick data periodically to the Excel workbook. The workbook reacts to these changes and triggers macro functionality which further stores the tick data into a desirable format.

5.3.2 Requirements

The software requirements for the data collection system are listed below.

The data shall be in a format easily accessible from the backtester application. The backtester application operates in the Windows environment and it will be developed with C# language. An application written in C# language is able to read data conveniently from Excel spreadsheets by importing the Excel library. Another possibility is to convert the collected data into a more generic Comma Separated Values (.csv) file format.

The data shall be human readable. The rationale for this an ability to verify correct behavior of the backtester by comparing its buy and sell decisions in relation with the input data. Therefore the ability to manually quickly find tick data of certain stock at any moment in time is important. Excel spreadsheet suits well for this purpose.

Depicting the tick data. While designing a trading algorithm, drawing a chart of any tick property, for example the last sell price in relation with time, can become an essential part of the process. Data collection shall support the possibility to quickly draw charts of any tick data property. Excel provides a comprehensive and easily deployable set of various charting tools.

Chronological data. Ultimately the collected historical tick data is read by the backtester. It is essential that the backtester drives current trading algorithm under evaluation with chronological data in order to avoid different biases discussed earlier. The algorithm must make all its decisions based only on the current and the passed ticks. Therefore,

the backtester needs to access the data chronologically. For performance reasons the backtester shall not need to sort the incoming data. To solve this, one Excel spreadsheet shall contain tick data of all stocks in the chronological order.

Data collection effortlessness. In an ideal case the data will be collected from each trading day. Dasdaq Helsinki trades from Monday to Friday. The data collection setup shall be quick and straightforward to set up running. Preferably it shall be something along the following lines: Turn on the computer, start Wintrade and login, click the Excel import functionality start button and select the correct spreadsheet.

Minimization of data conversion steps. The data flow can be thought to take place from the Wintrade application being the data source to the backtester application being the data sink. In this process the minimum number of data conversion steps is desirable as those are extra effort and increase the risk of accidental data manipulation.

Data storage size. Real time tick data is voluminous by nature. Even more so if redundant data is saved. Wintrade copies the tick data of all stocks to the Excel once per second, whether a timestamp of an individual tick has changed or not. Excel shall perform a validity check for the tick data based on the timestamp. Tick data shall not be saved if it already exists with the same timestamp.

Excel spreadsheet row limitation. An Excel 2013 spreadsheet can contain a maximum of 1 048 576 rows. This limitation can be reached in matter of days as the following example demonstrates. Most liquid stocks may generate a new tick almost every second and the exchange is running from 10:00 to 18:30 resulting in around 30k ticks for such a stock alone. Including multiple stocks with high liquidity and all other stocks, the Excel row limitation can be reached quickly. Therefore, it shall be a minimum effort process to switch data collection from one Excel workbook to another.

5.4 Excel Workbook

The data collection requirements presented in the previous chapter result into the following tick data collection implementation. Tick data is imported from Wintrade to an Excel workbook. The workbook contains multiple spreadsheets, illustrated in Figure 4.

A

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Nimi	Ostovol.	Osto	Myynti	Myyntivol	Viimeisin	+/-	%	Ylin	Alin	Vaihto	Määrä	Tulos/osa	Subs./osake	Päivitetty
2	ACG1V	10	1,06	1,1	3064	1,1	0,01	0,917431	1,1	1,04	15594,95	14536	-0,31	1,66	19:00:00
3	AFAGR	1300	0,42	0,42	3000	0,42	-0,02	-4,54545	0,435	0,4	36507,07	87240	0,01	0,68	19:00:00
4	AFE1V	700	2,92	2,96	197	2,93	-0,01	-0,34014	2,97	2,93	17331,76	5857	-0,07	2,69	19:00:00
5	AHL1V	170	7,35	7,39	255	7,39	0,04	0,544218	7,41	7,34	24281,61	3300	0,08	6,86	19:00:00
6	AKTAV	9857	10,39	10,39	13211	10,39	0,24	2,364532	10,39	10,1	486589,1	47772	0,79	9,37	19:00:00
7	AKTRV	50	10,87	12	133	0	0	0	0	0	0	0	0,79	9,37	19:00:00
8	ALBAV	30	16,33	16,5	969	16,5	0,2	1,226994	16,5	16,5	1171,5	71	1,12	13,59	19:00:00
9	ALBBV	200	15,55	15,6	40	15,6	-0,11	-0,70019	15,71	15,6	8515,94	544	1,12	13,59	19:00:00
10	ALN1V	153	2,98	2,98	1358	2,98	-0,11	-3,55987	3,08	2,95	88932,94	29797	0,19	1,17	19:00:00
11	AMEAS	120997	27	27	129107	27	-0,95	-3,39893	27,83	26,85	14137400	519070	0,47	7,11	19:00:00

B

9340	AMEAS	994	26,96	26,98	458	26,98	-0,97	-3,47048	27,83	26,85	9618538	351663	0,47	7,11	18:02:51	2015-12-18
9341	AMEAS	2163	26,96	26,98	458	26,98	-0,97	-3,47048	27,83	26,85	9618538	351663	0,47	7,11	18:02:54	2015-12-18
9342	AMEAS	2163	26,96	26,98	458	26,98	-0,97	-3,47048	27,83	26,85	9618538	351663	0,47	7,11	18:02:55	2015-12-18
9343	AMEAS	2163	26,96	26,98	670	26,98	-0,97	-3,47048	27,83	26,85	9618538	351663	0,47	7,11	18:02:58	2015-12-18
9344	AMEAS	49	26,97	26,98	670	26,98	-0,97	-3,47048	27,83	26,85	9618538	351663	0,47	7,11	18:03:02	2015-12-18
9345	AMEAS	606	26,97	26,98	583	26,98	-0,97	-3,47048	27,83	26,85	9618538	351663	0,47	7,11	18:03:13	2015-12-18
9346	AMEAS	104	26,95	26,97	634	26,96	-0,99	-3,54204	27,83	26,85	9695706	354525	0,47	7,11	18:03:16	2015-12-18
9347	AMEAS	473	26,95	26,98	612	26,96	-0,99	-3,54204	27,83	26,85	9695706	354525	0,47	7,11	18:03:17	2015-12-18
9348	AMEAS	100	26,97	26,98	458	26,97	-0,98	-3,50626	27,83	26,85	9703743	354823	0,47	7,11	18:03:26	2015-12-18
9349	AMEAS	132	26,97	26,98	458	26,97	-0,98	-3,50626	27,83	26,85	9703743	354823	0,47	7,11	18:03:42	2015-12-18
9350	AMEAS	542	26,97	26,98	458	26,97	-0,98	-3,50626	27,83	26,85	9703743	354823	0,47	7,11	18:03:44	2015-12-18

C

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
532858	YTY1V	5922	5,12	5,13	3243	5,13	-0,32	-5,87156	5,44	5,1	2430878	465863	0,44	4,21	18:02:51	2015-12-18
532859	FUM1V	5171	13,24	13,25	5767	13,25	-0,07	-0,52553	13,43	13,19	24572641	1847080	3,55	12,23	18:02:51	2015-12-18
532860	KNEBV	576	38,55	38,57	720	38,55	-0,73	-1,85845	39,1	38,33	20380458	527148	1,44	3,93	18:02:53	2015-12-18
532861	NDA1V	19781	9,855	9,865	7920	9,875	-0,145	-1,44711	10,02	9,785	17376888	1755141	0,83	7,37	18:02:53	2015-12-18
532862	NOKIA	10810	6,32	6,325	64450	6,32	-0,175	-2,69438	6,46	6,3	1,56E+08	24542356	0,92	2,3	18:02:52	2015-12-18
532863	NRE1V	114	32,83	32,84	741	32,84	-0,61	-1,82362	33,51	32,71	9886611	299158	1,56	9,05	18:02:52	2015-12-18
532864	ORNAV	400	31,1	31,16	108	31,17	-0,31	-0,98475	31,48	30,99	162466,6	5204	1,5	3,65	18:02:53	2015-12-18
532865	ORNBV	598	31,28	31,31	463	31,3	-0,3	-0,94937	31,66	31,11	3091836	98369	1,5	3,65	18:02:52	2015-12-18
532866	SAA1V	1448	3,812	3,822	1230	3,818	-0,036	-0,93409	3,832	3,76	566197,7	149055	0,36	5,54	18:02:52	2015-12-18
532867	STERV	7537	8,47	8,475	1152	8,475	-0,125	-1,45349	8,61	8,41	13536138	1592797	0,13	6,43	18:02:52	2015-12-18
532868	TIE1V	115	24,46	24,48	230	24,48	-0,24	-0,97087	24,78	24,34	2291937	93659	0,48	6,39	18:02:53	2015-12-18

Figure 4. Excel spreadsheet types A, B and C, used during the data collection

Three different spreadsheet types are as follows. A) A designated spreadsheet for Wintrade where it periodically copies latest tick data of monitored stocks. B) A stock specific spreadsheet for each monitored stock. These contain unique tick data in chronological order, therefore making it convenient to inspect by a human reader. C) A total spreadsheet containing all unique ticks from the monitored stocks in the chronological order. This spreadsheet will be utilized as a data feed to the backtester application after it is saved into the .csv format.

5.5 VBA Macros

The next step is to move onward into the tick data storage system development. Visual Basic for Applications (VBA) macros can conveniently process Excel spreadsheet data. In Excel a macro can be run explicitly or from built-in trigger functions. In this case a trigger function is used, being called every time a spreadsheet receives new data. Name of the trigger function is *Worksheet_Change*. That function then calls self-developed VBA macro code that does the actual data processing. High level data collection macro is described by the pseudo code below. A real implementation of the macro is attached in Appendix 1.

```
loop each stock
  if tick timestamp changed
    find last row in stock sheet
    copy tick data to last row + 1 in stock sheet
    find last row in total sheet
    copy tick data to last row + 1 in total sheet
  endif
endloop
```

Even if the main data functionality of data copying macro is straightforward, more tick data handling macros were developed during the project. Those serve different purposes and were written ad-hoc as the need for them raised. Such macros include for example the following: A date conversion macro that changes the date from dd-mm-yyyy presentation to a naturally sortable yyyy-dd-mm presentation. Tick delete macro deleting all ticks from the stock spreadsheets outside the given date range. Data composition macro combining tick data from the stock spreadsheets to the total spreadsheet. Totally around 500 lines of VBA macro code was written throughout the project.

5.6 Data Collection Results

Tick data collection was started in the beginning of November 2015. All stocks available for trading in Helsinki stock exchange were selected into the collected data set. That sums up as 134 individual stocks. No separation was made between low, medium and high capital stocks, all of them were included. Even in practice some of the low capital stocks are so seldom traded and in such low volumes that they are not suitable for the short-term algorithmic trading. Nevertheless, any data might proof itself worthy later on, thus all available stocks were included.

Some effort was required to correct initial programming errors in the macros. The errors were discovered during the early days of the data collection. Also one usability related issue forced partial rewriting of the macros. That was because the macros were initially using the Windows clipboard to copy data between the spreadsheets, rendering the clipboard useless for other applications during the data collection.

After the problems were solved, the data collection operated without problems. WinTrade provides a reliable connection to the Nordnet servers and smooth operations otherwise

as well, for example application crashes were not observed. Tick data provided by Nordnet does not contain noticeable numeric errors and seems to correlate with the information from other market information providers. However, when the market is not open zero values are included in some parts of the data. But this is no real issue as the data outside the market open times can easily be discarded.

On average the data generation speed observed during the first couple of months was 150 000 – 250 000 ticks per trading day. When taking into account the Excel sheet row limitation of just over one million lines, one sheet was filled typically in about one week. An Excel workbook can contain several spreadsheets with the maximum number of lines. But this was not considered a suitable approach as the Excel workbook size would increase dramatically, thus making the workbook too clumsy to operate with. For example, opening of an Excel workbook of hundreds of MB in size might take several minutes. With the current practice the workbook size stayed between 120 and 180 MB, which is yet manageable.

Relatively modern computer hardware is able to perform the data collection without performance issues. CPU time is spent into running WinTrade application and the VBA macros. The CPU utilization for Wintrade is typically around two percent and memory usage less than 100 MB. For the VBA macros the corresponding numbers are around five percent and 150 MB. These numbers mean that the computer can simultaneously be used for other tasks as well, while the data collection runs conveniently in the background. The system was running on Intel i5-750 processor with Windows 10 operating system.

6 Backtester Application

The practical work in this thesis is divided into two parts, the data collection and the backtester application design and implementation. This chapter describes in detail the different stages of the backtester application development, as well as the outcome. Creating a new application from the clean table is a process with multiple stages. Traditionally at least the setting of design goals or requirements, implementation and testing can be separated as individual phases. However, the process does not only work one way, but the later phases give feedback to the earlier phases improving the application iteratively. In addition new ideas not foreseen in the beginning typically arise during the development work.

6.1 Design Goals

The main function of the backtester application is to simulate algorithmic trading as it would take place in the real life. An outcome of this data processing answers the question how the algorithm currently under evaluation would behave if it had been running in live during the time period in question. The outcome can also be used to identify reasons behind the decisions made by the algorithm.

6.1.1 Functional Design Goals

On a high level the application shall read the tick data and feed it to an algorithm. The algorithm will do its calculations and form a decision whether to perform a buy or a sell transaction or neither of them. When the transaction is performed the application shall simulate details related to the transaction and store the transaction details. It shall calculate financial numbers during each transaction, such as profit and loss.

Tick data compatibility. Naturally the application shall be able to read and understand the collected tick data. However, the application shall be implemented so that feeding it with different style of tick data format is possible with minimal effort. That is, the application shall have a class taking care of the tick data interpretation and forming an internal presentation of the tick data. If another kind of data needs to be used later on, only the interpretation part requires changes.

Algorithm support. Developing a trading algorithm can be seen as an iterative work by nature. It can start with a rough idea and develop further through several draft versions. Also totally new ideas can arise forming new algorithms. As only one algorithm can be run at once, it shall be possible to easily switch between the algorithms.

Batch data execution. The tick data is stored in multiple Excel workbooks due to the Excel limitation and workbook handling convenience. The application shall be able to take several workbooks as an input at once. This enables processing of large data masses without user intervention.

Trading commissions. Typically each transaction with the broker company, being a buy or a sale transaction, involves a commission. Without the commissions, an algorithm making frequent transactions just to probe the situation, could be the best alternative. But with the commissions the situation changes substantially as the profit from trades must be able to cover the transaction costs, just to break even. Therefore the application shall simulate realistic transaction commissions.

Start balance. Especially due to minimum commissions the account balance can have a significant impact on the final outcome. Therefore the user shall be able to easily set desired start balance for his/her simulated trading account.

Look-ahead bias avoidance. The look-ahead bias is essentially a programming error where the backtester application utilizes a future data compared to the timestamp of the tick it is currently processing. Design of the application shall be such that the algorithms are not able to access the future data.

6.1.2 Visual and Usability Design Goals

Graphical User Interface. Application shall have a GUI in a typical manner for a Windows application. All information in the GUI shall be refreshed periodically during the execution.

Start and stop. The user shall be able start and stop the trade simulation with a push of a button. Stopping is a beneficial feature since executions can take long time. Sometimes it can be seen right from the start that the latest changes are not working properly, therefore an ability to stop the execution is required.

Progress information. Simulating trades over a long time period of historical data can take significant amount of time. Required time depends ultimately from the complexity of the algorithm i.e. how many calculation the algorithm needs to perform per each tick. This may vary greatly between different algorithms. The progress information shall show for example an input file name the application is currently processing and how much time is still required for the completion.

Financial summary. The summary section shall display result as key financial numbers. These numbers contain at least the trading account balance, maximum profit and maximum loss, number of trades generated and the Sharpe ratio. Purpose of the summary section is to give quick overlook how the algorithm performs without going into details.

Trade specific information. The application shall display closed trades in a list format. The columns shall represent relevant information about the trade, for example a stock name, a buy and a sell time, amount of shares, a buy and a cell price etc. This section aims to provide necessary information when inspecting what kind of individual trades the algorithm generated.

Visualisation of data. It is cumbersome to get complete picture of a price development involved in the trade just by looking at the numbers. Therefore the application shall support drawing of charts of the closed trades. The chart shall display selected data over suitable period of ticks. At least the current price information, buy and sell points should be plotted.

6.1.3 General Design Goals

Windows application. Since the tick data collection takes place in the Windows environment, the target environment for the application is naturally also Windows. The application shall be built on top of .NET architecture and written in C# language, as those are proven to be well suited tools for the modern Windows development.

Financial accuracy. In the tick data different type of price information is presented in the accuracy of one tenth of the cent. The application shall maintain the accuracy level of the tick data. Special attention shall be paid when rounding and multiplying the financial

numbers. In order to maintain the accuracy it is preferable to use the Decimal data type for variables presenting price information.

Live trading extensibility. The application is currently used for simulation purposes with the historical data. However, it shall be designed and implemented in such manner that extending its functionality towards a real trading platform is possible. In such case one needs to implement major new components such as a broker interaction and a transaction handling. Still the actual core logic, being the trading algorithm, would run exactly same way. Therefore, the trading algorithm and the internals of the application shall not be aware whether the data is historical or live data. This is a crucial requirement as the investor wants to be sure that the algorithm behaves exactly same way in live than in simulations.

GUI smoothness. Running trade simulation shall not freeze or block the GUI. This is important for the operability as the processing can take long time. Solution to this is to utilize multithreading. One thread shall handle the actual trade simulation processing while another thread is dedicated for the GUI. The GUI thread is responsible for drawing and updating the GUI, handling the GUI related events and user interaction with the application.

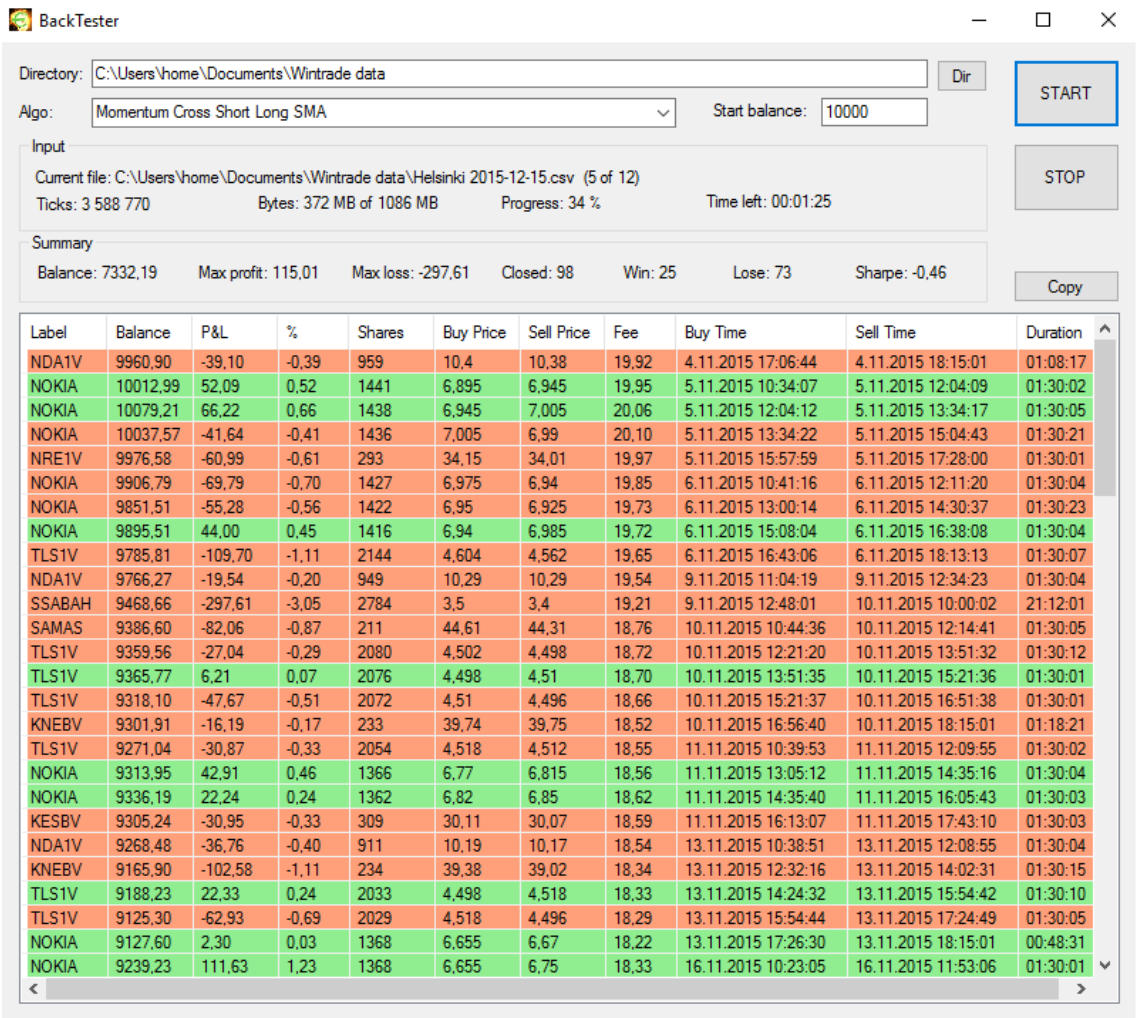
Separation of GUI and the inner logic. The application logic shall not be bound to the GUI. That is the GUI can be replaced with a completely different way of displaying the information and the actual data processing parts will be compatible as such. This can be achieved by following the model-view-controller architecture.

6.2 Implementation

In this section the design and the implementation of the backtester application are explained. The application is a traditional Windows application. Therefore the most suitable tools for the application development were selected. The factors influencing the decision were productivity and user friendliness of the development environment, among others. As a result the application is written with C# language and utilizes Microsoft .NET framework. The development environment was Visual Studio 2015. Object oriented programming was followed throughout the implementation. The following chapters give a detailed description of the application and its components.

6.2.1 Main Form

Inside Windows application the windows are called forms. Perhaps the most illustrative and reader friendly way to start describing the application is from its user interface. The main form opens when the application starts. Giving input parameters, starting and stopping the simulation run and inspecting the results, all take place through the main form. A screenshot of the main form while an algorithm is running is shown in Figure 5.



Label	Balance	P&L	%	Shares	Buy Price	Sell Price	Fee	Buy Time	Sell Time	Duration
NDA1V	9960,90	-39,10	-0,39	959	10,4	10,38	19,92	4.11.2015 17:06:44	4.11.2015 18:15:01	01:08:17
NOKIA	10012,99	52,09	0,52	1441	6,895	6,945	19,95	5.11.2015 10:34:07	5.11.2015 12:04:09	01:30:02
NOKIA	10079,21	66,22	0,66	1438	6,945	7,005	20,06	5.11.2015 12:04:12	5.11.2015 13:34:17	01:30:05
NOKIA	10037,57	-41,64	-0,41	1436	7,005	6,99	20,10	5.11.2015 13:34:22	5.11.2015 15:04:43	01:30:21
NRE1V	9976,58	-60,99	-0,61	293	34,15	34,01	19,97	5.11.2015 15:57:59	5.11.2015 17:28:00	01:30:01
NOKIA	9906,79	-69,79	-0,70	1427	6,975	6,94	19,85	6.11.2015 10:41:16	6.11.2015 12:11:20	01:30:04
NOKIA	9851,51	-55,28	-0,56	1422	6,95	6,925	19,73	6.11.2015 13:00:14	6.11.2015 14:30:37	01:30:23
NOKIA	9895,51	44,00	0,45	1416	6,94	6,985	19,72	6.11.2015 15:08:04	6.11.2015 16:38:08	01:30:04
TLS1V	9785,81	-109,70	-1,11	2144	4,604	4,562	19,65	6.11.2015 16:43:06	6.11.2015 18:13:13	01:30:07
NDA1V	9766,27	-19,54	-0,20	949	10,29	10,29	19,54	9.11.2015 11:04:19	9.11.2015 12:34:23	01:30:04
SSABAH	9468,66	-297,61	-3,05	2784	3,5	3,4	19,21	9.11.2015 12:48:01	10.11.2015 10:00:02	21:12:01
SAMAS	9386,60	-82,06	-0,87	211	44,61	44,31	18,76	10.11.2015 10:44:36	10.11.2015 12:14:41	01:30:05
TLS1V	9359,56	-27,04	-0,29	2080	4,502	4,498	18,72	10.11.2015 12:21:20	10.11.2015 13:51:32	01:30:12
TLS1V	9365,77	6,21	0,07	2076	4,498	4,51	18,70	10.11.2015 13:51:35	10.11.2015 15:21:36	01:30:01
TLS1V	9318,10	-47,67	-0,51	2072	4,51	4,496	18,66	10.11.2015 15:21:37	10.11.2015 16:51:38	01:30:01
KNEBV	9301,91	-16,19	-0,17	233	39,74	39,75	18,52	10.11.2015 16:56:40	10.11.2015 18:15:01	01:18:21
TLS1V	9271,04	-30,87	-0,33	2054	4,518	4,512	18,55	11.11.2015 10:39:53	11.11.2015 12:09:55	01:30:02
NOKIA	9313,95	42,91	0,46	1366	6,77	6,815	18,56	11.11.2015 13:05:12	11.11.2015 14:35:16	01:30:04
NOKIA	9336,19	22,24	0,24	1362	6,82	6,85	18,62	11.11.2015 14:35:40	11.11.2015 16:05:43	01:30:03
KESBV	9305,24	-30,95	-0,33	309	30,11	30,07	18,59	11.11.2015 16:13:07	11.11.2015 17:43:10	01:30:03
NDA1V	9268,48	-36,76	-0,40	911	10,19	10,17	18,54	13.11.2015 10:38:51	13.11.2015 12:08:55	01:30:04
KNEBV	9165,90	-102,58	-1,11	234	39,38	39,02	18,34	13.11.2015 12:32:16	13.11.2015 14:02:31	01:30:15
TLS1V	9188,23	22,33	0,24	2033	4,498	4,518	18,33	13.11.2015 14:24:32	13.11.2015 15:54:42	01:30:10
TLS1V	9125,30	-62,93	-0,69	2029	4,518	4,496	18,29	13.11.2015 15:54:44	13.11.2015 17:24:49	01:30:05
NOKIA	9127,60	2,30	0,03	1368	6,655	6,67	18,22	13.11.2015 17:26:30	13.11.2015 18:15:01	00:48:31
NOKIA	9239,23	111,63	1,23	1368	6,655	6,75	18,33	16.11.2015 10:23:05	16.11.2015 11:53:06	01:30:01

Figure 5. The backtester application main form while running a simulation

The main form contains the necessary controls for user input. Firstly, there is a directory path where the application parses all .csv files containing the tick data. The directory path can be written, pasted or selected through the “Dir” button. Secondly, the user can select an algorithm for the execution from the dropdown menu. Thirdly, the user must input a start balance of the simulated trading account. The application remembers the

previously given input values making it more convenient to perform new simulation rounds later on.

The start button naturally starts the simulation run and the stop button stops it. The user can also start the simulation by pressing F5 key. The stop button is practical when testing algorithm changes that immediately show negative results, in that case it might not be interesting to wait until the simulation ends. With the copy button user can copy the content of the closed trades list to the clipboard. This way the information can be analyzed further for example in a spreadsheet application.

The “Input” group of statistics aims to give status of the current simulation run. It provides information on which input file is currently processed, how many input files there are altogether, how much there are ticks and data, what is the current progress as percentage and how long time the execution is still going to last.

The “Summary” group presents the most interesting financial data of the simulation run that can be presented as a single numerical value. Pages of financial information could be presented from an algorithm simulation. However, here is selected the information that is seen necessary at this point. Potentially later on the financial data may need to be expanded. The summary contains the following information: the current account balance, the maximum profit and loss, the number of the closed trades with separation into winning and losing trades and the Sharpe ratio.

Finally the main form contains a list of all closed trades. A closed trade contains a buy and a sell transaction. The trades are depicted by colors, the green color implies a winning trade whereas the red color stands for a losing trade. The objective here is to provide an instant view of the distribution and the quantity of the winning and losing trades. Detailed information in each row includes the following. The name of the stock, the account balance after the trade, the profit or a loss in currency and percentage value, the number of traded shares, the buy and sell prices of the share, the combined buy and sell commission, the buy and a sell timestamp and the holding period.

6.2.2 Chart Form

The purpose of the chart form is to provide detailed visual information about a closed trade. The visual information is necessary when estimating is the algorithm generating

sensible buy and sell transactions and could those decisions still be improved. The chart form is opened by double-clicking a closed trade row on the main form. All daily ticks of the selected stock are drawn on the chart. A screenshot of the chart form is presented in Figure 6.

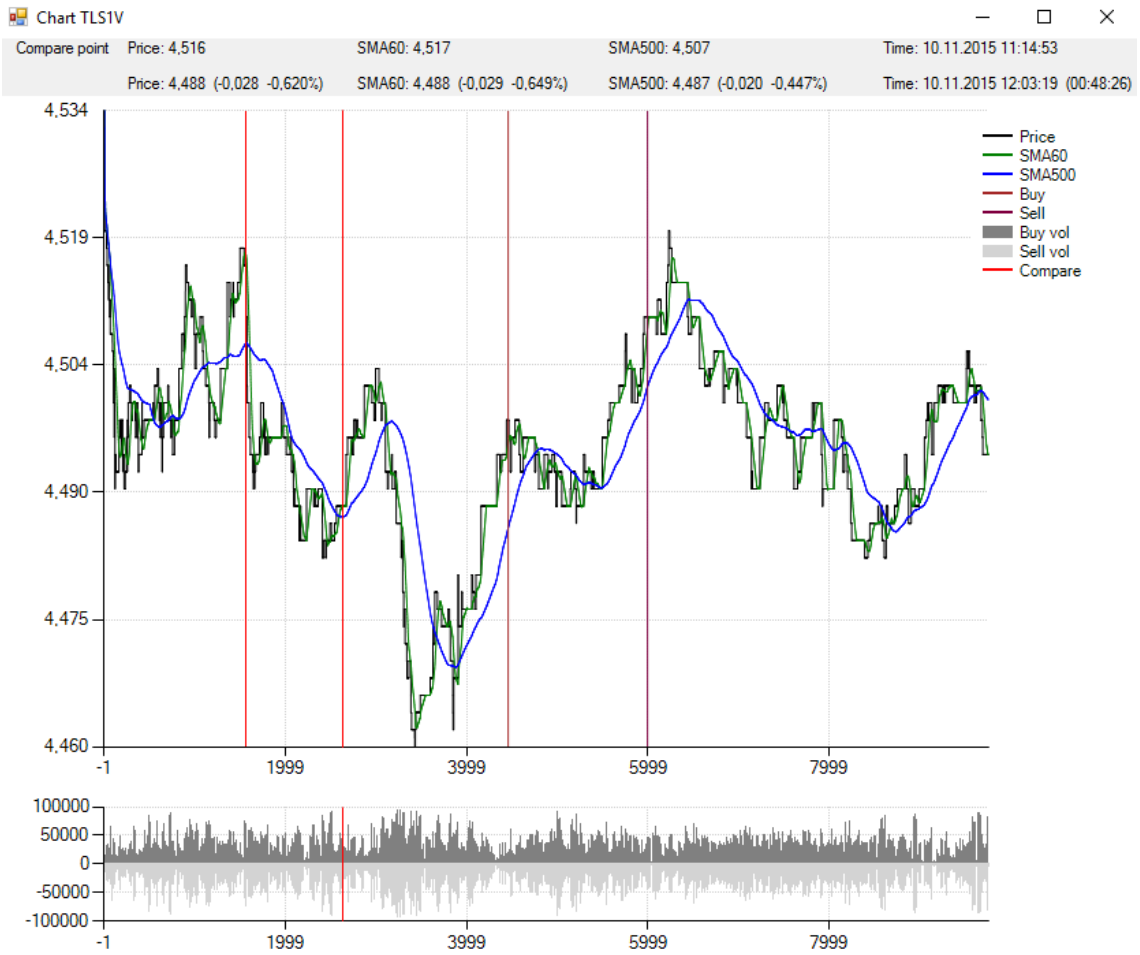


Figure 6. The backtester application chart form with the compare mode activated

In Figure 6 the chart form consists of two chart areas and a group of labels on the top of them. The price chart area displays the price information per tick. The X-axis presents the daily tick numbers and the Y-axis the price information as currency. Currently three price related curves are drawn in the chart. Those are the current price and the short and long term simple moving averages calculated from it. In this example, the moving averages are 60 and 500 samples long. In Figure 6 information of TeliaSonera stock (TLS1V) is displayed. During the day the trading of TLS1V generated about 10 000 ticks.

The volume chart area is displayed under the price area. This chart area draws the volumes of buy and the sell events in the market in relation to the tick number. The X-axis

presents the daily tick numbers and the Y-axis the volumes, buy volumes as positive numbers and sell volumes as negative numbers. The purpose of the volume chart is to bring into attention if the market contains any correlation between the volume and the price movements. Such a correlation could naturally be used as an advantage in the algorithm. The example shown in Figure 6 does not have any clear correlation.

The buy and sell moments are perhaps the most interesting points in the chart. They are illustrated in the colored vertical lines. In the example the brown line stands for a buy transaction and the purple line for a sell transaction. By looking at the price curve one can almost instantly notice where the buy and sell lines should be in order to maximize the profit in the particular situation.

In Figure 6 the right hand side red vertical line is a cursor line. The cursor is always redrawn when the user moves the mouse over the chart. At the same time the price, moving averages and time labels are updated. With the cursor line the user can quickly see numerical and hence accurate information throughout the chart.

The functionality of the chart also contains a compare mode. The compare mode extends the usability of the cursor line by adding another line. In Figure 6 it is the left hand side red vertical line. This line is static. When the compare mode is activated and the user moves the cursor, the lower labels show the difference in the price, the moving averages and the time between the lines. Therefore enabling easy answering to questions such as how much there is price increase between two chosen points in the chart. By default the compare mode is off. It is activated by left-clicking on the chart. At the same time the static data point for the comparison is set. The compare mode is deactivated by right-clicking on the chart.

The application supports opening several charts simultaneously. This can be done for example for comparison purposes. Chart functionality can also be extended and new information added quite conveniently.

6.2.3 Threads

The application is multithreaded, consisting of a GUI thread and a background worker thread. The GUI thread takes care of the user interface handling, including waiting for the user inputs and updating information for the user periodically. Separating the GUI

handling into an own thread the application remains responsive during intensive and potentially long lasting data processing. The background thread is set up when starting the simulation and disposed when completing or stopping the simulation. The background thread takes care of the tick data processing. This includes reading a tick, directing it to a selected algorithm, running the algorithm, updating the results and then continuing from a next tick.

Since the data processing and GUI handling are in separate threads, data sharing between the threads is required. The data sharing could have been implemented with the event handling mechanism. In that case the background thread would send an event when something interesting occurs and the GUI thread would receive it. However, the events occur in such high frequencies that it would be unnecessary to update the GUI at this speed. This would also waste CPU resources. Therefore, the data is shared through a static class, illustrated in Figure 7.

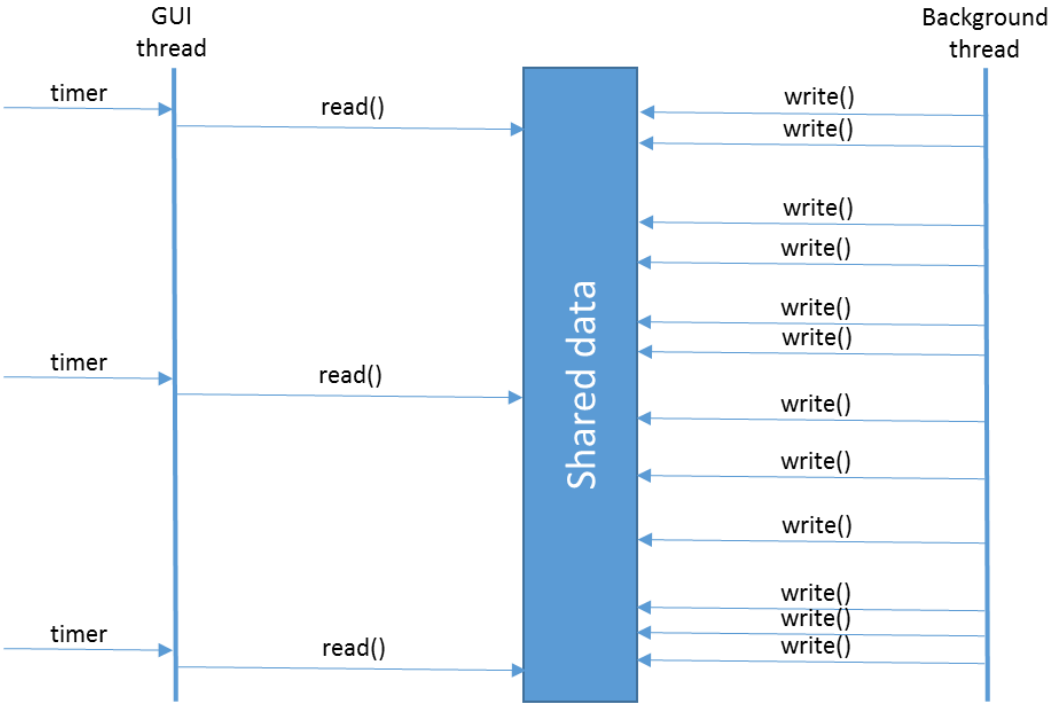


Figure 7. Application data sharing between the threads

When the simulation is started the GUI thread creates the background thread and sets a periodical timer for data reading. The timer expiration interval is 200 ms. With that frequency the user interface can be updated fast enough for the human eye. The background thread writes data to the shared object each time a certain data is updated, for example the number of ticks processed. Figure 7 presents an abstraction of the read and

write interfaces. In the real implementation each type of shared data has its own read and write interfaces, such as writeBalance() and readBalance(). The shared data is protected by a lock mechanism ensuring that the read operation gets always complete data, instead of partially written. By providing multiple read and write interfaces the likelihood for a collision is reduced.

6.2.4 Classes

In an application written in an object oriented programming language all functionality is gathered into classes. Therefore, describing the functionality of the classes also presents the functionality and structure of the application. This chapter gives an overview of the most important classes in the application. These classes are designed and written by the author, the classes generated automatically by Visual Studio are not covered. All together the author written classes contain approximately 2000 lines of C# code. A complete class diagram generated by Visual Studio tools is attached into Appendix 2.

TickReader. This class is responsible for producing a tick stream for rest of the application. It takes care of the details when reading .csv files containing the tick data. The files are read in alphabetically ascending order, line by line, with the help of the system class *StreamReader*. The class also utilizes Language-Integrated Query (LINQ) for the file sorting. If the data source of the ticks would change, for example to a network interface, changes can be hidden solely behind this class. As an output, the class returns an instance of a Tick.

Tick. A class presenting a single tick. It contains most of the data fields from the collected tick data. The fields are the stock name, bid and ask volumes, bid and ask prices, the last price, the price difference compared to yesterday, the day's highest and lowest prices, the traded volume and capital and the timestamp. The class constructor parses a line of .csv data into a tick presentation. The parsing can be adjusted into any tick data format when needed.

Dispatcher. Rationale of this class is to guide the tick to the correct algorithm instance. Algorithm instances are stock specific. Therefore the class maintains a dictionary mapping a stock name and the algorithm instance together. In addition the class takes care that the user selected algorithm type is instantiated.

TickQueue. The class encapsulates FIFO type queue functionality, tailored for the ticks. An instance of the tick queue has a specified length, for example 3000 ticks. The tick queue is implemented as a ring buffer, overwriting obsolete records. It provides methods to insert a new tick to the queue and to fetch a certain passed tick from the queue.

SMA. A class for the simple moving average calculation. The length of the averaging window is set during the class instantiation. The class provides a method for inserting a new decimal sample, typically a price, to the window. It naturally provides a method for inquiring the averaged value. The class stores samples into a ring buffer. The implementation contains the basic SMA calculation and the optimization, therefore covering both equations 3.1 and 3.2 presented earlier.

BaseAlgo. This class acts as a base class to algorithm classes. The principle is to keep common functionalities for all algorithm types in one place. The functionalities include the following. Updating the tick queue and the moving average windows. Performing tick data validity check. Inspecting preconditions for buy and sell transactions, such as market time related checks, current holdings related checks, market liquidity checks etc. Ultimately, after filling all the preconditions, a buy or sell decision is formed by running the technical analysis. The class triggers the analysis by calling virtual methods analysisBuy and analysisSell, which are overridden by the actual algorithm.

AlgoSMACross. This class is an example algorithm. It inherits BaseAlgo class and implements the analysisBuy and analysisSell methods. This particular algorithm generates buy and sell signals based on a short and a long term moving average. When the moving averages cross each other, that point is considered a potential moment for the buy or sell transaction, depending of the direction of the movement.

AlgoUnfilteredPrice. This is another example algorithm. The buy and sell signal generation is based on a sudden rise and fall in the current price information. The price information doesn't use any filtering techniques such as the moving average.

AlgoBuySSMASellLSMA. This is yet another example algorithm. The idea behind the algorithm is that the buy event should be triggered rather quickly in order to benefit from the ongoing price movement. The sell event, on the other hand, should not be signaled too hastily as the price fluctuations could trigger it too early. Therefore, the algorithm

uses short-term moving averages for the buy decision and the long term moving averages for the sell decision. This algorithm, as well as other example algorithms, can be categorized as alpha algorithms relying on the price momentum.

Holding. This static class takes care of the holdings i.e. shares currently owned. It also maintains the trading account balance. The class offers sell and buy methods. These methods take a tick as a parameter and perform potentially a buy or a sell transaction. The buy method determines amount of shares, subtracts a transaction fee, marks the shares as owned and decrements the balance. The sell method sells the owned shares with a current price, subtracts a transaction fee, increases the balance and calculates the profit and loss. The class performs also the financial portfolio management, even that part is strongly simplified. For example, an amount of shares acquired is decided solely by minimization of the transaction costs.

Utils. A static class collecting together market and financials related functionalities. The market related functions include calculation of the commissions and defining available trading times in the market. The commissions are calculated according to current commission rules of Nordnet. The class provides also calculation of the Sharpe ratio.

Statistics. This is the data sharing class between the GUI thread and the background thread. The class is static, therefore it can be accessed anytime and anywhere within the application. Multiple classes write statistical data to the class, including TickReader, Dispatcher, BaseAlgo and Holding. The data covers anything from the simulation progression to a trade specific details. Most of the data visible in the GUI is delivered via this class.

MainForm. The class inherits Form class. It contains functionality required by the main window of the application and running the background thread. The functionality includes following. The class provides event receiving points for the control generated events such as button clicks, drop down menu selections and list view clicks. It takes care of setting and resetting the periodical GUI update timer, as well as offering receive point for the timer event. The timer event updates all the GUI elements, such as the labels and the list view, based on the statistics data and partly on local calculations. The class holds main operating loop for the background thread. The loop gets ticks from TickReader and feeds them further into the application. The loop continues until all ticks are processed

or the user has stopped the execution by clicking the stop button. The class also performs input data validations and displays appropriate warning messages when necessary.

ChartForm. The class inherits Form class. It contains procedures for drawing the charts. The class is instantiated when the user double-clicks on a closed trade. After creation the class fetches required data from the statistics class and creates the charts using classes from MSChart collection. The class also implements handlers for mouse move and mouse click events. The mouse move event is used to draw the cursor line. The mouse click event, either a left- or a right-click, determines activation and deactivation of the compare mode. When the compare mode is active, mouse move event causes recalculation of data difference between the current point and the compare point. This information is updated to the labels. The class implements also a key event allowing user to close the chart by pressing the Esc key.

6.3 Testing

Software testing is an important step of the software development. Traditionally it takes place during the last stages of the project. However, in modern software development testing is closely bound to the actual implementation work. This is especially true when the same person takes care of the both tasks. This project was no exception. Implementation and testing cycles followed each other typically one small feature at a time. The testing of the backtester can be divided into two parts, testing the application functionality and financial performance of the algorithms.

6.3.1 Application Functionality

Visual Studio provides a C# unit test framework. The framework was utilized to verify the application core classes and the most fragile functionality. A bug in those classes could potentially produce a systematic error affecting the whole application. In addition such an error could potentially be hard to detect. Especially financial calculations require verification with the unit tests. Unit tests offer also one major benefit, a safety net for core functionality enabling safer modifications later on.

Unit tests were written for the following classes and functionalities. *Tick* class is verified by creating ticks from various tick data, presented as a line of the .csv file data. All the tick member fields are carefully inspected. *TickQueue* class is tested by creating tick queues of various lengths, inserting various amount of ticks and inquiring them. The ring buffer functionality is verified too. *Holding* class tests verify the buy and sell transaction effects to financial data such as account balance, share count, fees and profit and loss. *SMA* class testing verify that the averaging produces correct values right from the first samples inserted into the window. It also ensures that the quick calculation mode produces correct values and the ring buffer operates as expected. *Utils* class testing covers calculation of the Sharpe ratio, among others.

The higher level functionality of the application, including the GUI, was tested by using the application and running the example algorithms with the tick data. Typically GUI testing took place right after implementing a new feature into it. After the implementation it was convenient to start up the application and try out the new feature, making possible adjustments when necessary. Testing the tick data processing included manual cross checks between the information displayed by the application and the actual tick data. Deterministic operation of the simulation was also verified. That is, a simulation run with the same inputs parameters produces same results every time.

6.3.2 Example Algorithms

Verifying the buy and sell events generated by the algorithms was the final stage of the testing. Ultimately, this can also be seen as the purpose of the application. Here the chart windows offer valuable aid when inspecting individual transactions more closely. In the chart all the price information working as an input to an algorithm is visually presented. An inspector typically focuses on one transaction at a time, whether it being a buy or a sell transaction. By looking at the price developments before the transaction and comparing them to the assumed algorithm logic, one can come to a conclusion whether the algorithm is doing what is expected. Moreover, this process also gives improvement ideas to the algorithms. Three example algorithms were verified by running them with tick data spanning from 4. November 2015 to 26. February 2016, consisting of all together 78 trading days. The results from the simulation runs are presented in Figure 8.

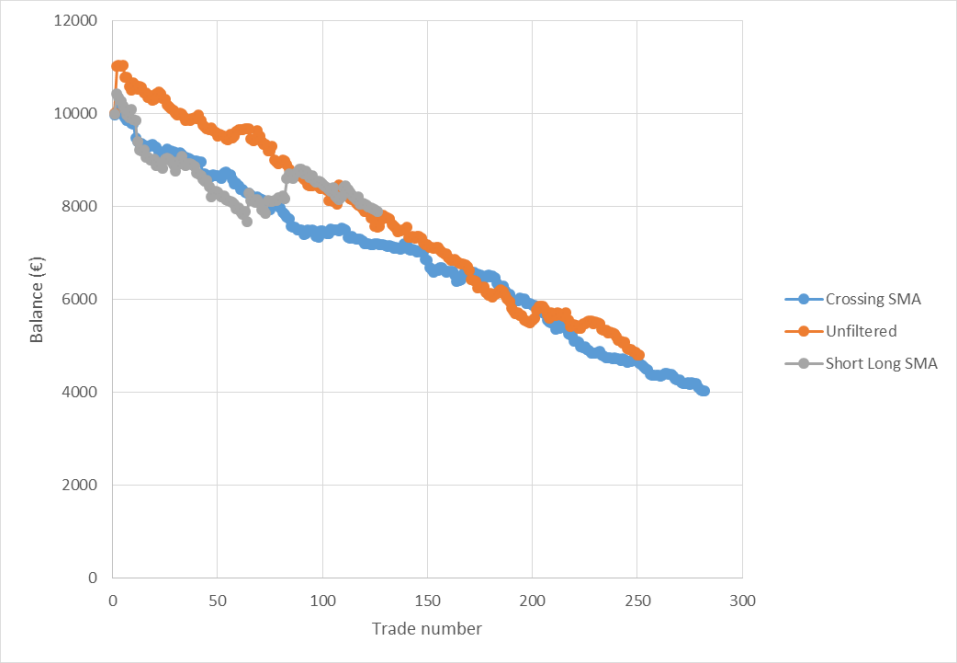


Figure 8. Financial results of the example algorithms

In Figure 8 the trading account balance is plotted in relation to the closed trades. The account balance is calculated after the shares have been sold and includes the effect of commissions. The initial account balance was 10 000 €. Three different algorithms were executed, each trying to forecast trading opportunities based on the price momentum. The “Crossing SMA” algorithm makes decisions when short and long SMA values are crossing each other. The “Unfiltered” algorithm relies on raw price information without any filtering such as the moving averages. The “Short Long SMA” algorithms makes buying decisions based on 60 samples long SMA and selling decisions based on 500 samples long SMA.

Different algorithms generated a varying number of trades during the same observation period, which is expected. But more importantly, all of the algorithms clearly produced losses. Occasionally they managed to generate a larger profit, visible as a rising bump in Figure 8. But a greater number of the non-profitable trades eventually destroyed the financial performance. The best performing algorithm was the “Short Long SMA”. Nevertheless, it still managed to vanish about 21 % of the initial account balance. Needless to say, all of these candidates would be unsuitable for the live trading in their current state.

6.4 Performance

Performance is one of the metrics used in the software evaluation. The backtester application has no strict performance requirements due to its simulative nature. However, the fluid performance still brings benefits. One obvious benefit for the backtester is a quicker feedback cycle. After making a change to an algorithm the developer typically wants to verify the effect immediately. Here a faster execution clearly improves the usability of the backtester application. Another need for performance would be due to mechanical optimization loops, iterating over all possible parameter combinations. Here a duration of a single execution round matters, as potentially hundreds or even thousands rounds are executed. However, as of writing this the application not yet has support for the optimization.

Application performance was measured by running four simulation rounds. In each round the application was fed with data collection consisting of known number of ticks. Then the execution time of the simulation was measured. The largest tick collection consisted of approximately 10^7 individual ticks, equaling almost four months of real time market data. The same algorithm was used throughout simulation rounds. The results of the performance measurement are presented in Figure 9.

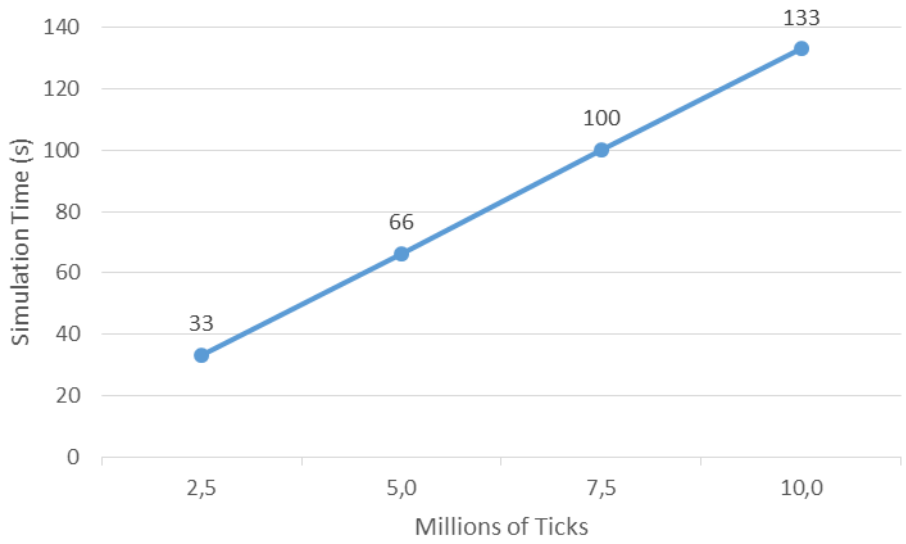


Figure 9. Performance of the backtester application

Figure 9 clearly shows that the application performance is linear. The simulation duration depends from the number of ticks processed in a linear fashion. The result is as expected as the application logic is designed to process one tick at a time, paying no attention to

the already processed ticks. A non-linear performance could have been indication of some design problems in the application. The simulations were executed on an Intel i5-750 processor running the Windows 10 operating system.

Naturally the application performance depends on all code currently included into the simulation. The more complex algorithms are developed, involving potentially heavy number crushing, the greater the effect will be on the performance. An algorithm effect was tested by excluding the technical analysis part of an algorithm by making analysisBuy method to return immediately. Interestingly, this had virtually no effect on the total processing time. The behavior can be explained by the fact that the example algorithms are relatively simple containing just little additional processing, as the rest of the application provides them with pre-processed data, such as the tick queue and the moving averages.

Another aspect for the backtester performance is the usability of a computer during the simulation. As the financial simulation runs on one thread, it stresses out one core performance to 100 %. In the case of a quad core processor this means 25 % total CPU usage. Operations in the GUI thread, performed five times per second, do not cause much load to a core running the GUI thread. However, if the charts are opened and closed frequently, then the GUI thread will be more heavily stressed as well. All in all, the backtester application runs fluently in relatively modern processors, having two or more cores. It leaves processing headroom for the user to perform other tasks during the simulation.

Application memory consumption was not an issue either. It hovers somewhat during the simulation, typically between 90 MB and 160 MB. The greatest single factor affecting the memory consumption is the number of stocks involved in the simulation. This is due to fact that each stock is handled by own algorithm instance, allocating separate buffers for ticks, statistics, moving averages etc.

7 Discussion and Conclusions

In this chapter conclusions from the results of the backtesting environment development are drawn and the experiences and ideas encountered during the work are discussed. The first section presents general observations for implementing the backtesting, the second section lists improvement ideas for the backtester application arisen during the project and the third section discusses algorithm design and behavior. In the fourth section the project is viewed in retrospective from the personal development point of view.

7.1 General Observations

The backtester application demonstrated the importance of backtesting well. Without backtesting developing a trading algorithm that would not demolish the account balance in a few months becomes very challenging. The trading algorithms used in the application verification and demonstration were surprisingly sensitive even to modest changes. For example, tuning certain threshold values by some tenths of percent could result in completely different financial performance. This on the other hand, speaks for the importance of the algorithm optimization loop support.

A backtester application aims to simulate the progression of real trading events. Therefore, the data input for the application should mimic the real live trading. An algorithmic trading system running in live processes a tick immediately after it is accessible, this is also how the simulator application should behave. This behavior can be easily violated for example by storing the tick data into a table before processing it. Processing ticks immediately offers two clear benefits. Firstly, the data snooping bias is efficiently eliminated as the future data is simply not accessible to the algorithm. Secondly, the algorithm can be applied to the live trading more securely, when differences between a simulation and a live code and are kept minimum.

The design of the classes in the backtester application should be done carefully and individual tasks should be clearly assigned to each of the classes. Especially the tick data encapsulation and flow inside the application should be carefully designed. Separation of the application logic and user interface is also a well proven technique. Special attention should also be paid to that financial numbers are handled and calculated correctly.

The application should be designed from the beginning with an expansion in mind. This enables developing the application further towards a complete trading platform functionality. In that case components requiring development are for example an ability to handle real time data feed and broker interaction.

The backtester application is aimed for algorithm development and therefore it should bring added value to the process. While inspecting the simulation results a detail view of events during an individual trade is required. This can be offered by a visual presentation of the data. The backtester should provide visualization of the data that is used in algorithm's decision making, whether it is price developments or something else. The accuracy of the data presentation should be at least the same as available to the algorithm, in this case one tick.

The results of a backtester simulation run should reflect real world performance of the algorithm. However, transaction execution is an area which is difficult to simulate. In addition it has a major impact on the financial results in short-term algorithmic trading with fast price developments. The backtester aims for rapid transaction execution by calculating that the market has enough liquidity before signaling the buy and sell transactions. Nevertheless, as the market liquidity varies constantly the transaction execution times in live market could drift from the simulation.

7.2 Backtester Application Improvements

Visibility of missed trading possibilities. The real time tick data contains vast amount of information, including several price movements that could potentially be seen as an advantage. However, currently the backtester only displays the charts related to closed trades, missing all potential close calls. Therefore, the application could have an option to include data matching certain conditions to the closed trades list, available for a graphical inspection. One of such condition could be an exceptional strong price development in percentage, available at the end of the trading day. The conditions should be easily configurable by the user.

Support for an algorithm optimization. When developing the example algorithms it was noticed that the algorithms can be surprisingly sensitive to parameter changes. Even the simplest algorithms typically contain a few adjustable parameters. For example, in the tested algorithms such parameters were the window lengths for the moving averages

and the threshold values for buy and sell decisions. The sensitivity of the parameters changes suggests that parameter values can have a dramatic influence on the financial performance. Therefore, the different parameter combinations should be verified. The problem is that even with a few parameters, each containing a set of try-out values, the number of combinations rise quickly. Testing all combinations manually becomes practically impossible. An optimization functionality solves the issue by running the simulation with each possible parameter set and picking up the best parameter set by comparing the results.

7.3 Algorithm Considerations

The project focus was on the backtesting environment development, where the algorithms can be clearly seen in the side role. Still a reasonable amount of effort and thoughts were put into the algorithms as well. Many algorithm ideas may first seem unbeatable when sketched out on a paper. However, many times the effect of non-profitable trades is neglected and they quickly bring down financial results to an unacceptable levels.

Before considering the live trading with an algorithm it needs to be profitable, preferably by a large margin. In order to minimize risk a profit distribution needs to be considered as well. An algorithm generating small profits constantly can be seen a better choice than an algorithm generating large profits seldom, since the first type of algorithm involves less risk and is more predictable. The Sharpe ratio can be effectively used to evaluate the risk profile.

During simulation runs a trading strategy generating a few carefully calculated buy events produced better results compared to a strategy generating more frequent buy events. The behavior can be explained by the fact that a hastily placed buy order needs to be more likely reverted soon due to an unfavorable price development. This generates losses at least in the form of the transaction commissions, perhaps also due to downward price movement. Not even a few strongly profitable trades were enough to revert the effect of too frequent buy events.

Data smoothing is an important part of the algorithmic trading. The collected tick data contains a few sudden and drastic price fluctuations. The fluctuations were several per-

cent in altitude and occasionally disturbed the tested algorithm efficiently. Such high fluctuations are not very common, but the collected data shows that they do exist. Therefore, a raw price data should be rarely used for decision making. A common way to smooth out the fluctuations is moving averages of different lengths. Generally, the longer the averaging window, the smoother the data. However, longer averaging windows have a downside, they react to trend changes with a longer delay. This can be somewhat improved by utilizing weighted moving averages.

7.4 Personal Development

The author has a background in the telecom industry software development, working mainly with embedded software. This project offered interesting viewpoints into a completely different field of computing. There were many new things to learn. First of all, financial computing, its terms and techniques were rather unknown area to the author in the beginning of the project. As the work progressed, they became increasingly more familiar. A door to a new world of knowledge was cracked open, as the financial computing field contains vast amount of information and topics.

The algorithmic trading is one of the topics. The field of an algorithmic trading contains lots of information. It is a field of information technology that develops constantly as new techniques and ideas become possible to implement due to ever more capable computer hardware, software and network connections. The algorithmic trading and especially backtesting was carefully studied during the project. Nevertheless, one can surely elaborate that this thesis only scratches the surface of the algorithmic trading.

The author has utilized Excel spreadsheets for various purposes before. However, writing VBA macros for the workbook data manipulation was a completely new topic. Learning VBA from the scratch took place by studying some of the principles and following several examples illustrating how things can be accomplished. VBA is definitely a useful addition to any programmer's toolbox. For example, Excel is commonly used for various data analyzation tasks. Perhaps with the power of VBA those task can be executed more efficiently.

Windows development with modern software tools became also very familiar during the project. The Microsoft development environment Visual Studio 2015 was used in the backtester application development. Therefore some of the features offered by .NET

framework become familiar through the project. The author had no earlier experience of the C#, which was used as the programming language. However, as being familiar with other C family languages and Java, the C# felt immediately familiar. The personal impression is that the C# enables rapid application development compared to more traditional languages. This is achieved by offering capable system classes for various tasks and a powerful development environment.

8 Summary

The objective of the project was to research how to develop a backtesting environment for the algorithmic trading in the stock markets. The end user for the backtesting environment is a retail trader. The project started from an empty table where no historical stock market data was available. By following the methods presented in the thesis traders may develop their own backtesting environment for the algorithmic trading. The project was divided into two parts, the data collection part to gather historical stock market data and the backtester application development part.

Nordnet was chosen as the stock market data provider for the data collection. The WinTrade trading client application was utilized in the data collection due to its ability to export the real time stock data into an Excel spreadsheet. In the beginning the data storage format was designed. As a result, the data was harvested into various Excel spreadsheets with the help of VBA macros. The data was collected from Nasdaq OMX Helsinki exchange, comprising of 134 different stocks.

The data collection results were successful. The collection met all of the preset design targets. During the project the real time market data was collected from over five month time period, forming a sufficient amount of information for the algorithm development. Naturally, the data collection process still continues.

The backtester application development started by laying out the design goals. The purpose of the backtester application is to process the collected historical market data, execute a trading algorithm and measure the financial performance of the algorithm. The backtester application offers valuable information while developing a trading algorithm. Without a backtesting functionality, developing a profitable trading algorithm is in practice impossible.

The backtester development succeeded fluently with the modern tools for Windows development. The application provides an easy to operate platform for the trading algorithm development. Adding new algorithms is a straightforward operation by adding a new class to the application. Common functionalities needed by an algorithm can be inherited from the algorithm base class. Moreover, the application may be further developed into a direction of the trading platform.

As the final conclusion the project proofed that it is feasible to develop a trading algorithm backtesting environment for a retail trader. Less effort was placed on the algorithms itself as they were clearly in the side role while the focus was on the backtesting environment. However, the impression is that developing a profitable trading algorithm for the stock markets with a decent risk profile is a challenging task.

References

Aldridge, I. (2010). *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*. New York: John Wiley & Sons.

Chan, E. (2013). *Algorithmic Trading - Winning Strategies and their Rationale*. New York: John Wiley & Sons.

Chiulli, R. (1999). *Quantitative Analysis: An Introduction.* Boca Raton: CRC Press.

Dacorogna, M., Gençay R. and Müller U. (2001). *An Introduction to High-Frequency Finance*. San Diego: Academic Press.

Davey, K. (2014). *Building Winning Algorithmic Trading Systems*. New Jersey: John Wiley & Sons.

Folger, J. (2016). *Investopedia - Advanced Guide to NinjaTrader*. Available at: http://www.investopedia.com/university/advanced-features-of-ninjatrader/ (Accessed Jan 5, 2016).

Folger, J. (2016). *Investopedia - Beginner's Guide to MetaTrader 4: Advanced Features*. Available at: http://www.investopedia.com/university/meta-trader-guide-intro4.asp (Accessed Jan 5, 2016).

Glen, J. (1994). *An Introduction to the Microstructure of Emerging Markets.* International Finance Corporation.

Gregoriu, G. (2008). *Encyclopedia of Alternative Investments*. Boca Raton: CRC Press.

Humphris, C. (2014). *Learn Basic Programming for Mathematics V10*. Lulu Press Inc.

Kissell, R. (2014). *The Science of Algorithmic Trading and Portfolio Management*. Oxford: Academic Press.

Leshik, E. and Cralle, J. (2011). *An Introduction to Algorithmic Trading - Basic to Advanced Strategies.* New York: John Wiley & Sons.

Nordnet. (2016). *Kaupankäyntisovellukset*. Available at: https://www.nordnet.fi/palvelut-ja-tuotteet/sijoittamisentukena/kaupankayntisovellukset.html (Accessed Jan 11, 2016).

Nuti, G., Mirghaemi, M., Treleaven, P. and Yingsaeree C. (2011). Algorithmic Trading. In: *IEEE Computer*. Volume 44 (1), 61-69.

Pardo, R. (2008). *The Evaluation and Optimization of Trading Strategies*. 2nd edition. New Jersey: John Wiley & Sons.

Ponomareva, N. and Calinescu, A. (2012). Extending and Evaluating Agent-based Models of Algorithmic Trading Strategies. In: *Engineering of Complex Computer Systems (ICECCS)*. 2012 17th International Conference, 351-360.

Reinkensmeyer, B. (2016). *TradeStation Review 2015*. Available at: http://www.stockbrokers.com/review/tradestation (Accessed Jan 6, 2016).

Sellberg, L. (2010). *Algorithmic Trading and its Implications for Marketplaces.* Stockholm: Cinnober.

Thulasidas, M. (2010). *Principles of Quantitative Development*. New York: John Wiley & Sons.

Treleaven, P., Galas, M. and Lalc, V. (2013). Algorithmic Trading Review. In: *Communications of the ACM.* Volume 56 (11), 76-85.

Yan, X. (2009). *Linear Regression Analysis: Theory and Computing*. Singapore: World Scientific.

VBA Macros for Tick Data Copying

Part 1

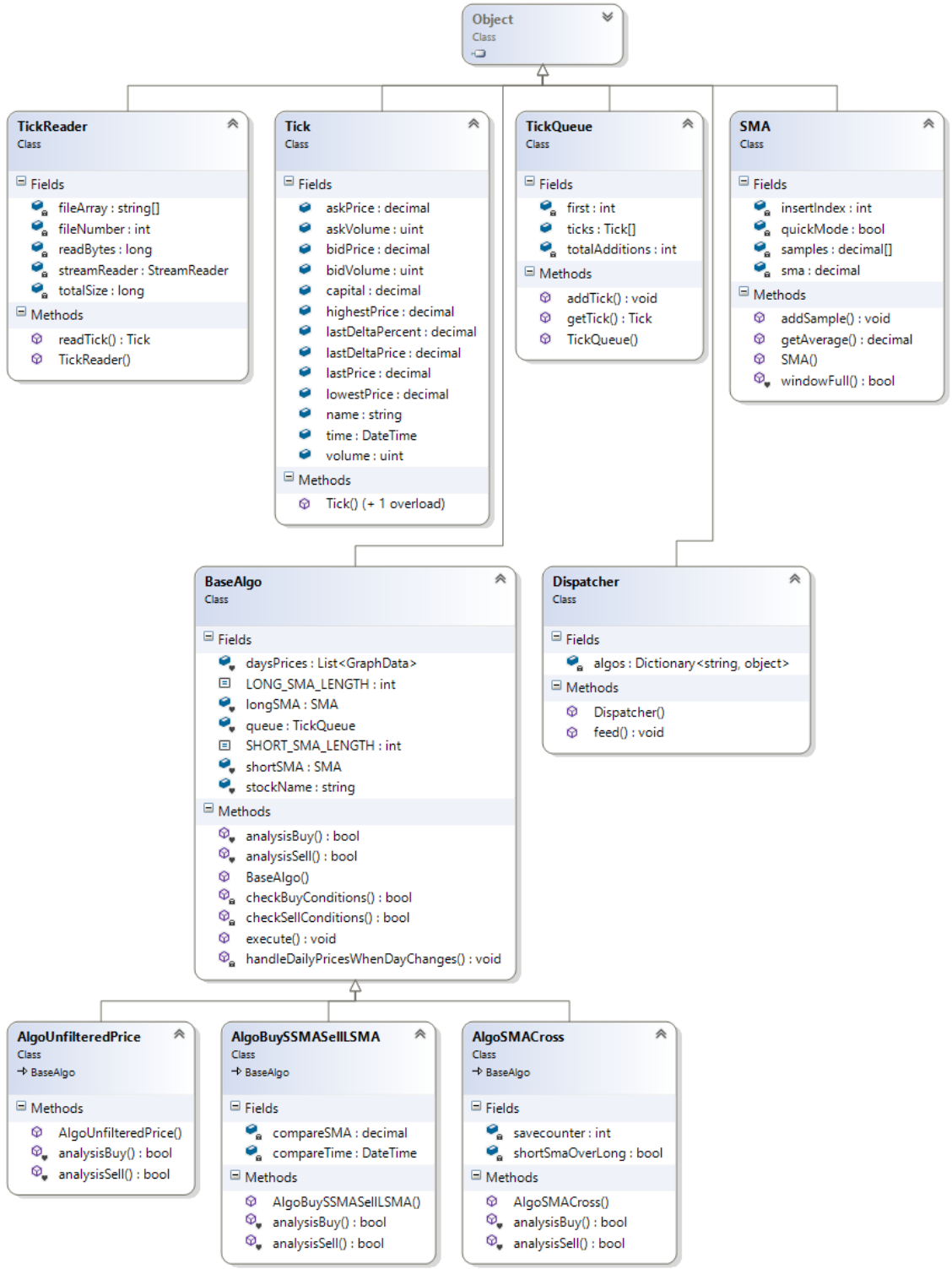
```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Copies each stock row to own sheet and the total sheet if the row is updated.
Sub copyDataToSheets()

  Dim name As String
  Dim srcRow As Long
  Dim destRow As Long
  Dim totalDestRow As Long
  Dim Src As Range
  Dim Dst As Range

  rowCount = getLastRowInSheet(SOURCE_SHEET) - 1
  For srcRow = 2 To rowCount

    If isStockUpdateTimeChanged(srcRow) = True Then

      ' Store stock name for referring destination sheet
      name = Sheets(SOURCE_SHEET).Cells(srcRow, NAME_COL).Text

      ' Get number of new row in dest sheet
      destRow = Sheets(name).Cells(Rows.Count, 1).End(xlUp).row + 1

      ' Copy to stock sheet without clipboard. However, formatting is lost
      Set Src = Sheets(SOURCE_SHEET).Rows(srcRow).Resize(1, 16)
      Set Dst = Sheets(name).Rows(destRow).Resize(1, 16)
      Dst.Value = Src.Value

      ' Change time format and insert current date in the last column
      Call fixTimeFormatAndInsertDate(name, destRow)

      ' Find last row in Total sheet
      totalDestRow = getLastRowInSheet(TOTAL_SHEET) + 1

      ' Copy to total sheet without clipboard. However, formatting is lost
      Set Dst = Sheets(TOTAL_SHEET).Rows(totalDestRow).Resize(1, 16)
      Dst.Value = Src.Value

      Call fixTimeFormatAndInsertDate(TOTAL_SHEET, totalDestRow)

      ' Got data waiting to be saved
      unsavedData = True
    End If

  Next srcRow

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Returns last row number of given sheet.
Function getLastRowInSheet(name As String) As Long

  getLastRowInSheet = Sheets(name).UsedRange.SpecialCells(xlCellTypeLastCell).row

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Sets time stamp formatting and inserts current date
Sub fixTimeFormatAndInsertDate(sheet As String, row As Long)

  ' Change time format
  Sheets(sheet).Cells(row, TIME_COL).NumberFormat = "h:mm:ss"

  ' Regenerate date
  Sheets(sheet).Cells(row, DATE_COL).NumberFormat = "@"
  Sheets(sheet).Cells(row, DATE_COL).Value = getDateString

End Sub
```

Part 2

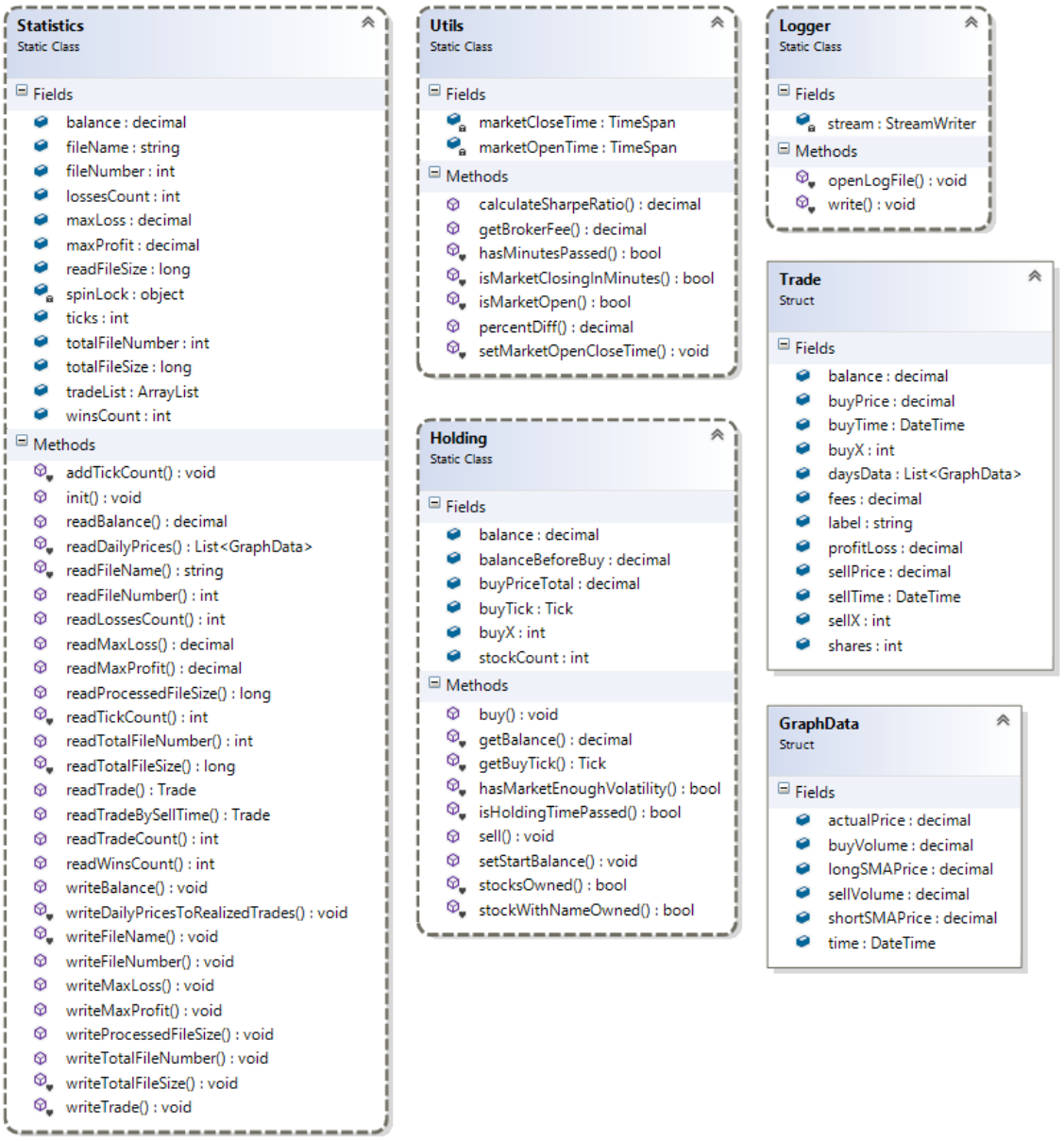
```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Compares update time from source sheet and stock specific sheet
Function isStockUpdateTimeChanged(srcRow As Long) As Boolean

  Dim srcTime As String
  Dim sheetTime As String
  Dim name As String

  ' Get update time in source sheet
  srcTime = Sheets(SOURCE_SHEET).Cells(srcRow, TIME_COL).Text

  ' Get update time in destination sheet
  name = Sheets(SOURCE_SHEET).Cells(srcRow, NAME_COL).Text
  sheetTime = getUpdateTimeByStockName(name)

  ' Compare timestamps
  If StrComp(srcTime, sheetTime) = 0 Then
    isStockUpdateTimeChanged = False
  Else
    isStockUpdateTimeChanged = True
  End If

End Function


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Returns update time from stock's own sheet.
Function getUpdateTimeByStockName(name As String) As String

  Dim rownbr As Long

  rownbr = getLastRowInSheet(name)

  ' In the very beginning sheet has only heading row
  If rownbr = 1 Then
    getUpdateTimeByStockName = "99:99:99"
  Else
    getUpdateTimeByStockName = Sheets(name).Cells(rownbr, TIME_COL).Text
  End If

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Gets current date as string. Use US style that is naturally sortable.
Function getDateString() As String

  getDateString = Format(DateTime.Now, "yyyy-MM-dd")

End Function
```

Backtester Application Classes

Dynamic classes



Static classes and structures



User interface classes