Master Thesis
Fachhochschule der Wirtschaft (FHDW)

A Concept for Automated Trading in Value Investing

Ein Konzept zum automatisierten Handel im Value Investment

Created by:
Edwin Stang

Lecturers:
Prof. Dr. Wilhem Nüßer
Prof. Dr. Heiner Langemeyer

Handed in on:
March 22, 2013

Contents

List of Figures

List of Source Code

List of Tables

Nomenclature

4M ..... Meaning, Management, Moat, Margin of Safety *(Page 11)*

AI ..... Artificial Intelligence *(Page 5)*

API ..... Application Programming Interface *(Page 44)*

Ask ..... This represents the price being asked by people who want to sell a specific instrument. *(Page 33)*

Backtest ..... This is a simulated run of a trading system through historical data with the aim of determining if the implementation can produce profits. *(Page 20)*

BAG ..... Big Audacious Goal *(Page 15)*

Bar ..... A Bar is a data point for an instrument that includes time, open-, high-, low-, close-prices and volume. They summarize ticks over a specific time (Time-Bar), count (Tick-Bar), price (Renko-Bar), ... *(Page 33)*

Bid ..... This represents the price being offered by people who want to buy a specific instrument. *(Page 33)*

Building Block ..... This is a part of a strategy that addresses a specific domain of decisions. For example Money Management, Entry, Exit, Stop Loss, ... *(Page 1)*

BVPS ..... Book Value per Share *(Page 12)*

CDS ..... Credit Default Swap *(Page 8)*

CFD ..... Contract for Difference *(Page 8)*

Decision Point ..... This represents a building block of a strategy which is divided into its smallest possible grade of freedom. *(Page 33)*

Design Parameter ... A parameter that is used during strategy development to automatically try different rules during a testing iteration. *(Page 62)*

DNA ..... Deoxyribonucleic acid which holds the genes of life forms and characteristics of genetic algorithms. *(Page 67)*

End-of-Day Data ... This includes one bar for each day (One-Day-Time-Bar) as a data point for an instrument. *(Page 2)*

Entry ..... The starting point of a trade. Also used as a name for a type of rule that initiates this. *(Page 35)*

Entry Filter ..... A rule that overrides entry rules to disable them. Thus being able to filter entries. *(Page 35)*

EPS ..... Earnings per Share *(Page 12)*

ETF ..... Exchange Traded Fund *(Page 6)*

Exchange ..... This is the place where stocks or other instruments can be traded. Historically this was a place where people met to do their trading, though today this is effectively a digital system connected with other such systems. For example a stock exchange or a foreign exchange. *(Page 6)*

Exit ..... The end point of a trade. Also used as a name for a type of rule that initiates this. *(Page 35)*

FCF ..... Free Cash Flow *(Page 12)*

Fitness Criterion .... A mathematical function that is used to optimize an AI for. *(Page 32)*

Forex ..... This means Foreign Exchange and is an exchange that manages currency pairs markets. For example EURUSD. *(Page 7)*

Forward-Test ..... This is a simulated run of a trading system that runs on live data arriving from the market. This makes the test much slower than a backtest, but more authentic to live conditions. *(Page 22)*

Indicator ..... A quantifiable measurement of some sort that can be used as a parameter or a basis for a rule. *(Page 24)*

Instrument ..... This might be a stock, a currency pair or a futures contract. Thus being a comprehensive word for tradeable things that have their own markets. *(Page 4)*

Intraday Data ..... This includes ticks and bars that occur during price movement during the trading day of an instrument. *(Page 2)*

Intrinsic Value ..... This is the value a company is actually worth in the eyes of the investor, disregarding a current over- or undervaluation by the market price. *(Page 4)*

Live-Run ..... This is an actual run of a trading system on real money and with the effect that trades may directly affect the market

by changing the volume, ask/bid of the instrument by trading actions, which is impossible to simulate in testing conditions. *(Page 21)*

Long ..... This represents a trade where the entry is a buy order and the exit is a sell order. This is done for speculation on rising prices. *(Page 14)*

M&A ..... Mergers and Acquisitions *(Page 11)*

MA ..... Moving Average *(Page 41)*

Margin of Safety .... A difference to the calculated value of the company at which it is bought to ensure a cushion of price movement that accounts for imprecise value calculations. *(Page 16)*

Market ..... A market is a place where demand and offer meet. This might be a tradeable stock, a currency pair or just the individual vegetables of the supermarket next door. Sometimes exchanges are themselves called markets, but this depends on the grade of abstraction that is used. *(Page 6)*

Market Price ..... This is the price that a stock can currently be traded on the exchanges. *(Page 4)*

Modular Strategy API An API that is designed to allow the reuse of strategy rules and to make strategy development more convenient. *(Page 44)*

MOS ..... Margin of Safety *(Page 18)*

Optimization Parameter A Parameter that is used for optimization purposes to fit the strategy to the current market conditions. *(Page 62)*

OR BO ..... Opening Range BreakOut *(Page 36)*

Order ..... This is an action performed on an instrument to change an investors position in the given market. *(Page 5)*

OTC ..... This means Over-The-Counter trading and specifies trading that is not done over an exchange, but just between a broker or a bank and a trader. *(Page 8)*

P/E ..... Price per Earnings Ratio *(Page 16)*

Parameter ..... A decision that can be made before running an automated strategy. This can be numerical, boolean or an enumeration. *(Page 24)*

Payback Time ..... Counting how many years a company's earning require to accumulate until the market capitalization is reached. *(Page 17)*

PDF ..... Portable Document Format *(Page 48)*

PTBV ..... Price to Tangible Book Value *(Page 17)*

ROIC ..... Return on Invested Capital *(Page 12)*

Rule ..... A condition or behavior component of automated strategies that represents decisions that are made by the automated strategies. *(Page 5)*

Safety Price ..... This is a 50% discount off the sticker price. *(Page 16)*

SDK ..... Software Development Kit *(Page 44)*

Short ..... This represents a trade where the entry is a sell order and the exit is a buy order. This is called an uncovered sale, because the person doing the initial sell, does not possess the shares being sold, but wants to buy them later when the price has fallen. Depending on the country and the instrument in use, this type of trade might be unavailable (stocks in Germany). *(Page 6)*

Signal ..... Either buy, hold or sell, depicting the action to be taken on an instrument. *(Page 35)*

SL ..... Stop Loss *(Page 38)*

Sticker Price ..... This is the price the investor calculates for a stock when counting in the future growth over ten years and calculating back with a 15% expected yearly return. This leads to a price estimation that should represent the current "recommended retail price" for the stock. *(Page 16)*

Stock ..... This is a tradeable paper which represents an investment in shares of a company. *(Page 2)*

Strategy ..... This might be a manual or automated approach to trading markets or instruments. *(Page 1)*

Tick ..... This is a data point that represents the smallest possible measure of change in price movement. It gets updated when changes in offer and demand occur on an instrument in an exchange. It is composed of time, ask-, bid-price and volume. *(Page 33)*

TP ..... Take Profit or Profit Target *(Page 38)*

Trade ..... This is an action an investor performs in a market that is composed of an entry order and an exit order to effectively buy/sell some amount of an instrument. *(Page 1)*

Trading Platform ... This is a sort of framework and tool set with which trading systems can be developed and run with. For example MetaTrader 4, TradeStation or JForex. *(Page 1)*

Trading System ..... This is an implementation of a trading strategy that is thus automated. This is often called a Strategy in JForex or an Expert Advisor in MetaTrader terminology. Thus it depends on the Platform being used. *(Page 1)*

TradingSystemLab .. A fully automated strategy development platform that is based on genetic programming. *(Page 60)*

UML ..... Unified Modeling Language *(Page 45)*

Underlying ..... In derivative markets the underlying is the stock, commodity or currency pair on which a contract operates. *(Page 7)*

Value Investing ..... This is a long term trading paradigm that invests in undervalued stocks in the hope of the stock being able to recover. Often also falsely said to be a type of speculation. *(Page 10)*

Variability Model ... This is used to visualize decisions that can be made on a blueprint of a product to create an instance of it by binding specific variants the blueprint supports. *(Page 40)*

Volume ..... This is the amount traded during a bar or a tick and represents how much demand has been filled by offer on the market during that time. *(Page 72)*

Walk-Forward-Analysis A testing type where backtests are nested to provide optimization cycles during a backtest to measure the robustness of a strategy. *(Page 29)*

1 Introduction

This thesis is about creating a concept for automated trading in value investing. This means integrating two different worlds:

- the functional topic of long term trading done by professional investors
- and the technological aspects of automating that by the help of computer systems

Luckily there are already a lot of systems out there that automate trading. Also there are many documented methods and approaches out there with which professional investors do their business. Most of the automated trading system developers choose some of these professional methods, implement those in some trading platform as an automated strategy and run that on the financial markets. Maybe the developer backtests the strategy on historical data and optimizes it to become more confident in it. Nevertheless after some time the strategy may produce losses, the developer starts to doubt his selection of building blocks and thus goes back to the drawing board to make changes or to invent something new.

This trial and error approach is very time consuming and not very effective. Because of that, this thesis goes a step further and develops a concept with which the strategy development process can be automated to a high degree. The system behind this concept should be able to combine strategies from building blocks, parameterize those and backtest that. Multiple strategies can then be evaluated and the trading system developer then just has to choose with which strategy he wants to trade his specific market. Or he can enhance the system by developing more alternative building blocks that are then automatically tested for fitness in various possible strategy combinations.

To make this concept technically possible, the author chose the boundary of value investing, which is a slow-paced form of trading. This ensures fast backtests for many strategy combinations with end-of-day prices on stocks instead of millions of data points which are being used in intraday trading. Another aspect that comes into play with value investing is the fact that fundamental data (like business reports) becomes relevant for the strategy itself, which is completely ignored by most of the automated trading systems today and is not even supported well in the trading platforms they are implemented in. Thus it is explored if and how automated strategies can include the fundamental data and decide for themselves which stock should be invested in, while following the value investing paradigm.

1.1 Objectives

The resulting objective of this thesis is to construct a concept for a platform that can do the above mentioned two tasks:

1. automatically generate strategies from building blocks and test them
2. help in or automate the decision processes of choosing value investments and strategies to trade them with[1]

In the process of going further into the design, the author chooses concepts and technologies with which that platform can be realized. This thesis as a whole should function as a vision and drawing board which can later be used to develop that platform, while not going into detail about *"how"* this can be done, but more about *"with what"* this can be done. Also out of scope here is in *"which business model"* the platform can be used, because there is another master thesis planned by a fellow student with that specific topic.

[1] This literally means automating the value investing portfolio management strategy and the strategy development process as far as the following chapters about those topics describe that.

1.2 Approach

People who are most interested in this thesis might be trading system developers who want to change their approach of developing strategies. But also people with interest in new ways of investing, or technically adept persons might be interested in this work.

Because of the vastness of the two discussed worlds of trading and technology, this thesis will need to balance the depth of detail in which topics can be discussed. Thus compromises will have to be made to stay in the limits of this work. The reader is supposed to have common knowledge about investing money with a broker and should have some understanding about software architecture and frameworks. Any further or specific knowledge required will either be included in the thesis or references will be made to sources with more detail.

This work is split into the following parts:

- first financial topics and value investing will be discussed
- then automated trading and strategy development will be discussed
- later the concept for the platform will be developed
- and finally the conclusion should tell if such a platform is feasible or not

The first two parts will be mostly analytical in nature, while the later parts are going to be more practical. Though without being too strict in the distinction between those two styles, since references will need to be made in both directions.

2 Finance and Value Investing

This chapter classifies finance in the direction of value investing and explains what that strategy is comprised of. Also there will be some analysis on how this can be improved to be semi-automated.

2.1 Trading

Trading in general[2] is buying and selling of "things". Many of these "things" can be traded electronically via the internet, via telephone or direct communication on the street or in supermarkets. Trading in the context of this thesis only looks at the electronically traded "things" over the internet by using special platforms that display price movements and allow traders to make money from price changes. These "things" are here defined as financial instruments which have their own financial markets where demand and offer meet to conclude transactions, which are called trades.

To make money from price changes, traders need to have a plan or a scheme with which they do their trades. This is called a strategy and is used to exploit market inefficiencies or dynamics to gain profit. Sample strategies might be to follow trends in price movements, looking for patterns that tell in what direction the price moves, finding companies that have gaps to the index movement in which they are listed, trading for arbitrage by exploiting minimal price differences between markets of the same instrument and many more. A common saying is that as many traders there are, as many strategies there will be, since every trader has his own style of doing his business. Though this depends on the level of detail in which strategies are viewed. In the context of this thesis, strategies are seen as common principles on which many traders can operate on.

Value investing is one special strategy that tries to trade companies that are *"undervalued"* by the markets. This is done by finding the intrinsic value of a company and comparing it to the market price. How this actually works is explained in detail in the following chapters, but first other financial classifications need to be made.

[2]For the German reader, [Voigt, 2012] provides an excellent introduction to trading in general. Another introduction book in English is [Griffis and Epstein, 2009].

2.2 Actors

The actors involved in trading are the following:

- **Trader:** This is the person who decides to buy or sell securities on financial markets. He may do this according to some strategy.
- **Broker:** The trader has an account at a broker which manages his trade positions and equity. The broker connects the trader with the financial markets and makes data available about those.
- **Trading Platform:** This is a software that electronically connects the trader to the broker. It displays information about the financial markets, transmits orders to the broker and gives feedback to the trader. The trader can also enhance this software by developing automated strategies that do parts of his work or help him with it.

There are also other actors that are indirectly involved, like media, advisors, other traders, companies, banks, governments and much more. Though for this thesis and for the concepts described herein, it suffices to look at these three actors because they are the subjects of further analysis. Also one might argue if the trading platform is an actor, since it is not some person involved in decisions being made. This is not the case here, since the aim of automation is to give the trading platform responsibility about the decisions the trader normally makes. The ultimate form of automation would be to have an artificial intelligence (AI) that replaces the trader and is thus as itself being personified. Though despite this being very attractive, this goal has not been reached yet by humankind and this thesis will restrict itself in only trying to develop concepts about the automation of rules. Thus helping the trader instead of replacing him.

2.3 Financial Markets

When speaking of finance from a trader's perspective, one has to know in which financial market he operates. The tradeable financial markets can be divided into the following categories[3]:

- **Fund Market:** *Hire a Professional in a Collective Investing Scheme*
  Fund managers invest the money of many private investors according to some strategy or portfolio plan. Private investors have to pay fees for this service and have no control on the actual trading[4]. There are also exchange traded funds (ETF), which as the name states, can be traded on an exchange. Though these funds are mainly passive ones that try to reflect the movement of some benchmark. Actively traded funds are more of a rarity on an exchange. When trading ETFs, one has to pay transaction fees additionally to the fund management fees.

- **Bond Market:** *Loans to Governments or Companies*
  Here one can issue debt and receive an interest rate as a fixed income or one may trade debt securities which are called bonds. The interest rate or the security price mostly depends on the risk involved in the bond and rises with it. Mostly when trading debt securities, one speculates on the ability of the issuer to repay his debt.

- **Stock Market:** *Company Shares like Apple or Microsoft*
  This is a way for companies to raise money. They publish shares of their company on stock exchanges and receive the money from their investors who thus become company shareholders. The shares are then traded among the shareholders, while the price reflects the health and business outlook of the company. In Germany, short selling stocks has been banned by law [Assmann and Schneider, 2012, §30h WpHG] since 2010. Thus one can only speculate on rising prices when directly trading stocks there.

[3] For a detailed view on the financial markets and how they operate, one may refer to [Mishkin and Eakins, 2008].

[4] This is much like Private Banking, with the difference that one can hire a professional with much less capital available, because the fund manager operates on the pooled money of many individuals instead of only ones. For Private Banking one needs a few hundred thousand dollars in capital to invest, while only a few hundred dollars are required for investing in funds.

The following are leveraged markets where it is possible to make a huge profit or loss on minor price changes in the underlying. Or one can trade huge quantities of something without having the actual complete required capital to acquire those, because a loan or contract is coupled with the trade.

- **Foreign Exchange:** *Currency Pairs like EURUSD*
  Changes in exchange rates of currencies are speculated on in these markets. Actually to make a profit, one has to use a brokers margin credit to trade with large quantities of money instead of just ones direct deposit. This allows that a 0.0001 change in the exchange rate may result in hundreds of profit or loss. The leverage and thus the effective volatility and risk can be adjusted on a per order basis in these markets.

- **Derivative Markets:**
  These markets do not trade actual assets, but contracts between parties based on those. The prices are only derived from, or depend on the price of the underlying assets.

  - **Futures:** *Commodities like Oil, Gold or Rice*
    Here one buys something with a fixed price in the future without wanting it to be actually delivered. Instead it is hoped that the actual price of the commodity rises or falls until the given date to be able to make a profit depending on the position that has been taken initially. Prices are often set for specific quantities of units of the underlying in contracts. The risk can be adjusted by trading fractions or multiples of those contracts.

  - **Options:** *Commodities and Stocks*
    This is similar to futures with the difference that instead of having the obligation to buy or sell on the given date, one has the right to do it. Thus if the price goes into the wrong direction, losses can be minimized. One can adjust his risk by choosing among options with various fixed leverage values which are offered by banks.

- **Contracts for Difference (CFDs):** *Indices and Most of Above*
  With CFDs one speculates on price changes in an underlying without actually possessing it. This is very similar to futures, with the difference that the contract has no expiry date and it is only affected by the price movement of the underlying. With CFDs it is possible to speculate on falling prices for stocks in Germany or to trade indices, which represent a collection of stocks. The leverage can be chosen, just like in the foreign exchange markets, with the difference that the broker only gives virtual margin credit, because he does not actually acquire the underlying for the trader. These markets are unregulated and traded over-the-counter (OTC) between the broker and the trader, in contrast to being traded on an official exchange.

- **Swaps:** *Interest rate-, Currency-, Commodity-, Credit Default Swaps*
  Two parties exchange cash flow streams with each other. For example changing a fixed rate loan into a floating rate loan for interest rate swaps. Credit default swaps (CDS) are another example where one speculates that a credit will not default after acquiring the swap for it. In the last financial crisis around 2008, CDS became a bit unpopular in the media. This was because of credit ratings that have become too optimistic and thus were too risky without showing it. This resulted in banks specialized in this field going bankrupt. Nevertheless, this does not change the fact that it can be traded profitably today. This type of trading is also done OTC, but with higher entry barriers than the other markets. Thus this might be more interesting for companies than for private investors.

This thesis is going to further focus on the stock market, since this is the basis for the value investing strategy. In itself, the platform should differentiate between two disciplines[5]:

- **Portfolio Management Strategy:** which selects investments and decides when to discard them
- **Trading Strategy:** which handles the buying and selling of the investment, while it is in the portfolio

Value investing operates on stocks and reaches through both of these disciplines (with a bias on portfolio management). Thus the platform should aim to automate both. Nevertheless, other markets might also be interesting for the platform and it should thus support generating trading strategies for the other types of markets as well, even if this is not in the scope of this thesis and cannot be elaborated completely.

But the decision that can be made here is, that the part about the generation of trading strategies should be usable on various types of markets without having to change the code to support this in a later stage of development. This is possible because the trading operates on common principles among these markets.

[5]Even though they both are parts of strategies in general.

2.4 Value Investing Strategy

The idea for this platform originates from reading [Town, 2007], where the author Town wrote an introduction to value investing and introduced a simplified approach to the selection process of viable investments. The approach is based on the original work by Graham in the books [Graham and Dodd, 2008] and [Graham and Zweig, 2003], where value investing was first defined. Town explains how one can find interesting investments and make a decision with the help of online resources[6] and simple calculations. In another book [Town, 2010], he enhances his approach by a few more calculations.

Town's portfolio management strategy will be discussed further in the following subchapters[7]. After that, it will be shown how Town's process has been adjusted and semi-automated in a system that is already implemented by the author of this thesis.

2.4.1 Basics

The general idea behind value investing is, that the price of a stock does not always represent the actual value of the company behind it. Stock prices may rise or fall without real changes in the prosperity of the company, just because investors feel differently for some time or the market in general goes through a boom or recession phase. Value investing tries to make investestment decisions based on the intrinsic value of the company. Investors do not just buy shares, but think about becoming the owner of the business when doing so. Thus they make a decision based on fundamental analysis[8] and wait for a buy opportunity when the price is low due to market inefficiencies.

The concept is in itself a contradiction to the widely known efficient market theory, which states that the price of an asset is always the true valuation of it. This is because of the efficient distribution of information that is instantly transformed into price action. Though market efficiency is itself today criticized in the scientific community, even if no conclusion has been finally made[9].

[6] Online resources like MSN Money, Yahoo Finance, Google Finance, ...

[7] Town also wrote about two trading strategies in his books, but those will not be discussed here in favor of the trading strategy generation process that seems more promising.

[8] An introduction to fundamental analysis is provided by [Krantz, 2009].

[9] For examples, one might look into the scientific papers [Lo, 2007] and [Malkiel, 2003]. In that

2.4.2 The 4M Approach

Town's approach is divided into four steps of analysis:

1. **Meaning:** *What company would you like to buy?*
   One can search for companies by looking at the things one buys and what he is interested in. If one does not know where to find companies for which stocks are available, one might look at industry listings for stocks and search for companies that are familiar. It should be a company that exists for a long time and which has a loyal customer base. One should be able to identify himself with the company and should feel proud about the possibility of being the owner of it. The company should have a story of success to it and should represent the values one admires. If the company is in the field of interest to the investor, the investor should be able to follow the company news without much effort and should be able to tell if the business outlook changes for the worse, or the better during the holding time.

2. **Moat**[10]**:** *Does it have a sustainable competitive advantage?*
   After having chosen a company, the investor should be able to easily tell what advantage that company has over its competitors. It should be something like patents, valuable assets or a secret that is hard to acquire for the competition. For things that qualify as this, one might look into analyzing the company with Porter's five forces model [Porter, 1998a] or value chain model [Porter, 1998b]. Another approach might be to look if the company has a special business model or is positioned in a market where it has a monopoly status like it is described in [Kim and Mauborgne, 2005] as a blue ocean strategy.

context, evidence is gained in [Penteado, 2013] that behavioral finance is a new concept worth studying.

[10] The naming of *Moat Analysis* comes from the middle age where castles were protected by large moats as a defense to besiegement. The middle age is also often used in the world of mergers and acquisitions (M&A) to shortly describe actions which companies take for or against one another.

After identifying the competitive advantage, the investor has to verify that it is sustainable by doing fundamental analysis on company reports over the past ten years if possible. The company reports can be viewed either in tabular form on websites or one has to skim through the original reports to get the numbers of interest. At this step it is also advisable to read the company reports to get a better understanding about how the company sees itself. The numbers one should extract for the past ten years annually are the growth rates for:

- Return on Invested Capital (ROIC)[11]
- Sales
- Earnings per Share (EPS)
- Equity or Book Value per Share (BVPS)
- Free Cash Flow (FCF)

These values should be above 10% on a yearly basis. Also the company's long term debt should not be larger than three times its FCF, so that one can be sure that the company could be debt free in a short time if it wishes to repay it.

To get a better overview on those values, one might want to look at them on a chart basis, like it is done on a website called *stock2own.com*. It shows calculations based on Town's approach for specific companies where an investor is interested in. In figure 1, it is visible that Microsoft is trending upwards in the last few years for example. A look on the quarterly financial data might give a clearer look on the short term movement of the company. It might be able to explain a price drop in the stock that is reflected in bad quarterly reports, which is not yet visible in an annual report.

[11] ROIC already is a growth rate, thus one does not need to calculate the growth rate of the growth rate.

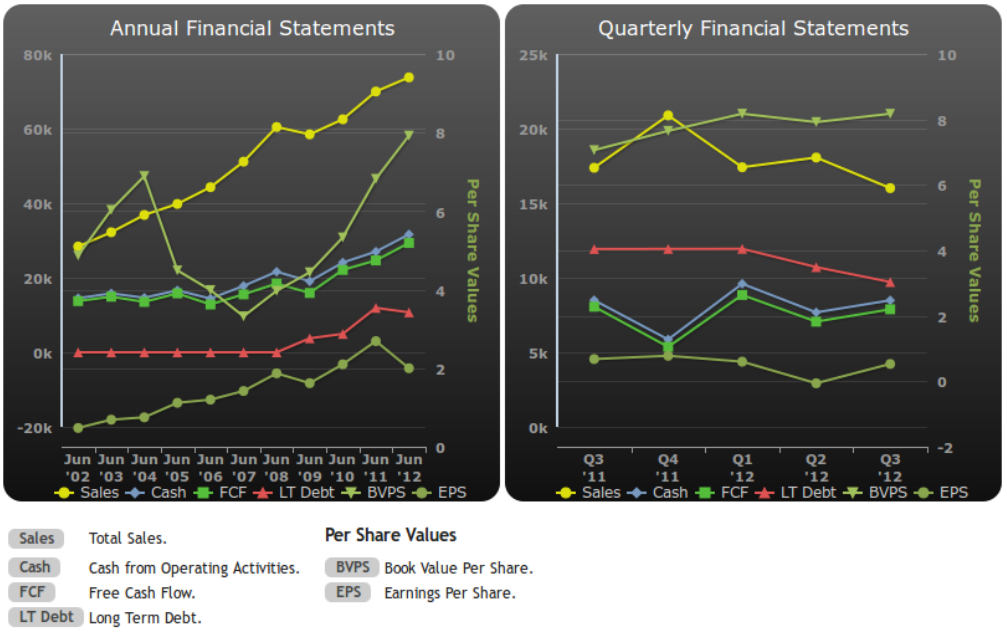


Figure 1: Growth Rates Chart for Microsoft from [Stock2Own, 2012]

When seeing that, one might think this could be a good investment. But when looking at the actual growth rates over the past then years in a tabular form (as visible in figure 2), it does not look like it is good enough according to the above 10% growth rule. There are too many years where this was not achieved for the various numbers.

Growth Grade is 57.20

Result[1]	Item Name	Latest Date[2]	1 year	2 year	3 year	4 year	5 year	6 year	7 year	8 year	9 year	10 year
bad	Equity	Jun '12	16.13 %	21.90 %	21.28 %	18.85 %	18.99 %	12.11 %	8.45 %	1.76 %	3.08 %	4.98 %
bad	EPS	Jun '12	-25.65 %	-2.41 %	7.28 %	1.69 %	7.09 %	8.89 %	8.64 %	13.04 %	12.55 %	15.34 %
bad	Sales	Jun '12	5.40 %	8.62 %	8.05 %	5.10 %	7.60 %	8.87 %	9.21 %	9.06 %	9.64 %	10.02 %
bad	Free Cash Flow	Jun '12	19.00 %	15.19 %	22.58 %	12.31 %	13.55 %	14.78 %	9.24 %	10.16 %	7.81 %	7.88 %
fair	Cash from Operating Activities	Jun '12	17.16 %	14.62 %	18.44 %	9.99 %	12.19 %	14.01 %	9.64 %	10.12 %	8.02 %	8.10 %
good	ROIC		18.20 %				30.60 %					
bad	Gross Profit	Jun '12	3.36 %	5.92 %	6.68 %	3.58 %	6.81 %	7.39 %	7.55 %	8.05 %		
			bad	bad	bad	bad	bad	bad	bad	bad	bad	bad

[1] Legend:

BAD There is at least one value less than 8% in the row or column

FAIR All values in the row or column are not less than 8%

GOOD All values in the row or column are not less than 10%

WARNING Growth rate is good, current value is positive, however underlying historical value is negative. You may want to review 'Raw Financial Data' section for details.

Figure 2: Tabular Growth Rates for Microsoft from [Stock2Own, 2012]

Anyway, if an investor is still thinking that the company might grow in value in the long term, or thinks it is a stable investment, he might still be interested in investing in this company. It is after all a very subjective decision on how to interpret these numbers. But it should be viable to guess that with these growth rates, the company might not be able to bring a yearly 15% rate of return. As this is what value investors aim at, when deciding to go for an investment.

This analysis on the numbers should be able to validate the previous competitive advantage analysis. If it does not, one might have made an error in the assumptions being made about the company's fortune. This analysis can also be made fast with the websites that specialize on these calculations. Though this only counts if their interpretation or enhancements on these calculations fit the investors views.

And anyway, an investor should check multiple sources to see if the same view is reflected on all sources of information. One such other source might be the growth estimates of professional analysts that are published regularly for large companies. Another way to check for errors here might be to look for hints in insider trade reports. If the business owners buy shares or use long options on their own stock, this might be a hint that an upward movement is going to last. If they are selling right now, it might be a warning signal.

3. **Management:** *Is the management good and durable?*
   The management consists of those people who are in charge of representing the investors as the owners of the company. They should act in their interest and should focus on increasing the long term value of the company. Management can be analyzed by reading the company reports, listening to the regular earnings conference calls, reading news and doing background research on these people.

The following questions are an excerpt of what an investor could answer during the research to evaluate the management:

- Is the management trustworthy and sympathetic? Do they act ethically and morally correct?
- What qualifications do they have for their job?
- Do they have a track record of proper management?
- Is the management under high fluctuation? Do they have a lot of experience with the company? Did they eventually stick with it since it has been founded?
- What big audacious goal (BAG) does the management follow? What motivates the management?
- What were the past failures and what have they learned from them? Are they open with their failures? Do they hide things?
- What are the greatest upcoming challenges for the company?
- Can they manage risks properly and take needed risks to accomplish the company's goals?
- Is the management owner oriented? Do they waste money?
- Are the managers more interested in their career than in the success of the company? Do they only consider short term targets?
- What does the management earn? Do they milk the company for their own benefit?
- How much insider trading has been done on the company? In what direction did they think the stock price goes compared to how it ended up?

Rating these answers is a very subjective matter. The result should be an awareness about whether the management is trustworthy to the investor. The investor should reconsider the investment if too many negative signals have been found here.

4. **Margin of Safety:** *Is it cheap right now?*
   In value investing, market inefficiencies are personified by Mr. Market, which represents a bipolar person that may be at times overly optimistic or totally depressed about the future of a stock or a market in general. Thus he either lowers the price of a stock or makes it rise. The value investor tries to buy stocks when Mr. Market is depressed and sell the stocks when he is optimistic, thus buy low and sell high. In fact, Mr. Market might be seen as a way to simplify the market sentiment that is built by individual investors acting under the same influence at times. This may be caused by news, changes in the business or trust in the company. This might result in too many people buying or selling a stock and thus making the price diverge from the intrinsic value of the company.

   In this step, the investor tries to find out whether Mr. Market is depressed about the company so that it is undervalued and represents a buy opportunity. If this is not the case, the investor might still put this company into his watch list and follow the news and price development until a buy opportunity is developed. Even though this might take a while.

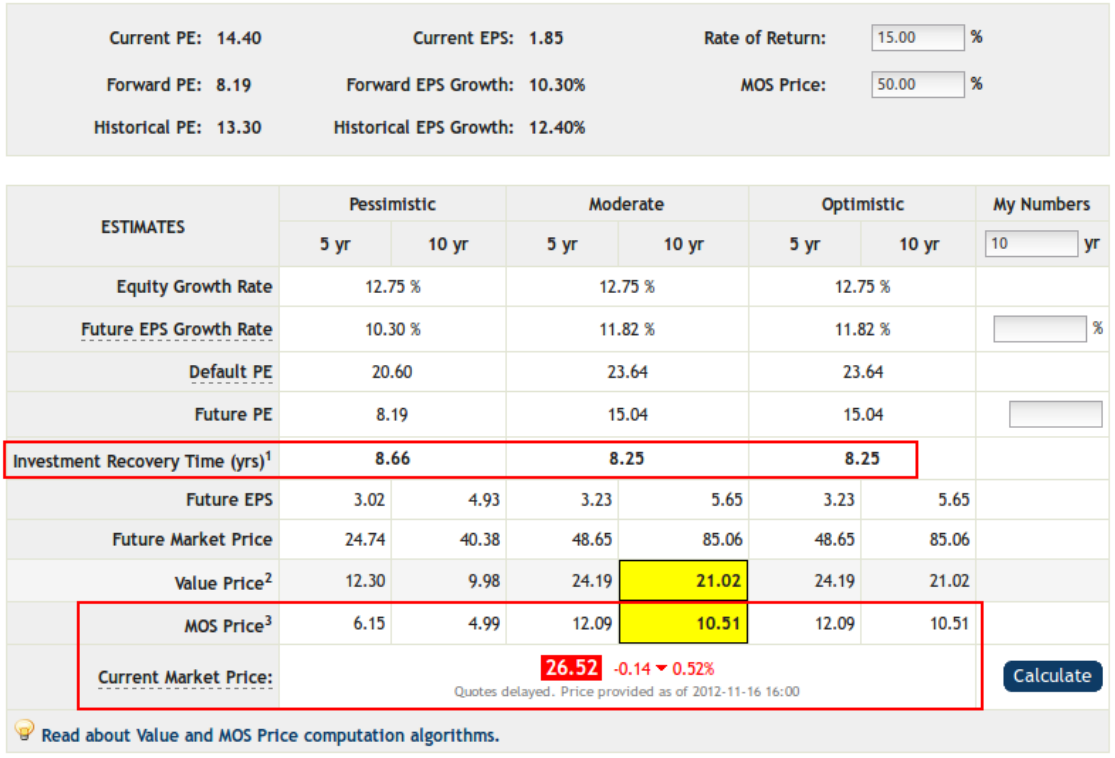
   There are two calculations that should be used together to determine a buy opportunity:

   - **Safety Price:** *What would be a cheap price for it and how does it compare to the current price?*
     This is calculated by taking the current EPS and multiplying it by the price per earnings ratio (P/E). Then making that current price based on earnings grow for ten years into the future with the average annual equity growth rate of the past ten years. This then is the future price that is expected if the company can keep up with the growth as it has been able to. To get the sticker price (expected todays price), the future price has to be shrunk ten times by the 15% yearly rate of return this strategy aims for. This price can be seen as a price tag that reflects the expected growth of the company if it stays stable. Since there can go wrong a lot with such a stable valuation, the investor only buys with a margin of safety of at least 50% as a discount to the sticker price. That 50% discount is called the safety price. If the current price is at or lower than that price, the company is seen as currently undervalued.

- **Payback Time:** *How many years would it take to get the invested money back based on profit?*
  This can be calculated by taking the current market capitalization of the company and compounding the current yearly earnings until the market capitalization is reached, while making the earnings grow with the previous years growth rate of the earnings. This tells how many years it would take to repay oneself, if one buys the whole company with its current market capitalization and would thus be able to receive all its future earnings from then on to repay himself. Thus this is a way to determine after how many years a private investor would be able to earn from his investment if he bought the whole company. Another way to look at this is: At how many multiples of yearly earnings is the company currently priced? If the value is ten or less, the company is eligible to be bought. If this is six or less, it would be very interesting to buy it. This works as another safety check based on the idea that earnings are a large factor in the increase of a stock price. This will filter out companies that are unprofitable or which are overvalued by high expectations of investors.

Despite these calculations here, it might be interesting to look at the company reports directly and include other valuation methods, like price to tangible book value (PTBV), or compare the book value and EPS with other competitors of that company to determine if the company might be undervalued in its own market segment. Also it might be interesting to look at the competitors to see if the company is undervalued because the moat is attacked by its competitors. If that is the case, one might rethink his evaluation in the first two steps, since there seem to be superior companies out there compared to the one at hand, which might lead to its demise.

As with the moat calculations, there are websites which do these calculations for the investor as seen in figure 3:



Current PE: 14.40		Current EPS: 1.85			Rate of Return:	15.00 %	
Forward PE: 8.19		Forward EPS Growth: 10.30%			MOS Price:	50.00 %	
Historical PE: 13.30		Historical EPS Growth: 12.40%					

ESTIMATES	Pessimistic		Moderate		Optimistic		My Numbers
	5 yr	10 yr	5 yr	10 yr	5 yr	10 yr	10 yr
Equity Growth Rate	12.75 %		12.75 %		12.75 %		
Future EPS Growth Rate	10.30 %		11.82 %		11.82 %		%
Default PE	20.60		23.64		23.64		
Future PE	8.19		15.04		15.04		
Investment Recovery Time (yrs)[1]	8.66		8.25		8.25		
Future EPS	3.02	4.93	3.23	5.65	3.23	5.65	
Future Market Price	24.74	40.38	48.65	85.06	48.65	85.06	
Value Price[2]	12.30	9.98	24.19	21.02	24.19	21.02	
MOS Price[3]	6.15	4.99	12.09	10.51	12.09	10.51	
Current Market Price:	26.52 -0.14 ▼ 0.52% Quotes delayed. Price provided as of 2012-11-16 16:00						Calculate

Read about Value and MOS Price computation algorithms.

Figure 3: Safety Price (MOS Price) and Payback Time (Investment Recovery Time) for Microsoft from [Stock2Own, 2012]

There it is visible that Microsoft might not be a buy opportunity at that time, because the current market price is higher than what a value investor would want to pay for it according to the safety price (or MOS Price). This does not mean that it is generally a bad idea to buy the stock, since it might still be a valid option for a dividend play, with its stable payouts, or it might be interesting for other portfolio management strategies. But at least for value investing, the investor can be sure to stay away here. For other cases where it is not that obvious, investors might be interested to try their own calculations with their own modifications and biases and compare those to calculations from other sources to get a clear picture. This is because even those ready made calculations might sometimes be a bit too optimistic in the opinion of the investor.

2.4.3 Changing the Process

Even if there are websites that already do most of these calculations for an investor, it is still a tedious process to select companies and then check all of the above criteria, just to find out that a company is terrific, but not yet in the right price spectrum (if it ever will be).

So to make this approach a bit more practicable and less time consuming, it would be helpful to have a database of all currently available companies and their stocks, with all company reports and daily stock prices. Then have an algorithm that automatically calculates the margin of safety values regularly. Thus later being able to use a screener (like a database query on the calculations) to filter all the companies that are currently undervalued. When the investor sees that list of undervalued companies, he might be able to identify companies that have a meaning for him, or which he might be interested in to do some more research for the moat and management criteria of this strategy. Thus this would reverse the order of the four steps of analysis, which were introduced by Town, to save time in using them.

Since the moat is partly analyzed by looking at the growth rates and stability of the before mentioned five numbers (ROIC, Sales, EPS, BVPS, FCF), it might also be interesting to include this as well in the screening filter as a criterion to only look at stable companies.

Another thing that might be interesting to do is, to calculate a weighted average of the growth rates (with more recent values being weighted higher). This rates companies lower that have had very good performance in the early years (of the ten years looked at), but have had really bad performance in the last few years, which explains why they might be undervalued at the moment.

2.4.4 Automating

At the time of this writing, there is no publicly available website that offers such a specific screener. But the idea has been tested by the author of this thesis before work on the thesis began. There it was possible to validate the idea of changing the process. Thus a database is available with all the calculations, even done historically (with the limited data available) to potentially support backtesting of the value investing approach. The database collects data from public websites regularly, normalizes and enhances that data and then creates value investing ratings that can be used to run the screener on.

The author also put some thought into how to use this screener in an automated strategy and identified the following problems:

1. When this database screener was used, the available ∼14,000 companies were filtered to about ∼30 undervalued ones. After execution of the manual steps of the analysis, that list was broken down again to about ∼5 potential investments. The most interesting company to throw out of the list was Advanced Battery Technologies (Symbol: ABAT), a Chinese battery and battery powered motorcycle producing company with very good historical growth and a good looking balance sheet. The company went down from a ∼4$ to a ∼1$ stock price in a short time. This was because a group of anonymous investors, that made a profit from the decline in price, accused the company of fraud in May 2011 [Prescience, 2011]. They claimed that the company reported wrong numbers in their reports and the owners tried to extract money from the investors. Even though these allegations have not been proven yet and the company can still be traded, the price went even further down to about ∼0.25$ in December 2012. The company still seems very undervalued from a numbers perspective, but the fundamental analysis might lead to a conclusion that an investment would only be possible, if the allegations can be proven wrong in an American court process, which is currently active. Something like this shows how important it is to do research beyond looking at the hard numbers to determine a valid investment.

2. An automated strategy would ideally make its own decision about what to buy or sell. Though since not all the criteria of this investing approach can be easily automated (as seen in the example above), it might only be possible to use this semi-automated in an automated strategy. The semi-automation would require the investor to regularly pick the undervalued companies that he checked himself and give that as allowed investment candidates to an automated trading strategy. With this, the two disciplines of strategies would be decoupled and the investor might reduce his risk by his own manual labor.

3. An automated strategy needs to be backtested well before attempting to run it on real money in a live environment. The value investing approach requires much data (ten years into the past) and publicly available data only reaches ten years into the past. To do real backtesting, one would want to start the backtest from a few years in the past and look how the strategy performs until now. In order to do that, the database would need to include more than ten years of historical data, which would be expensive to license from data providers. Or the strategy would need to be limited to only look at the last three years to make its decisions for the backtest. Also there are problems with historical reports that have been corrected after some time by updates. The original reports would be needed to create a realistic backtest[12], which would decide in the same manner as a live-run would have done. Since these criteria are hard to fulfill in a limited resources research environment, tradeoffs need to be made, which reduce the significance of the results.

4. When doing the backtests in a semi-automated manner, the investor cannot make decisions in the same way as he would have done in the past, since he will surely already have, or will accidentally find more recent information than allowed. This spoils the investment decision for that time in the backtest. Thus a significant rule of backtesting would be broken and the backtest itself would not have any useful meaning.

[12]More specifically, point in time fundamental data is required.

Even with these problems in mind, it would still be very interesting to create an automated strategy that can be backtested to see if the concept behind value investing can be automated to a high degree with an expected positive payoff. The tradeoffs to be made in the model for the backtest would be:

1. It trades companies that might have been a bad idea to invest in, with the current knowledge of the investor.
2. It automatically picks investments only by the numbers, discarding the subjective steps.
3. It backtests with only three years of looking back, starting from seven years in the past.
4. It uses only updated end-of-year data without looking at the original statements. It also discards quarterly statements because they are only available for the most recent year.

With this, it might be possible to get an idea about whether there is potential behind the automation. Being able to do backtests for the strategy would also increase the rate in which feedback can be received for improvements being made on the automation. Thus backtesting becomes essential to automate this in order to decrease the development time by a large magnitude. A significant result about the real profitability of the automated strategy can only be accomplished over a long-term forward-test, which should follow anyway after a backtest before running the strategy with real money. Even though this would be required, this is a very large investment of time, thus the backtests become more important to provide a significant outlook.

Another problem with backtesting the portfolio management strategy is the fact that it can only be automated when coupling it with an automated trading strategy. On the other hand, an automated trading strategy can work without a portfolio management strategy by simply running it only on a single market. The next chapter will thus talk about automated trading strategies in a separated point of view.

3 Automated Trading

This chapter discusses who is using automated trading, how automated trading strategies are generally developed and what they consist of.

3.1 Types of Traders with Automated Tools

Roundabout there are three basic types of traders regarding automated tools:

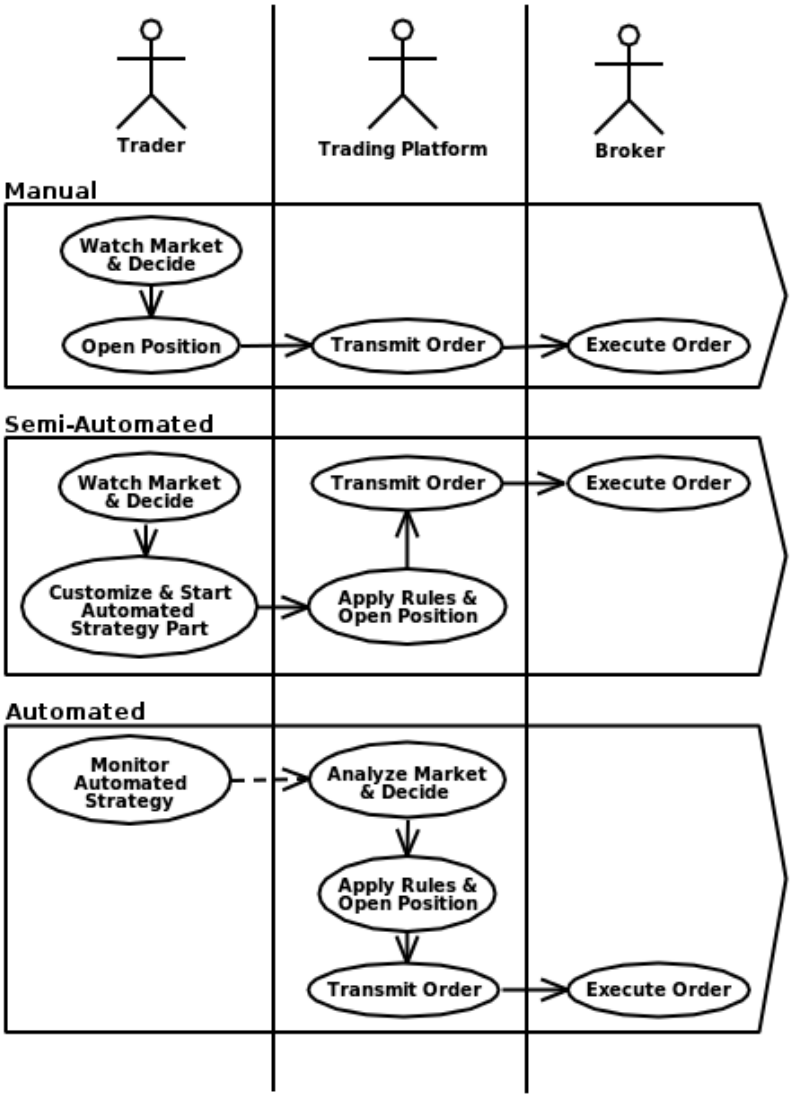


Figure 4: Overview of Activities for Opening a Position among Types of Traders

1. **Manual:** *I do everything on my own.*
   These traders use trading platforms as their only form of automation. They may use automated indicators[13] to help them analyze the market, but the decision when to buy or sell is their own. When this trader does not sit in front of the chart, his only means of automation are the take profit and stop loss levels he manually sets on his orders.

2. **Semi-Automated:** *I am in control, though I use automation to improve my success.*
   These traders have automated strategy parts which they turn on or off in their platform depending on the market situation. The strategy parts are like an improvement with which they can automate buying at a favorable moment that is hard to catch otherwise. Or the strategy parts automate a complex trailing stop mechanism or anything that involves many calculations but is otherwise straightforward. The strategy parts used by this type of trader would not make profit when run unsupervised, because they do not automate every aspect of the trading. The trader handles specific aspects of his strategy manually, because they cannot be easily automated. This trader does not necessarily have to be in front of the charts the whole time, because he can automate the things he would normally just wait for. Often the strategy parts are individually customized by parameters every time they are used. Even if the trader uses completely automated strategies, but needs to regularly manually adapt the parameters to market conditions, this is still called semi-automated trading.

[13] For an overview of various indicators and signals see [Colby, 2002].

3. **Automated:** *I know my rules and they work on their own.*
   This type of trader does not like doing things manually, or does not want to sit in front of the chart a lot. Instead he focuses on developing strategies that run completely automated. He tries to only pick strategy parts that can be automated. His strategies need to be simple and robust. They have been tested a lot and are statistically validated to perform well. His main work is optimizing his strategies without overfitting them and developing new strategies.

These three types of traders do not represent hard boundaries. They rather represent a type of scale by which strategies can be rated as mostly automated or mostly manual. Since the aim of this thesis is to have a concept for completely automated strategies, the focus will be set on that end of the scale.

Going from manual to automated trading is done with the following optimizations in mind[14]:

- Replace subjective parts of the strategy as much as possible with objective criteria. Thus increase understanding of the strategy and open new ways of automation and improvement.
- Reduce the traders menial labor by letting the computer do it for him. Replace that saved time with creative work to improve the strategy instead of simply executing it.

The most interesting part about the automated trader is what he actually does. In fact he does a lot of analysis and development work, which can be seen as a process. This is called the strategy development process, which will be discussed in the following chapters.

[14] Another optimization that is fetching ahead of this chapter is attempted by this thesis. It consists of also automating the strategy development process by using modular strategies and generating rule combinations from those. This also reduces menial trial-and-error labor during strategy development for the automated trader.

3.2 Strategy Development Process

As stated above, the main work of an automated strategy developer is to build and optimize strategies. While doing this, he follows a strategy development process that ensures that the developed strategies perform well.

3.2.1 The Simple Way

A simple strategy development process, that is observable with strategy developers and is similarly described in [Wright, 1998], looks like this:

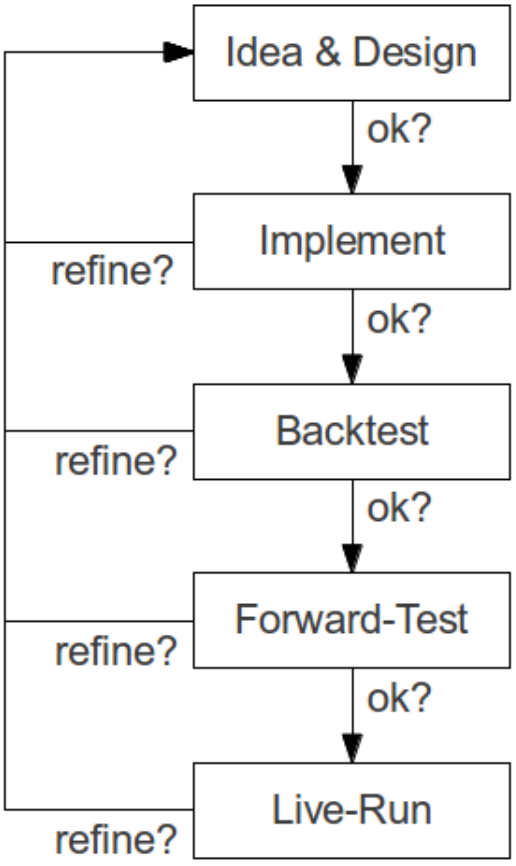


Figure 5: Simple Strategy Development Process

While at every step, the developer also has the option to stop what he is doing by giving up.

This process begins with an idea of a strategy, which the developer tries to formulate as a set of rules. During the formulation as a set of rules, he figures out if the strategy can be automated or not.

The next step is to implement the rules in a trading platform which is capable of testing the strategy. While implementing the strategy, he runs smaller tests and looks at reports to verify that his rules are implemented correctly. He adds output to his strategy execution to provide information to verify calculations and decisions being made during a strategy run. If he identifies anything that is not appropriate to what his idea was, he corrects it with the help of code debugging.

If the strategy works as planned, he tests if the strategy is profitable over a longer period of time and in different types of markets. He runs the strategy in bull (rising), bear (falling) and sideways periods for various instruments on historical data. This is done easily in a backtest over one or more years. While being at this step, he might also try different parameters to improve the performance of the strategy, which always brings him back to the first step. Though this time the change is quickly implemented and backtested again.

If he found out that some backtests were profitable, he decides to test if the strategy also runs properly in a simulated live-run, called a forward-test[15]. Here he ensures that the rules still apply correctly on a live data feed. This is especially important if he decided to improve backtest performance by decreasing the data feed quality by reducing the data points. Since the forward-test runs in real time, it takes too long to verify if the strategy would be as profitable as it was in the backtest. Thus he only checks if it technically runs properly and does some profitable trades.

If he saw some profitable trades, he will surely be impatient to make some real money with the strategy, so he decides to let the strategy run on real money. After all, he wants to benefit from all the work he has put into the strategy and he already saw that it was profitable in the backtests.

[15]Live-run and forward-test imply the usage of intraday feed data to ensure that the strategy also performs well on high quality data outside of the simulated market of the backtesting engine. Even though value investing works long term, the trading strategy coupled with it might use short term rules in order to determine the best trade entry or exit. There are also other reasons to do this, which are out-of-scope of this thesis. The reader is encouraged to investigate this on his own. A starting point could be [Pardo, 2008].

3.2.2 The Problems with Simplicity

After some time, while the strategy runs with his hard earned money, he will notice that the strategy loses money. He might respond to this by going back to the drawing board and changing his strategy, or by keeping his trust in the strategy and letting it run some time longer. Anyway after some more time, he will notice that the strategy somehow is still unprofitable or not as profitable as it was in the backtest. In the worst case, the strategy has already lost half of his money and he decides to give up on the tedious work of finding something that makes this strategy profitable. He either gives up this work completely, or tries his luck again with a different strategy.

The above described scenario is in fact common in the automated trading community, since highly profitable automated systems traders are still rare as of today. The problem those unprofitable traders must face is, that they might have fallen victim to one of the many pitfalls in automated strategy development.

One of the most common pitfalls is, that while they optimize their strategy on the backtested data, they might effectively just be curve-fitting the historical data. They add rules after rules to prevent those loss trades they saw in their historical data, to improve their report results. This makes the backtest look good, but leads to worse results in the live-run, or in sample data outside their normal backtests (called out-of-sample data).

Another pitfall might be that they optimize their parameters only once for their whole backtesting period and expect that the same optimized values will be the best for the future data. When they find out this is not the case, they somehow reoptimize their strategy at random times during their live trading. Like every time they lose trust in their setup. This causes that the statistical results of their backtest do not apply anymore to their live trading, since it is in fact a completely new strategy they are running.

3.2.3 A More Scientific Approach

The previous simple strategy development process shows itself to be very time consuming and not quite effective, neither efficient. Pardo developed a more scientific approach which was introduced in his book titled *"The Evaluation and Optimization of Trading Strategies"* [Pardo, 2008].

There he does not find a solution to the high effort of time while developing strategies. In fact he makes it more time consuming by adding more steps to the process. Though these additional steps address some pitfalls of the simple approach and provide a higher chance of producing robust trading systems in the end.

He tells that a profitable strategy needs to be optimized regularly in order for it to accommodate to the ever changing markets. Also every market should be optimized on its own, because it is unique and the strategy operating on it should be unique as well. Since the markets change, the strategies have to change and evolve with the markets to produce positive results. So he focuses on how to properly optimize strategies and how to avoid the common pitfalls he himself experienced during his career.

Especially, he teaches what types of optimizations should be avoided that lead to curve-fitting and thus decreased profitability. And he shows a way about how to test a strategy for robustness and how to make this measurable. This measure of robustness measures the difference between the profitability of the optimized backtest runs with the profitability of applying those optimized parameters to backtest runs with out-of-sample data.

By incorporating the optimization into the backtest, in the same manner as it would be done in the live-run, these backtest results on out-of-sample data become much more statistically significant as a measure of the future performance of the strategy than a normal statically optimized backtest would be.

This type of testing is called Walk-Forward-Analysis and is incorporated into his process as another step. How Walk-Forward-Analysis works will be discussed later. First a look on the scientific strategy development process itself should be made.

His process looks like this:

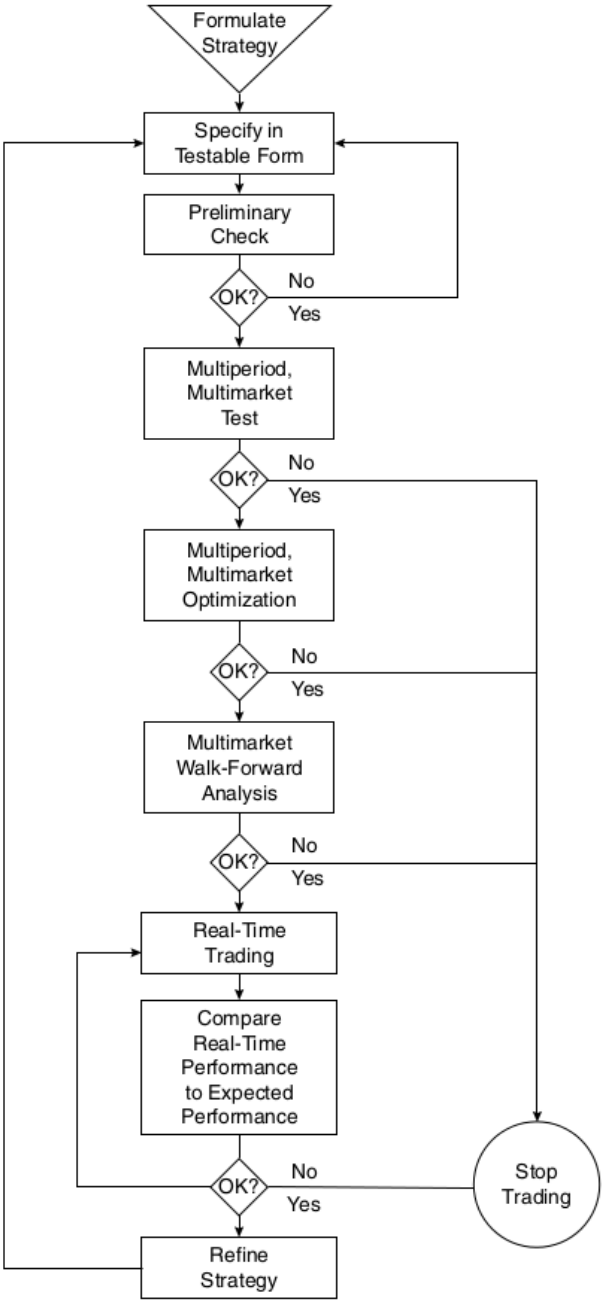


Figure 6: Scientific Strategy Development Process from [Pardo, 2008]

This process is similar to the simple process in the beginning, as the strategy is broken down into rules, implemented and tested to work as expected.

The next step of "Multiperiod, Multimarket Optimization" differs in the way that optimization is done individually for the different periods and markets.

If the strategy is profitable in these optimized tests, he goes further to doing a full Walk-Forward-Analysis on multiple markets.

When that was successful, the real time trading begins, which is not distinguished between forward-test and live-run in this process description.

Also as a difference, the "Refine Strategy" step is modeled as the last step, which might not in fact represent the real development procedure that is lived by the developer. He more likely decides in every step of the process about whether to stop or to refine the strategy again. But this is not so important here after seeing the similarities of the processes, because the interesting part is the Walk-Forward-Analysis, which will be explained now.

3.2.4 Walk-Forward-Analysis

This is what the Walk-Forward-Analysis looks like:

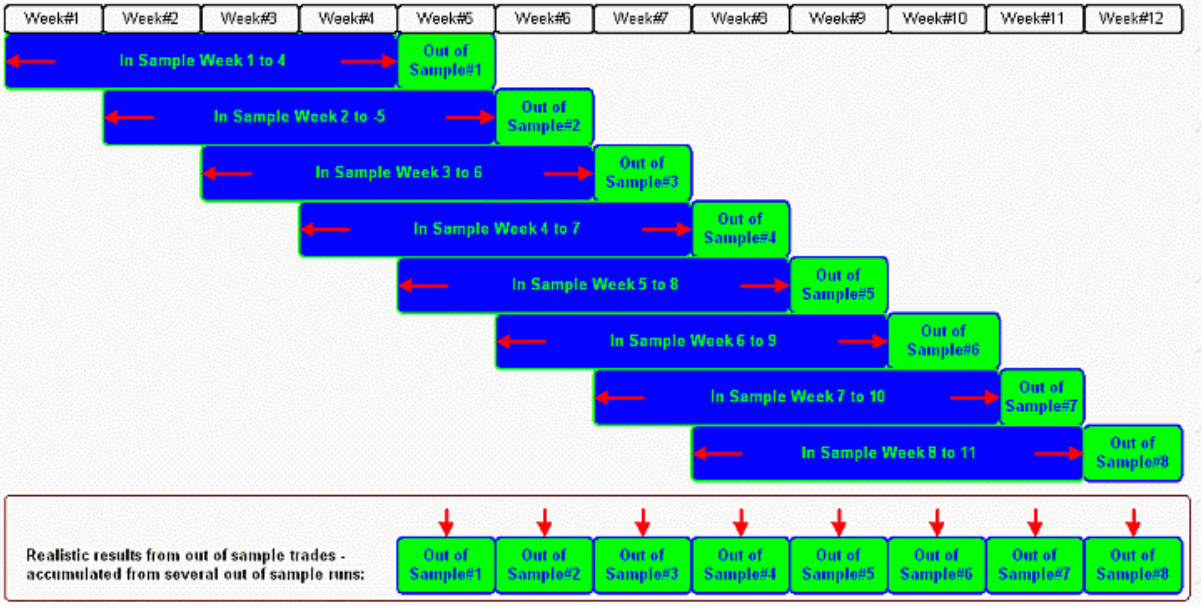


Figure 7: Walk-Forward-Analysis from [Johns, 2011]

It is a special type of backtesting that is composed of multiple smaller backtests on optimization periods. These optimization periods are split over the whole backtesting period and are always followed by out-of-sample tests with the optimized parameters.

After the optimization runs with different parameter combinations, the ones that fit best a selected fitness criterion are chosen automatically (with the possibility to decide manually).

The fitness criterion might be the produced equity profit, profit/loss ratio, Sharpe ratio or some other statistical value of the backtest that measures the success. Even combined criteria can be used here.

The out-of-sample period following the optimization period should be about 25% of the length of the optimization period.

After having done multiple iterations with optimization and out-of-sample runs, while moving forward in time, the realistic profitability can be measured by combining the out-of-sample runs and looking at it as a single backtest. This concatenated out-of-sample backtest shows what a live-run over the historical timeframe would have resulted in.

After this is achieved and the fitness criterion has been calculated for the concatenated out-of-sample backtest, it can be compared to the average fitness criterion result of the optimization runs. This results in the Walk-Forward-Efficiency-Ratio, which is the measure of robustness of the strategy. A value above 0.5 indicates that the strategy should be able to produce profits in a live-run.

Another thing the Walk-Forward-Analysis helps to determine is the period length and frequency of, and between, optimizations during live-runs. This can be evaluated by doing multiple Walk-Forward-Analysis runs with different periods and determining which period fits the strategy or the market best.

3.2.5 Incorporation

The aim of this thesis is to develop a process to find trading strategies for the value investing portfolio management strategy with generative means.

Incorporating the scientific strategy development process and the Walk-Forward-Analysis into this concept provides a huge benefit, because it provides a way to test and analyze the generated trading strategies in a realistic way.

What cannot be used entirely are the steps of development, which will be different. This is because the developer does not implement whole strategies like it is done here, but instead implements only additional decision points variants, which he tests individually and tests in combinations that create the actual strategy for him. So in the concept part of this thesis, a different strategy development process will be presented that fits this approach.

The problem that has to be overcome when applying parts of the scientific strategy development process, is the huge time needed to do the backtests. Since the generated strategies not only need to be tested for parameter combinations, but also for decision points combinations.

These decision points can be modeled as parameters after changing the nature of the trading strategies, which will be discussed in the following chapters. Important here is that the Walk-Forward-Analysis often only tries to optimize about four parameters simultaneously[16]. Every parameter added increases the testing time exponentially. So having decision points as parameters increases the complexity and amount of the testing to a manifold.

Though while thinking about the concept, it still seems possible with todays computers and a few tradeoffs:

1. Trading should operate on daily bars, to reduce the number of ticks to be processed. This is not a problem, since value investing is a very low frequency trading approach that works fine on daily bars. This enables single backtests to run over multiple years in mere seconds.

[16] This is just a rule of thumb that actually also depends on the scan ranges of the specific parameters and what optimization strategy is used to test all possible combinations. Another rule of thumb is to define the scan ranges for quantitative parameters at 5% intervals of the parameter range to be scanned, in order to reduce the values of a given parameter to a maximum of 20. These rules of thumb are recommended by [Pardo, 2008].

2. Decisions about which parameters should be optimized should be made intelligently. The developer has to choose only those decision points that he wants to improve. Doing this in a brute-force-fashion might not be possible. Though it might be possible to develop optimization algorithms that could automate this choosing process.

3. Tests should be able to run on multiple computers simultaneously. Reports need to be written to a single database with metadata about how the strategies were automatically composed. With this information it, is possible to filter which generated strategies performed best to further analyze them. The metadata allows the developer to recreate the generated strategy to do manual testing of it, or to deploy it in a live-run environment.

The first tradeoff is something that needs no further discussion, as the second chapter already addressed this.

The second tradeoff will be discussed shortly in the concept chapter of this thesis. Though the discussion will only contain a few hints on where to start research in that direction, since addressing this issue can only be done after the strategy generation via the brute-force-approach has been implemented, in order to optimize that with the research.

The third tradeoff is the most interesting part that is required to be solved to generate the strategies. It divides itself into two subjects. The first is the parallel processing of the backtests, which can be solved easily by todays technologies and will thus again be mentioned in the concept chapter. The second subject is the metadata, which tells what should be tested in parallel. This metadata tells what building blocks a strategy is composed of and what combinations can be created of them.

Since the metadata has been identified as the most interesting part of knowledge, this will be analyzed in the next chapter.

3.3 Strategy Building Blocks

In order to structure the metadata, strategies need to be dissected into the parts they consist of. Besides the scientific strategy development process, Pardo also writes about those components [Pardo, 2008] and gives some hints about how they can be categorized[17]:

- **Entry and Exit:**
  This defines on what price level and at what time the strategy opens and closes trades. The entry and exit rules can be based on entirely different methods. He says these are the *drivers* or the *engine* of the strategy. He also notes that they can be filtered in many ways. So this major strategy part can be divided into the following subparts:
  - **Entry:** *Where/When to open a trade?*
    Entries can occur when a price is reached in the market, when a trade signal occurs, at specific times of the day, when a chart pattern is recognized, or anything else that can be though of. The potential ways in which this can be done is endless. The entry has also to decide in which direction to enter. Either long or short depending on the method applied.
  - **Exit:** *Where/When to close a trade?*
    Here the same variety applies as with the entries. Though the exit can be applied by a different method unrelated to the entries.
  - **Entry Filters:** *Where/When not to open a trade?*
    The filters are rules that prevent entries from happening, even though the entry strategy gave a signal to do so. Filter examples are a maximum amount of open trades, limiting the times at which entries are allowed, checking that the market volatility is not too high in order not to impose too much risk, and many more.

[17] A development platform that already allows reuse of strategy building blocks on a coarse granularity is [TradingBlox, 2013]. It incorporates Walk-Forward-Analysis and allows optimization of strategies. Thus it is a good example of a platform that implements the concepts presented by Pardo. It has to be noted here that the concept presented later in this thesis not only builds upon this same knowledge, but also goes a few steps beyond what is available to innovate in strategy development concepts.

Figure 8: Example of an Entry Signal that is Based on Opening Range Breakout with Long (Red) and Short (Blue) Price Levels

- **Risk Management:**
  It is essential to risk money in order to gain profit in trading. The aim of risk management is to use the least amount of risk to achieve the maximum profit in a strategy. This is divided into the following subparts:
  - **Trade Risk:** *Reduce the loss on each individual market position.*
    An example here is the overnight risk, which occurs on positions that are held over multiple days. While trading is suspended overnight, trades accumulate on the market, which might cause large price movements on the open price of the next day. These might increase losses significantly without the strategy being able to react properly as it is possible intraday. Strategies might decide to close positions overnight to prevent this form of risk.
  - **Strategy Risk:** *The amount of capital risked for a strategy in order that it realizes its return potential.*
    Losses will definitely occur while a trading strategy operates, though with proper risk management, these losses can be limited. There are many forms of risk that can be managed, though one important one is that the strategy is monitored for losses beyond those seen in the testing phase. If that is the case, the trader should be alerted that the strategy does not work as designed anymore. On the other side, risk management should keep the losses at the expected levels.
  - **Portfolio Risk:** *The amount of capital risked for one or more strategies on multiple timeframes or markets to realize the return potential.*
    When trading with multiple strategies on multiple markets or timeframes, the strategies have to be managed in order that they do not impose too much risk at the same time. If that is the case, the portfolio might be wiped out by a failure of too many strategy instances at the same time and trading has thus to be stopped. To prevent this from happening, the portfolio should consist of instances that operate on completely different strategy concepts and uncorrelated markets. This ensures a strategy diversification that increases the portfolios stability and thus decreases the likelihood of portfolio failure.

- **Profit Management:**
  This major strategy part is similar to risk management in its complexity, but has a different focus. It is focused on capturing as much profit as possible from the trades while preventing immature exits. This is divided into the following two subparts:

  – **Trailing Stop:** *Move the stop loss in order to secure profit.*
  When an order moves into profit, the order stays dynamic in the regard that it moves its stop loss price level with the profit. This is done without retreat, so that profit is not reduced by lowering the stop loss. The stop loss has to give the price enough room to fluctuate on its move in the profit direction, while not giving too much space in order to get as much profit as possible from the movement. Only when the movement trend reverses, the stop loss should act as a safety guard for the profit.

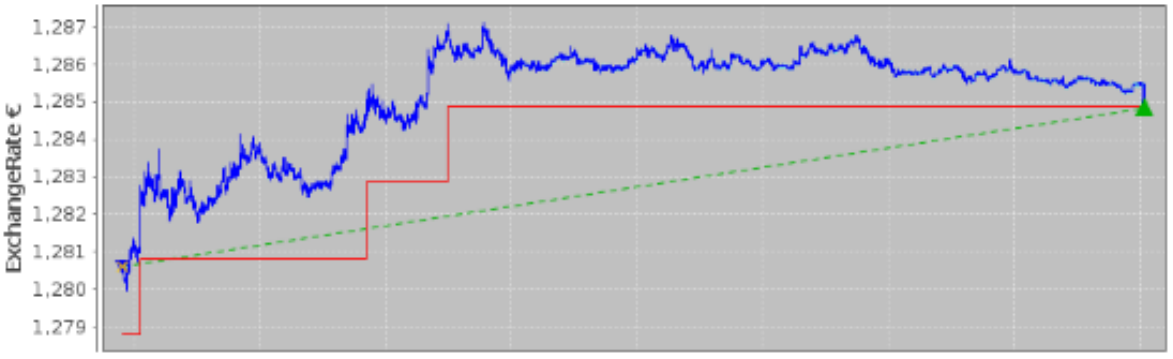


Figure 9: Example of a Trailing Stop (Red) on a Long Position

  – **Profit Target:** *Set a limit on profit to close the order with that amount.*
  Here the order has a specific price level at which it is automatically closed. The profit is not secured on the downside, but ensured to be taken if a desired amount is reached. Strategies that use this method can be less profitable than those using trailing stops, but on the contrary they can have a higher percentage of winning trades. It is also possible to combine both methods in a strategy.

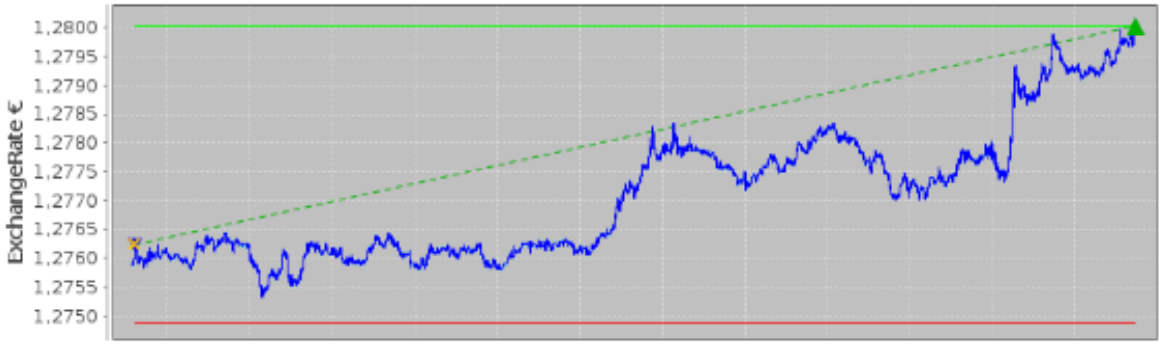


Figure 10: Example of a Profit Target (Green) Combined with a Fixed Stop Loss (Red) on a Long Position

- **Position Sizing:**
  Some traders say that the position sizing is more important than the trading itself. Position sizing can be a fixed amount of money or units of the instrument. Though this might not be the most profitable way of doing position sizing. Instead, position sizing should be adapted to the way a strategy behaves and to changes in the market. The aim of position sizing is that the equity is compounded at the most efficient rate. This strategy part is not divided into subparts, but Pardo has some examples how position sizing could be done:
  - **Volatility Adjusted:** Adjust position size relative to a percent of equity risk measured on the volatility of the instrument. Thus reduce position size on high volatility and increase position size on low volatility.
  - **(Anti-)Martingale:** Increase position size after each loss, start at one unit after each win. The reverse for Anti-Martingale is: Increase the position size after each win, start at one unit after each loss.
  - **Kelly Method:** Calculate the maximum equity risk percent by measurement of the win-loss-ratio and the average profit and average loss. Then calculate the position size that matches that risk in equity percent.
  - **Scaling In and Out:** Increase position size as the trade further moves into the profitable direction until a threshold is reached. Then decrease position size accordingly on further gains in the step size the increase has been done.

3.4 Variability of Strategies

Now that there is a schema with which strategy building blocks can be categorized, it is important to get an understanding about how they depend on one another and what possibilities they provide. As laid out in the previous chapter, strategies consist of several rules that implement specific segments of a trading strategy. These rules can be combined with one another, might have dependencies on one another or will not work when applied together.

When looking at it like this, it seems a concrete strategy consists of a selection of rules inside these constraints. So these constraints need to be understood well to do the rule selection properly. To help with this, it might help to visualize the possible combinations in some sort of model.

Luckily there already exists a model for this type of visualization, which is called a feature model. A feature model visualizes variability, while variability is a concept of decisions that can be made when creating a product from a blueprint. The blueprint has options that can be enabled or disabled and it might have different alternatives with similar functions. For example, when buying a car, the blueprint might allow the customer to choose among different engines with varying performance characteristics, or he might be able choose if he wants a smoker set or not. These are the variants that were chosen for the variability model that the blueprint of the car offers. When he has made his decisions, the car goes into production and the variants get put together to form a complete car as desired. This is the binding of the variability according to the constraints of the feature model of the car, which is based on the blueprint.

When comparing this to automated trading strategies, the same decisions have to be made on a blueprint of possible trading rules, in different variability areas for decision points. The trader might choose a moving average crossover signal entry coupled with a fixed stop loss and a fixed profit target on trades with a fixed amount of units traded. This is a simple strategy that only uses one variant for each decision point. To make this example more complex, the trader could choose to not only use the moving average crossover, but also allow trade entries based on an opening range breakout. This shows that it is possible to bind two variants to the same decision point of entries. Since now there are two signals, they might occur at the same time, so the trader has to decide which rule should be applied on how to combine these signals. Thus there are dependencies among rules. When deciding about the behavior of the signals, he has to consider that a change to the position sizing rule might be needed. But here he does not know which of the possible variants are the best and thus decides to test some variants to make a reasonable decision. Though this becomes time consuming as he figures out that multiple rules seem to work and work not depending on the parameters applied to them. This shows that variants can also have parameters as inner decision points. This also shows that some variants might lead to more decisions to be made with more constraints that have to be kept in mind.

Feature models are mighty enough to model the possible decisions in some sort of tree like structure. See figure 11 for an overview of the notation of a feature model:

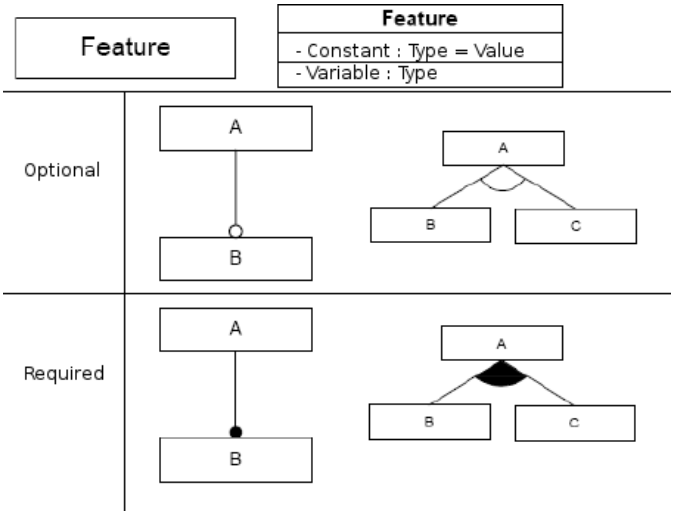


Figure 11: Extract of the Feature Model Legend based on [Kang et al., 1990]

This is how a simplified feature model of that strategy might look like during the development and testing phase[18]:

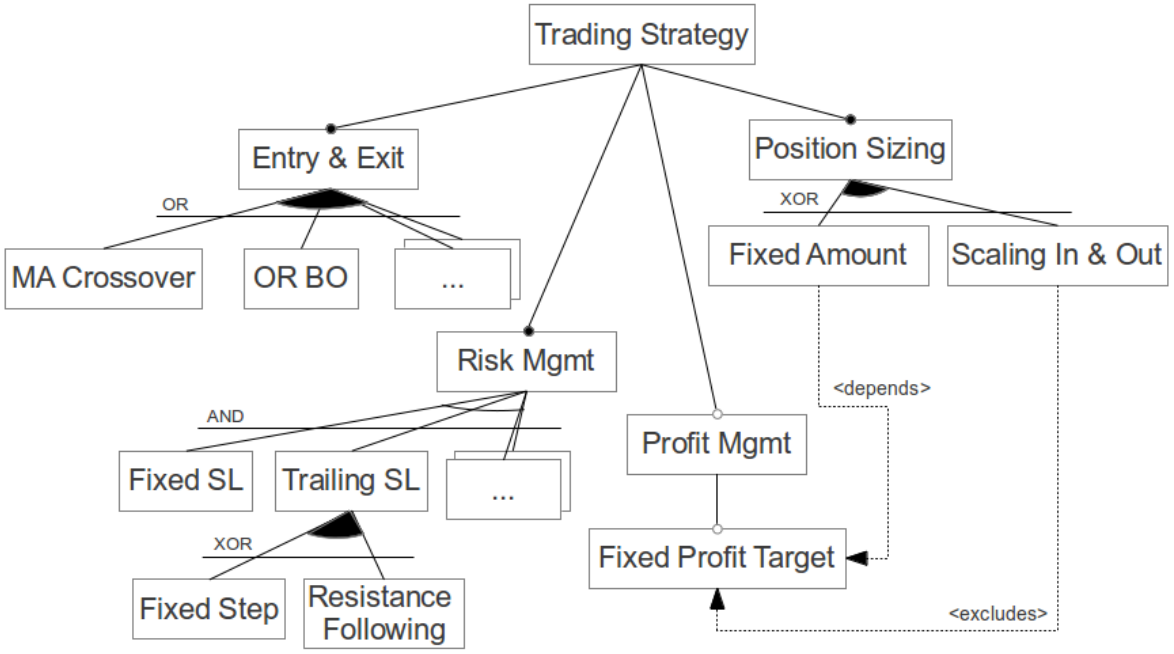


Figure 12: Simple Strategy Feature Model

With this approach, the trader adds more variants to the strategy feature model during implementation. They can be selected by configuration parameters. These configuration parameters are then tested in various combinations. According to the results, the trader might try to add more variants or might want to remove unsuccessful variants until he finds a combination of rules that seem to work well together.

When he has found the rules that his strategy consists of, he can proceed with optimizing the parameters of the rules to increase the potential profit. This is where the Walk-Forward-Analysis might be applied, to also check whether the selected combination of rules is robust enough for a live-run.

[18] Trailing stop loss has been especially put under risk management instead of profit management, since Pardo himself does not distinguish thoroughly between these two aspects. This should highlight that it is up to the trader in the end, about how he categorizes his strategy rules according to his own argumentation. The programming model presented later in the concept chapter will also distinguish itself from Pardo in many ways.

3.5 Metadata Summary

To sum up the last two subchapters, it can be noted that with the overview of components in a strategy, the trader knows what aspects he has to think of when designing a strategy. In the metadata of the strategy, these aspects can be used to distinguish different classes of rules and to put them into a hierarchy.

With the feature model on the other hand, the trader can get an overview of what constraints his rules infer on one another in the developed strategy and which configuration combinations are possible. The selection of the features not only determines the rules that are used during testing, but also which parameters can be optimized during further testing of the strategy.

The idea behind looking at these two points of view here, in regard to the initial goal of this thesis, unravels itself to be an analysis of what needs to be implemented in the platform to automate the generation of the strategies.

The platform has to find an object oriented programming model for these concepts, to manage rules as separately developed building blocks that can be reused among strategies. And it has to have a grasp of the constraints that these rules infer on one another to generate combinations of rules in order to automatically test those.

The trader then does not have to manually configure each backtest during the strategy design phase, but instead can just implement the variability he wants. Then the platform can determine the best combination of rules and measure the performance and robustness of them with the Walk-Forward-Analysis. Then after the tests are finished, the trader can just look at the results and see which rules seem to have performed best in order to enhance the strategy by objective decisions.

This not only saves the trader precious time of previously manual labor, but also increases his chances of finding profitable strategies by automatically being thorough in the process.

With this knowledge, the next chapter goes into more detail by making architectural decisions and showing design patterns with which the platform can be developed to achieve this automation of the strategy development process.

4 Platform Concept

The previous chapters described the areas that needed to be addressed in order for the platform to fulfill its purpose. In the following, these chapters will be wrapped up and composed into a new concept by going the inside out direction. Thus a programming model for modular strategies, then a highly automated strategy development process and after that an incorporation of the value investing will be addressed in that order.

4.1 Modular Strategies

The strategy building blocks were shown in the previous chapter to be a complete model of components a strategy needs to consist of. The variability of the strategies also leads to the conclusion that it is possible to come up with a variable strategy that is in fact composed of multiple possible strategies which can be selected by parameterization before start. Together, these two knowledge pieces form the basis with which a modular strategy can be developed as a programming model. This resulting programming model will be described here[19].

4.1.1 Basis Strategy API

The first thing needed for a modular strategy is a strategy application programming interface (API) it can be based on. This strategy API is developed in a way so that is stays market neutral. This has been given as a secondary goal in Chapter 2.3. The reason for this is that it would not be very effective to develop this from scratch when there are already strategy APIs available that can be adapted for this purpose. So the author chose the JForex platform from the Dukascopy Bank, which provides a free software development kit (SDK) [Dukascopy, 2013].

[19] This is based on the Java programming language. A reference book is provided by [Cadenhead, 2012].

This SDK has been chosen because the forex market has the most widespread tooling for strategy APIs, with many trading platforms to choose from that are free to use and develop on. The stock market requires subscription fees to be paid for access to the APIs, which often do not even provide support for automated backtests[20]. Instead they require a trader to buy other tools or develop their own backtesting engine to succeed with that. Thus instead of trying to develop the wheel from scratch, the modular API is first developed with the help of the established JForex SDK. Then later it becomes easier to develop a stocks backtesting engine, when being able to test and compare with backtest results from a working platform. Sadly this detour over the JForex API is enforced by the circumstances, but at least this will not prevent the goal of this thesis from becoming a reality. Instead, this detour enables it to become reality.

The main parts of this API are shown in the following class[21] diagram[22]:

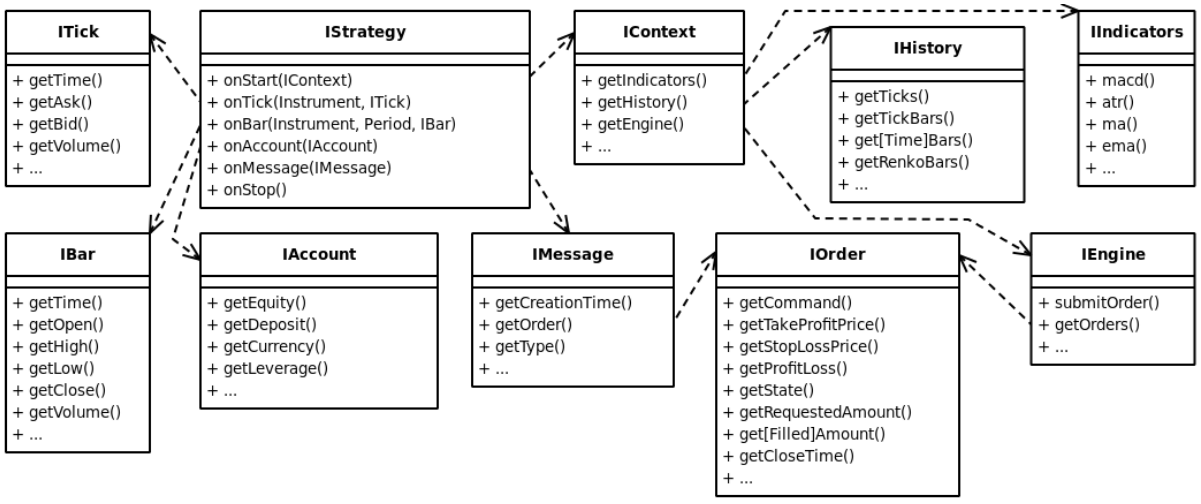


Figure 13: JForex Strategy API Informal Class Diagram Excerpt

[20] The stock broker in mind is Interactive Brokers [Brokers, 2013], as the most accessible and established one of the automated stock market brokers. Even if they are relatively cheap, it is still unviable to invest 10$ per month and put 10.000$ as the required minimum deposit into the broker account for an unfinanced research project. This only becomes an option as soon as the research is far enough to make profits that return that investment. Until then it looks like this is just paying money for years without knowing if it ever will be returned, which should be avoided and can be avoided.

[21] Instrument is just an enumeration for currency pairs like EURUSD and GBPUSD which define forex markets. Thus it is not listed as a class itself.

[22] An introduction to UML as the basis for class diagrams is provided by [Fowler, 2003].

The following classes provide the core functionality:

- **IStrategy:** This interface is the most important one. The strategy developer writes an implementation for his strategy using it. The developer has access to every information and all events he needs to implement his rules with. The entry/exit rules are normally placed in the onTick/onBar method, which gets called for every new tick/bar on subscribed instruments.
- **IHistory:** This class provides historical data for different types of bars in various configurations and the orders history.
- **IIndicators:** This class provides common indicators like moving averages.
- **IAccount:** This class provides information about the account usage to calculate risk management rules.
- **IEngine:** This class allows to submit orders and to get a list of active orders.
- **IMessage:** This event class gives information about changes to orders. Like accepted stop loss changes or an order close.

A good point about this API is that it provides every piece of information needed to develop strategies. Though writing variable strategies is very hard here, because this is on a very low abstraction level. Things like conversions from instrument prices to trade amount, or account money is something that has to be done with a specific conversion formula everywhere it is needed. The API also only uses static parameters for methods, instead of using fluent APIs which would make the code easier to understand and could prevent illegal parameter combinations where appropriate. This would be especially useful on the submitOrder()-Method to prevent common mistakes when opening positions. Also, code reusability is completely up to the strategy developer. There is also only a concept for numbers based indicators without a concept for reusable signals. The historical ticks/bars are complicated to fetch and store inside the strategy without decreasing code maintainability and causing a major decrease in the strategy runtime performance. Last but not least, it is hard to write strategies that operate on multiple instruments, because they are not differentiated very well in the API. Thus it is impossible to run one strategy with differently optimized parameter sets for different markets without duplicating the strategy itself. Strategies developed on top of this API will most likely result in a maintenance burden without applying design principles to correct these issues.

Since it is a hard task to write modular strategies with this API, it has been decided to only use this API as a leading example for the design of a new, more developer friendly API. This API is developed with market neutrality in mind so that it can later be used in the same way for the stock market. This API works on a higher abstraction level and thus makes the above issues easier for the developer or solves them completely by adding a layer of transparency where required. Like a cache mechanism for historical data to improve performance. Another example is domain specific classes for amounts, money and prices that automatically convert values correctly among one another. The JForex API is thus only considered as a backend for the basis strategy API on top of which the modular strategy API is built. Another backend for a stock broker API with a self written backtesting engine will also be implemented in the future to realize the goal of this thesis. The currently developed state, that is presented further on, nonetheless suffices as a proof of concept for this chapter.

The basis API that has been developed on top of the JForex API is shown in the following class diagram[23]:

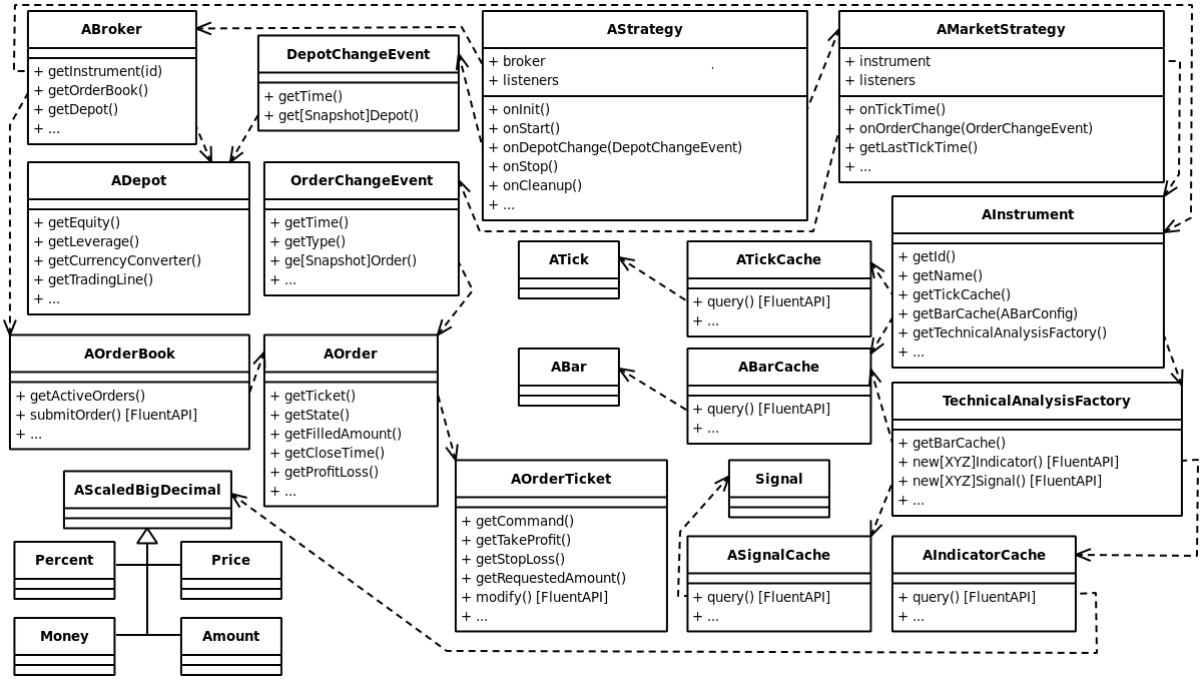


Figure 14: Basis Strategy API Informal Class Diagram Excerpt

[23] This time instrument is a class on its own, since it now stands for stocks, currency pairs and other market instruments in a reusable form.

The information available stays the same as it has been in the JForex API, though it is restructured a bit and improved in some ways. The main differences in addition to the ones stated earlier are the following:

- **AOrderTicket:** Splitting the order ticket from the order allows a fluent API[24] to be developed for easier order creation. When a ticket is submitted, it becomes an order. Ticket builders can also be reused as templates in the strategy code.
- **Listeners:** Being able to add listeners to the strategy for any kind of events and strategy behavior analysis allows easy collection of statistics without needing to implement this directly in the strategy. This makes it possible to simply add a DocumentReportListener (which internally uses a ChartReportListener and a StatisticsListener) to a backtesting run to generate a PDF after the test is finished. These reports can be used as a helping tool for debugging rules or strategies and later to reverse engineer the generated strategies. The statistics from the StatisticsListener can also be used as fitness criteria for the backtests in a Walk-Forward-Analysis. Listeners can also be used in rules to access all the relevant data needed without breaking the design of the strategy.

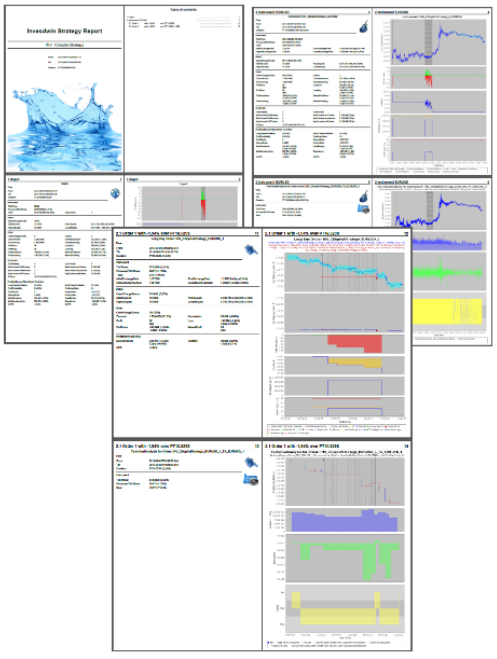


Figure 15: Overview of a Document Report

[24] The benefits of fluent APIs and how they work are explained in the paper [Singh et al., 2011].

- **TechnicalAnalysisFactory:** Strategy rules can also be implemented as custom indicator or signal caches. These can be reused among strategies. They provide access to previous calculations in a cached manner to reduce the calculations necessary in the strategy during data point iteration and thus improve runtime performance. Complex rules can be implemented by building a composite indicator/signal cache that combines various other indicator/signal caches in a specific way that is suited for the strategy. A TechnicalAnalysisFactory is instantiated for a specific BarCache. This makes it easy to instantiate indicators/signals for various types of bars (like different time periods or price aggregations) and combining those.
- **AStrategy:** The strategy has been degraded to a place solely designed for implementing portfolio risk management and to manage individual market strategies. The usual strategy lifecycle is still implemented here as well.
- **AMarketStrategy:** This is the most important change regarding the underlying JForex API. Most of the strategy rules are implemented here. Market strategies make it possible to reuse and customize rules among markets. During a Walk-Forward-Analysis, the strategy can do backtests for individual market strategies to regularly find optimized parameter sets for them and to apply those directly in a live-run or a Walk-Forward-Analysis backtest. Thus the actual implementation of the Walk-Forward-Analysis will be done in an implementation of AStrategy and the strategies to be optimized are implementations of AMarketStrategy[25]. The modular strategy API, which will be explained in the following, is itself also just a special implementation of AMarketStrategy.

[25] This is similar to the approach seen in [Systems, 1997], where sub-strategies are wrapped in each other to do parameter optimizations by running backtests multiple times.

4.1.2 Modular Strategy API

The modular strategy API that is built on top of the basis strategy API by implementing AMarketStrategy, has the aim to model the strategy building blocks of chapter 3.3. This makes the rules for those building blocks reusable among trading strategies. Also this API should make it easier to plug rules in and out depending on the variability model of the strategy.

The static part of the solution to this problem is the design of the modular strategy API which is shown in the following class diagram:

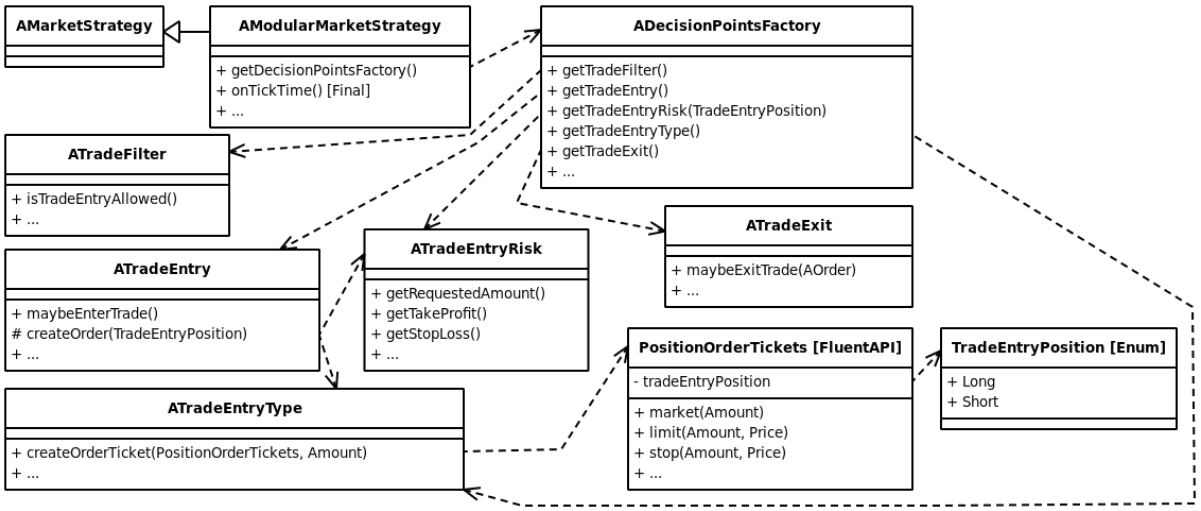


Figure 16: Modular Strategy API Informal Class Diagram Excerpt

The purpose of these classes is explained as follows:

- **ADecisionPointsFactory:** This is the class that actually needs to be implemented by the strategy developer. It consists of the variability model and the binding of the variability to the reusable rule implementations that get added in the specific factory methods. The results of the factory methods get cached and the same instances of the rule combinations are returned by the getter methods during the lifetime of the decision points factory.

- **AModularMarketStrategy:** This class manages the decision points factory and simply executes the strategy logic by invoking the given decision points inside the onTickTime()-Method for the given market data in the following order:

  1. Check the trade filters[26] and invoke the trade entry if they did not deny it.

     a) The trade entry decides if a position should be opened and if it should be a long or short one.

     b) When a position gets opened, the position sizing, profit and risk management settings are determined by calls to the trade entry risk implementation.

     c) Then an order ticket gets created by the trade entry type implementation according to the order command that gets chosen with the help of the fluent API.

     d) This order ticket then gets submitted and executed.

  2. For every open trade that is not still in a pending state, execute the given trade exits[27] to readjust the profit and risk management for open trades, or directly close a trade according to an exit rule.

- **ATradeFilter:** A trade filter checks if some condition is met that should prevent the strategy from opening a new position in the given market. A filter might be to restrict entries to a maximum number of simultaneous positions in the market or the overall strategy portfolio. Other filters could be to restrict trading to a specific time range in the day or to check if market conditions are unfavorable regarding too high/low volatility or the absence of a trend.

- **ATradeEntry:** The trade entry acts on a signal from the technical analysis factory or acts on some other rule to open a trade in a specific direction. An example is the opening range breakout or the moving average crossover which was mentioned in the previous chapter.

[26] Trade filters can be combined with a CompositeTradeFilter, which is the default implementation to which other ATradeFilters can be added. The trade filters will be combined with AND-logic. Thus if one trade filter denies a trade entry, the overall result is a denial.

[27] Trade exits are combined by a CompositeTradeExit by default, which simply delegates the call to the added trade exits in the order of adding them.

- **ATradeEntryType:** This defines what kind of entry is attempted. This decision point is limited by the types of orders which brokers support. Thus this decision point does not provide as much flexibility as the other ones do. This decision has to be made nonetheless.
- **ATradeEntryRisk:** The trade entry risk specifies the initial profit target and stop loss for the order, as well as what position size should be traded. Each of these three strategy components can be seen as their own decision points nested in another decision point. Things like a fixed stop loss or take profit are applied here.
- **ATradeExit:** What applies to the trade entry risk also applies to the trade exit, with the difference that the exit acts once a trade is running. Besides the three strategy components of the trade entry risk, the trade exit component is also applied here as the name suggests. Things like a signal exit or a trailing stop are included here.

As described, the modular strategy API has a place for every strategy building block. The difference between the theory of defining the building blocks and practice of putting them into a programming model lies in the difference between focusing on the categories during definition versus having to focus on the algorithmic nature of the components. The programming model subdivides some of the strategy components among the lifecycle of the orders to properly apply the corresponding rules at the correct time. This is what makes up the difference.

The other part of the solution to the previously described problem, that the modular strategy API is supposed to solve, is of a more dynamic nature. This is in the first case the implementation of the variable strategy itself and in the second case the implementation of a configuration class which is used to control the chosen variants of the variability model. These two parts are both things that have to be implemented manually by the strategy developer himself. Though the modular strategy API not only enables the strategy developer to do this, it also is a framework that tells him how it should look like. Thus this task becomes a simple implementation according to simple rules instead of having to think about the design too much. This additional framework will be explained in the following chapter.

4.1.3 Variability Binding Framework

The variability binding framework first defines a configuration class that models the variability of the strategy. This configuration class consists of the following elements:

- **ADecisionPointsConfig:** This is the base class from which a concrete configuration class is extended. It provides validation for the conventions of configuration classes and adds functionality that processes these conventions to make them usable during the highly automated strategy development process. Configuration classes can be added as member variables of other configuration classes, to group variability constraints for the purpose of reusing these definitions at various points of the variability model.
- **Parameters:** Parameters are defined by annotating member variables with @DesignParameter or @OptimizationParameter. The purpose of the distinction between these annotations will be explained in the following chapter about the highly automated strategy development process. Parameters can be of the following types:
  - **Boolean:** This defines a yes/no option for the strategy, which can be used to enable/disable a specific rule or set of rules.
  - **Number:** This includes types that extend the Java type Number, like BigDecimal, Integer and Double. Settings for signals can be modeled with this for instance.
  - **Enumeration:** Defining enumerations enables the variability model to include semantic alternatives. This might be used to select different algorithms or rules for signals.

  Parameters can be constrained by using one of the following means:
  - **@NotNull:** This is used to define parameters as required. Deciding not to set them before a test will result in a validation error.
  - **@Range:** For numerical parameters, this defines the allowed range in which values are supported. This also defines the scan range during Walk-Forward-Analysis.

  - **validateCustom():** More complex validation rules can be applied by implementing this method and throwing exceptions if a given constraint is not fulfilled.

  Default settings can be provided for parameters by directly initializing the corresponding member variables in the configuration class. Parameters become constants for a strategy by making them final and removing the @DesignParameter/@OptimizationParameter from them, or by just removing them entirely from the configuration class and directly setting them in the strategy code itself.

- **Choices:** Groups of parameters that depend on a specific choice that has been made can be coupled to the choice via the following convention:

  1. Define an enumeration class for the variability point with the possible features it decides upon.
  2. Create configuration classes for each feature of the enumeration which contain the parameters for each of them. It is not required to have a configuration class for every feature.
  3. Create a member variable for the enumeration. This will be used to control which configuration class is actually used. The name of this member variable is used as a prefix for the configuration member variables.
  4. Create member variables for the configuration classes by following the naming convention of using the prefix defined by the enumeration, followed by an underscore and then the enumeration constant name that defines the feature for which the configuration class was created.

  During validation of the configuration class, this convention will do sanity checks on itself to warn the developer if he did anything wrong. The choice will be recognized and every configuration member variable that does not correspond to it will be set to null. Thus only the configuration class that has been enabled by the choice will be validated and the developer will not have to fear validation errors on the unused configurations.

This is in fact everything that is needed to transform a feature model, as seen in figure 12, into a variability model representation of code. An excerpt of the Java code looks like the following:

```
1  public class MovingAverageDPConfig extends ADecisionPointsConfig {
2      public final EntrySignalConfig entrySignal = new EntrySignalConfig();
3      public final ExitTypeConfig exitType = new ExitTypeConfig();
4
5      public static class EntrySignalConfig extends ADecisionPointsConfig {
6          @DesignParameter
7          public boolean signalOnlyOnDirectionChange = true;
8          @DesignParameter @NotNull
9          public EntrySignal choice;
10         public MovingAverageConfig choice_MovingAverage = new
               MovingAverageConfig();
11         public MovingAverageCrossoverConfig choice_MovingAverageCrossover = ...
12
13         public static enum EntrySignal {
14             MovingAverage,
15             MovingAverageCrossover,
16             MovingAverageConvergenceDivergence;
17         }
18
19         public static class MovingAverageConfig extends ADecisionPointsConfig {
20             @DesignParameter @NotNull
21             public MovingAverageType movingAverageType = MovingAverageType.SMA;
22             @OptimizationParameter @Range(min = 5, max = 200)
23             public Integer barsCount = 10;
24         }
25
26         public static class MovingAverageCrossoverConfig extends
               ADecisionPointsConfig {
27             @DesignParameter @NotNull
28             public MovingAverageType movingAverageType = MovingAverageType.EMA;
29             @OptimizationParameter @Range(min = 5, max = 50)
30             public Integer fastBarsCount = 10;
31             @OptimizationParameter @Range(min = 20, max = 200)
32             public Integer slowBarsCount = 25;
33
34             @Override public void validateCustom() {
35                 Assertions.assertThat(fastBarsCount).isLessThan(slowBarsCount);
36             }
37         }
38         ...
39     }
40
41     public static class ExitTypeConfig extends ADecisionPointsConfig {
42         @DesignParameter @NotNull
43         public ExitType choice;
44         public FixedTPandSLConfig choice_FixedTPandSL = new FixedTPandSLConfig();
45         public TrailingStopConfig choice_TrailingStop = new TrailingStopConfig();
46        ...
47     }
48 }
```

Source Code 1: Sample Moving Average Decision Points Config Class Excerpt

Instances of this configuration class can be serialized to be put in a database to store the strategy metadata during automated testing. To restore the information for analysis, the instance just needs to be deserialized later. This can be done with the default Java serialization or with a custom marshalling/unmarshalling mechanism.

The binding of the configuration class to the actual strategy rules is done in a straightforward manner:

1. Create the decision points factory class for the market strategy which is supposed to use the variability model defined in the configuration class.
2. The constructor of the decision points factory class should accept the configuration class, create a copy of it and invoke the validation procedure on the copy. Working on the copy here ensures that configuration member variables, that have been set to null during validation, are still available in the original configuration instance which might be used for further backtest runs during a Walk-Forward-Analysis.
3. Implement the strategy building blocks by binding the variability choices and parameters in the decision points factory methods according to the following convention:
    - For every choice enumeration parameter in the configuration class, write a switch/case-statement or cascaded if-statements which make use of the appropriate choice dependent configuration member variables.
    - For every yes/no option parameter, write a simple if-statement or pass the parameter to the rule directly.
    - For every normal enumeration and numerical parameter, pass the value to the rule or instantiate rules according to it.
4. Use this decision points factory class in a market strategy according to the highly automated strategy development process.

A sample of a decision points factory class for the above configuration class is shown in the following Java code listing:

```
1  public class MovingAverageDPFactory extends ADecisionPointsFactory {
2      private final MovingAverageDPConfig config;
3
4      public MovingAverageDPFactory(final AModularMarketStrategy parent, final
           MovingAverageDPConfig config) {
5          super(parent);
6          this.config = (MovingAverageDPConfig) Objects.deepClone(config);
7          this.config.validate();
8      }
9
10     @Override
11     protected ATradeEntry newTradeEntry() {
12         final AHistoricalCache<Signal> signal;
13         if(config.entrySignal.signalOnlyOnDirectionChange){
14             signal = getSignal().getOnDirectionChange();
15         }else{
16             signal = getSignal();
17         }
18         return new SignalTradeEntry(getParent(), signal);
19     }
20
21     private ASignalCache getSignal() {
22         final TechnicalAnalysisFactory taf =
               getParent().getInstrument().getTechnicalAnalysisFactory(OfferSide.ASK,
23                 new TimeBarConfig(TimeBarPeriod.DAILY), AppliedPrice.CLOSE, true);
24         final ASignalCache signal;
25         final EntrySignalConfig choice = config.entrySignal;
26         switch (choice.choice) {
27         case MovingAverage: {
28             final MovingAverageConfig c = choice.choice_MovingAverage;
29             signal = taf.getMovingAverageFactory()
30                 .getMovingAverageSignal(c.movingAverageType, c.barsCount);
31             break;
32         }
33         case MovingAverageCrossover: {
34             final MovingAverageCrossoverConfig c =
                   choice.choice_MovingAverageCrossover;
35             signal = taf.getMovingAverageFactory()
36                 .getMovingAverageCrossoverSignal(c.movingAverageType,
                       c.fastBarsCount, c.slowBarsCount);
37             break;
38         }
39         case MovingAverageConvergenceDivergence:
40             ...
41         }
42         return signal;
43     }
44     ...
45 }
```

Source Code 2: Sample Moving Average Decision Points Factory Class Excerpt

Additionally, there is a code generator that extracts the parameters of the configuration class and provides a class with constants that identify them:

```
1  public final class MovingAverageDPConfigParameters {
2
3      private MovingAverageDPConfigParameters() {}
4
5      /** TypePath: EntrySignalConfig -> EntrySignal */
6      public static final String entrySignal_choice = "entrySignal.choice";
7      /** TypePath: EntrySignalConfig -> MovingAverageConfig -> MovingAverageType */
8      public static final String entrySignal_choice_MovingAverage_movingAverageType
           = "entrySignal.choice_MovingAverage.movingAverageType";
9      /** TypePath: EntrySignalConfig -> MovingAverageConfig -> Integer */
10     public static final String entrySignal_choice_MovingAverage_barsCount =
           "entrySignal.choice_MovingAverage.barsCount";
11     /** TypePath: EntrySignalConfig -> MovingAverageCrossoverConfig ->
           MovingAverageType */
12     public static final String
           entrySignal_choice_MovingAverageCrossover_movingAverageType =
           "entrySignal.choice_MovingAverageCrossover.movingAverageType";
13     /** TypePath: EntrySignalConfig -> MovingAverageCrossoverConfig -> Integer */
14     public static final String
           entrySignal_choice_MovingAverageCrossover_fastBarsCount =
           "entrySignal.choice_MovingAverageCrossover.fastBarsCount";
15     /** TypePath: EntrySignalConfig -> MovingAverageCrossoverConfig -> Integer */
16     public static final String
           entrySignal_choice_MovingAverageCrossover_slowBarsCount =
           "entrySignal.choice_MovingAverageCrossover.slowBarsCount";
17     ...
18
19 }
```

Source Code 3: Sample Moving Average Decision Points Config Parameter Constants Class Excerpt

These constants can later be referenced to select which parameters should get scanned during the highly automated strategy development process. Any unselected testing parameter will need to have a default in the configuration class, or the developer has to set the values to a constant value before running the tests. Otherwise the validations will remind him of the missing information via exceptions. The developer should try to minimize the selected parameters during tests to shrink the required execution time. The previously stated number for the Walk-Forward-Analysis of around four simultaneously selected parameters can be respected using this mechanism. Testing should thus always focus on a specific target aspect of the variability model and not on the whole.

This concludes the variability binding framework. What can be seen from this standpoint is that the actual implementation effort for a given strategy depends on the variability model that is supposed to be used during the highly automated strategy development process. For this not to grow into a maintenance nightmare, the developer should try to keep the variability model as small as possible. This can be done by doing multiple iterations of the highly automated strategy development process while always trying to find the best variant for only one aspect of the strategy at a given time. When that best variant is determined, the variability model should be shrunk and the variant should become a constant for the further iterations of refinement.

Another factor that defines the implementation effort is the rate at which rules can be reused, have to be enhanced or have to be written from scratch to be used for binding the variability model. Though this factor can be mostly neglected in the long run, since the rules are designed to be reused. Thus as multiple strategies have been developed, a vast portfolio of rules should be available that can be experimented with. New strategy ideas can then be implemented by simply writing the decision points configuration and factory class according to the above conventions.

4.1.4 Computer Developed Rules

Despite hand written rules, it might be interesting to note here that it would also be possible to choose strategy building blocks for which the developer wants to let the computer do the work of rule development. This can be done by applying approaches like neural networks[28], machine induction or genetic programming to create small AI fragments that optimize specific decision points. How these approaches are generally used in strategy development can be read in [Ruggiero, 1997]. One approach that applies well on the modular strategy API is the way of training AI rules inside of working strategies that are composed of fixed rules. After one such AI rule has been developed and trained far enough so that it increases the overall profitability and robustness of the strategy, the developer can go a step further and choose the next decision point he wants the computer to try his best on.

[28] One interesting use case for neural networks is pattern recognition for signals as seen in [VisualPatternDesignerProfessional, 1992].

This can be done as often until maybe a strategy is reached where every decision point is controlled by an AI. This might be something interesting to study and see how effective this would be compared to approaches where complete strategies are trained as AIs from scratch.

A platform that automates the approach of training strategy AIs from scratch is [TradingSystemLab, 2013]. It uses a genetic programming engine coupled with sophisticated pre- and postprocessing of data in order to generate strategies for any market. Compared to the concept of this thesis, the concept of that platform goes the other way around and starts with AIs to reach the goal of generated strategies. It also gives the computer much more responsibility and takes away control from the strategy developer. The resulting strategies are also hard to reverse engineer, since the model on which the generated strategies operate is very hard to understand for the strategy developer.

The concept for this thesis comes from the other side at the problem, in order to let the strategy developer stay in control and to enable him to put his own knowledge into action to finding good trading strategies. This control also makes it easier to combine value investing with trading strategies, even though it might also be possible with TradingSystemLab by defining special value investing preprocessed data on which their genetic programming algorithm can operate on [29]. Sadly this could not be tried because there was no access to the platform and it will most likely be unaffordable to acquire for the time being. Also it will be shown in a later chapter that incorporating value investing into the concept that has been developed here is pretty easy and it feels more natural to strategy developers.

TradingSystemLab internally also uses the Walk-Forward-Analysis to train robust AI strategies. This is what their platform has in common with many manually operating strategy developers and with the platform described here. Though the strategy development process in which the Walk-Forward-Analysis is used ends up different among those three approaches. The next chapter describes how the original strategy development process can be adapted and used in an automated approach that is based on the modular strategy API.

[29] Some changes to TradingSystemLab would be needed to facilitate the portfolio selection filters to be possible though, according to them.

4.2 Highly Automated Strategy Development Process

The adapted sequence flow looks like this:

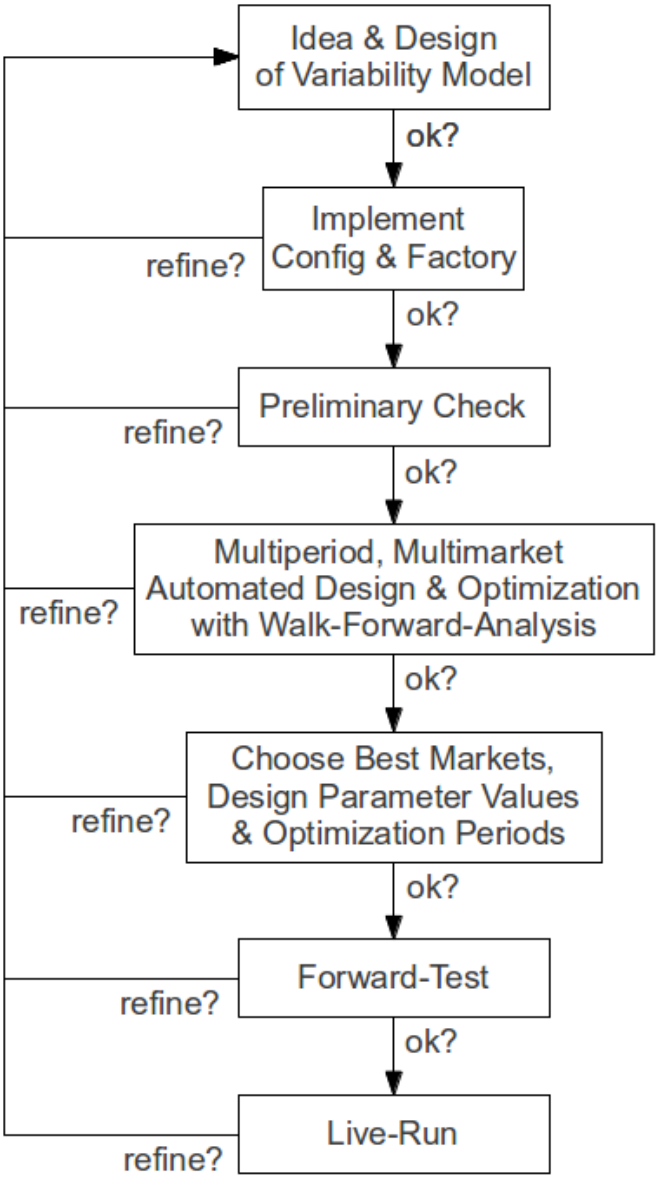


Figure 17: Highly Automated Strategy Development Process

The process starts with the developer designing the strategy variability model and implementing it via the modular strategy API. There he can reuse existing rules. He maps the variability model to the decision points configuration and factory classes he writes. In the configuration class, he defines design and optimization parameters. These include the bar periods which are used to feed indicators and signals. As preliminary checks, he verifies that the implementation works and that the code behaves like he intended it to do.

He then chooses a limited set of design and optimization parameters for the automated testing process step. Every other parameter needs to be defined with a constant value. Multiperiod distinguishes itself from the bar periods here, insofar that it defines the period of optimization cycles which are used by the Walk-Forward-Analysis. Multimarket means that multiple instruments are being tested. The developer defines which and how many instruments these are. When the testing process is started, it will build all possible combinations of the selected design parameter values and start a Walk-Forward-Analysis for these. During the Walk-Forward-Analysis, the optimization parameters will be used for the optimization cycles. The design parameters thus define the possible rule combinations from which individual strategies are derived from, and the optimization parameters define how these individual strategies can be optimized during the Walk-Forward-Analysis. If the developer did not choose any optimization parameters to be tested, the testing process will skip the Walk-Forward-Analysis and instead only do normal backtests for every individual strategy that was derived. As stated in earlier chapters, the developer has to choose which and how many parameters are relevant for the given tests to reduce the execution time. Though the parameters have to be chosen with a given goal of insight in mind, which helps to come closer to a fixed set of parameters and constants that can be used in a final strategy.

When the testing process is finished, the developer can look at the individual results and statistics. These can be used to gain insights and to refine the variability model by reducing or expanding it. When the developer has found a promising strategy combination with a limited set of optimization parameters that perform well, he can go on to test it in a forward-test. If that succeeds and the strategy still shows itself to be robust, the strategy can be used with real money in a live-run. During the forward-test and live-run, the strategy will itself automatically use the optimization parameters to regularly reoptimize itself according to the optimization period that was determined during the testing process.

4.2.1 Finding Possible Strategy Combinations

The possible strategy combinations are generated by parsing the decision points configuration class instance with runtime reflection in Java. For every previously selected design parameter that can be used for the testing process, the possible scan ranges are determined as follow:

- **Enumeration:** Every possible enumeration value is used in the scan range.
- **Number:** The @Range annotation is used to determine the scan range. This scan range is used in 5% value intervals to reduce the maximum number of variants here to 20. The @Range annotation is mandatory for number parameters.
- **Boolean:** True and false are used here.

Additionally for every parameter, the null value is tried as well as a way to express a deactivated parameter.

The possible strategy combinations are generated by going through the scan range of one single parameter, while all other parameters are kept on their initial values. When that parameter scan range is at its maximum value, a second parameter scan range is incremented by one. Then the first parameter is again traversed completely. When the second parameter has reached its maximum scan value, a third one is incremented with the first two parameters starting from scratch again. This happens for multiple parameters until every possible parameter combination is generated.

With this process, many parameter value combinations will be generated that are illegal due to constraints like @NotNull or definitions in the validateCustom() method. These constraints will cause exceptions to be thrown during the validation step, which will cause the invalid parameter value combinations to be removed from the final set of combinations. Thus only valid combinations will be used in the following backtests.

4.2.2 Running Huge Amounts of Backtests

Imagining a sample selection of four design parameters. This selection includes a boolean with two possible values (null excluded by @NotNull), an enumeration with four possible values and two numbers with one having ten and another having twenty possible values. This results in 2 * 4 * 10 * 20 = 1600 possible combinations. When imagining a highly optimized backtest execution time of one second, this would result in ∼26.7 minutes to go through all those tests. Though when thinking a bit more pessimistic because the strategy reports generation would take some time and the backtest might be run with a higher quality price data feed, the execution time could as well range between one minute and ten minutes. This would result in respectively ∼26.7 hours or ∼11.1 days of execution time with a brute force approach. Using a more intelligent approach, it would be possible to reduce the needed backtests to find good parameter values. Though this advantage could be used again to increase the number of selected parameters. Thus the problem of a long execution time persists.

In order to overcome this issue, parallelization comes to the rescue. Individual backtests should be run on a private computing cloud. This private cloud would not be used to handle large amounts of data, but simply to efficiently distribute the backtests to the computers that are free to execute them. For this, Hadoop with its MapReduce framework could be used [White, 2012]. This provides a high level API that makes it easy to build distributed computing applications. In that context, the map task would run a backtest for every parameter value combination of a given testing process and write the results to some database. The reduce task would then analyze the results and aggregate the data in a way where either an algorithm or the developer can choose which combination was the most robust and successful.

Both the strategy design backtests, as well as the optimization backtests during the internal Walk-Forward-Analysis steps could be parallelized individually in such a cluster.

The hardware this runs on can vary from a self managed cluster of Raspberry Pi computers [Southampton, 2013][30] up to renting virtual machines in the Amazon Elastic Compute Cloud [Amazon, 2012]. The minicomputer cluster would result in a cheap purchase with a good ratio of processing power in relation to power consumption. On the other hand, the Amazon alternative would result in low maintenance effort with higher performance, though with a higher price for this convenience.

4.2.3 Finding Good Strategy Combinations

When the backtesting results are available, the developer can look at the individual reports to do reverse engineering to find out which parameter values are the best. Since this is a tedious process, a better alternative to rely on is some sort of automatic rating of the results. This can be done by defining a fitness criterion like it is done in AI programming, in order to define a computed value on which basis combinations can be compared to one another. The fitness criterion can be the overall profit (which is a bad idea, since it does not measure robustness), the Sharpe ratio, the maximum equity drawdown or anything else. Even combinations of those statistics can be used. The aim is to find a fitness criterion that ensures that the given combination has the best chance of being a good candidate for successful live trading. The fitness criterion can also vary between refinement iterations based on the goal for which a limited set of parameters were selected.

It might also be of value to be able to define fitness criteria for parameters individually. Thus each rule could be optimized with its own goal during a testing process run[31].

[30] [Resin, 2013] provides a Java performance benchmark. Even though this minicomputer is not very well suited for memory intensive Java applications, there might be other minicomputers that could be suitable for such a cluster.

[31] See [Wright, 1998] for an example on how fitness criteria match to individual optimization parameters and why this individuality is needed.

Another way to look at the results is to see which parameter value performed best most of the time and what values are common among successful combinations. One way this might gain insight is to see which indicators performed the worst or the best among combinations[32]. The worst indicators could be removed from the variability model and the best indicators could be improved by adding more variability to tweak them, or finding similar alternatives to the good indicators. This knowledge can lead to a good understanding of what works and what does not, in order to write better variability models for further tests or other strategy ideas.

4.2.4 Finding Parameter Constants

When looking at individual parameter values and seeing which values were the most common in successful backtests, it might become obvious that some parameters only work well in a limited range of values out of the complete scan range. Or it is completely obvious, that only one value is superior to all other values. In that case, this gained knowledge can be used to convert a parameter into a constant and thus reduce the variability model. For number ranges, it is ok to choose the median successful value as a constant. For enumerations and boolean parameters, it might be more obvious that one specific value is superior to the other ones. Or the results could as well show that a given parameter has no great impact on the overall result. In that case, the parameter can be removed as well in order to reduce complexity. Doing this over and over until all design parameters have become constants and only a small subset of important optimization parameters are left, a good candidate for a final strategy can be reached. This is the final result of the highly automated strategy development process and the target of the strategy developer.

[32][Colby, 2002] evaluated indicators to find the best performing ones by letting them operate in the same test environment. Experience shows, that the best indicator varies depending on the market, timeframe and so on. Thus the results are of limited use. Instead of relying on that, the concept presented here allows to evaluate the best indicator individually per strategy, market, etc.

4.2.5 Making the Testing Process Faster

Constants finding is a process that can become automated and coupled with a fitness criterion. With one more mechanism, the developer can further automate this strategy development process[33]. One approach that could be used here is a genetic algorithm[34]. In these terms, a chromosome can be defined as a set of values from a DNA composed of parameters. The parameters could consist of all possible parameters in the variability model (which should not be too large). Some parameter value combinations could be used as a starting set of chromosomes. These sets get tested and a few of the most successful ones get selected to create offspring. This is done by doing a crossover on the differing parameter values and additionally doing mutations by changing some parameters to random new values. Then the offspring is tested again to repeat the process. In that process, the parameter reduction limits the possible degrees of freedom for further mutation. This should then lead to a combination that was otherwise reached by manually selecting the parameters to be tested.

This approach reduces the number of tests required and thus the time to find successful strategy combinations at the expense of the developer gaining less insights and thus him maybe working with suboptimal variability models. Though maybe a combination of both variability model refinement and genetic algorithm optimization (or something else[35]) could lead to equivalent results with less time effort. This is subject to further research.

[33] A complete automation of all steps and thus a start to finish automated strategy development process would only be possible if the developer defines some sort of meta variability model that consists of every possible rules and the combinations thereof. Since this variability model would be very large and unmaintainable, this is unlikely to be achievable. The developer still has to work with small variability models and multiple iterations of the development process.

[34] The approach is similar to the one described in [Deschaine and Francone, 2004] or [Bryant, 2010], even though that one describes genetic programming which is a special case of genetic algorithms.

[35] It would also be possible to test each parameter alone one after the other, thus the tests needed would only be the sum of the individual scan range sizes instead of the multiplication of those as stated before. This is suggested by [Wright, 1998], but it only speaks about optimization parameters. When allowing design parameters to be tested as well, which have dependencies on one another, a more thorough testing process might be needed. This is why the approach stated earlier uses the multiplication of the scan range sizes to reflect these dependencies. If it is possible to verify that the dependencies of rules do not represent a problem here, the sum approach might be a valid procedure to reduce the required test runs as well. This would also simplify things a lot.

Nevertheless it can already be estimated from this point of view that it is possible to highly automate the strategy development process using the generative approach based on a variability model as described here. To make this feasible in every day use, methods for reducing the overall execution time need to be further researched and developed, for which some starting thoughts have been presented in the last few pages.

After having described how automated trading rules can be implemented in a modular programming API and how they can be refined into robust trading strategies, using the highly automated strategy development process, the next step shows how the goal of automated value investing can be reached with this.

4.3 Value Investing Incorporation

The following chart illustrates how value investing effectively works:

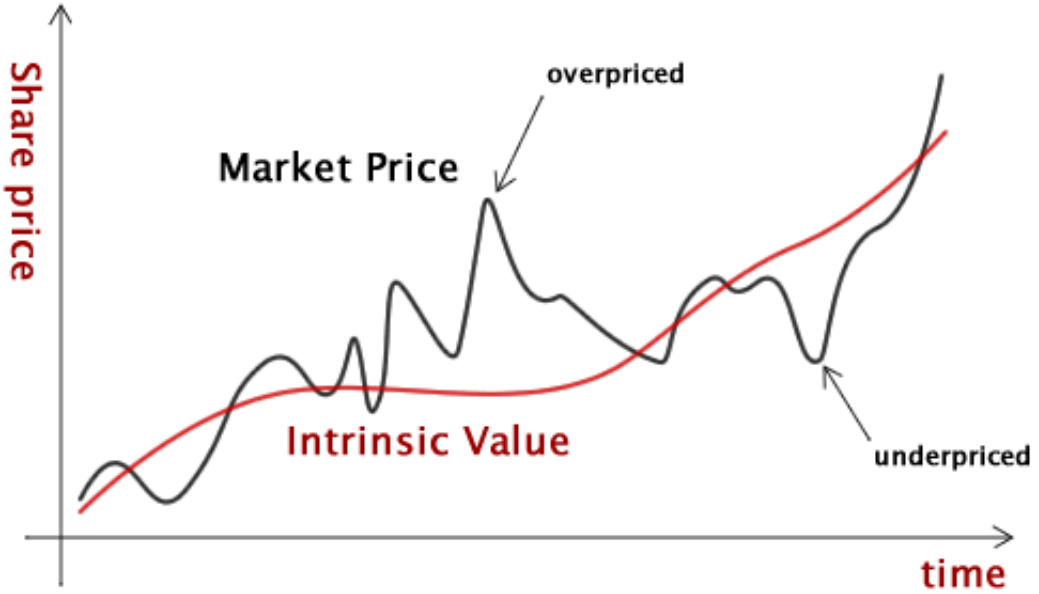


Figure 18: Value Investing Indicator Chart from [Voofie, 2013]

The best time to buy a company is when the market price is above the intrinsic value. The time to sell comes when the intrinsic value is less than the market price. The trick here is to have a way to determine the intrinsic value so it can be compared to the market price. One way this can be calculated has been shown in the first chapter of this thesis. There the sticker price was stated to be an estimation of the value of the company and the safety price (which is a 50% discount off the sticker price) was the price that is acceptable to buy the company at. This was needed as a safety cushion, since the estimation of the sticker price might not be very accurate.

Since the intrinsic value is not the only criterion on which good value investments get determined, it is wise to also find a way to implement some of the filters that were mentioned in the value investing chapters as well.

The next few pages will show how these can be incorporated as portfolio management strategy building blocks into the modular API for the trading strategies. Since each company defines its own market as an instrument in the API, the strategy building blocks can be implemented with only a single company in mind each time.

4.3.1 Value Investing Indicator

The first thing needed is an indicator that gives access to the intrinsic value calculations for each company. These calculations are done historically for each financial statement. On that basis a value for the intrinsic value gets ascertained coupled to a time frame for which it counts. Since there is already a database with these calculations that can be filtered by time, it is possible to write an indicator that simply does a query for those. The query defines a maximum date for which the most current value is determined. During a backtest, the indicator thus always returns the correct intrinsic value for any given company.

The second indicator needed is simply one that returns the current market price of the company.

4.3.2 Value Investing Signal

These two indicators can be coupled to implement a comparing signal. It gives the following signals:

- **Buy:** Market Price <= Safety Price
- **Hold:** Safety Price < Market Price < Sticker Price
- **Sell:** Market Price >= Sticker Price

While sticker price can be simply calculated from the safety price indicator by multiplying that value by two.

This signal could be used as a signal trade entry and a signal trade exit. Though the trader might be advised to couple these signals with other technical analysis methods in order to find the best point of entry and exit.

For example, the trade entry signal could be coupled with a moving average crossover to get a signal that triggers a buy when the price starts rising right after it was falling during the time after the value investing signal started to trigger a buy. In this way, the trade does not get opened too early and drawdown can thus be minimized.

On the other side, the trade exit signal should be coupled with a trailing stop loss that gets enabled as soon as the sell signal is triggered. The trailing stop loss will then follow profits while the company becomes more and more overvalued. It will close the trade as soon as the price starts to fall again. In this way, it is possible to squeeze out more profit from a trade.

The strategy developer could also come up with a lot more ideas on how this could be improved. The idea behind the modular strategy API was to enable him to easily test his ideas, which surely helps here.

4.3.3 Value Investing Trade Filters

Since the value investing signal could also trigger for companies that are not in fact proper value investments, the trading strategy should also incorporate the following trade filters as examples:

- **Payback Time:** This value should be less or equal to ten to only allow companies to be traded that actually make good profit in relation to their market capitalization. The profit in fact drives growth which is the basis for the intrinsic value calculation used here.
- **Market Liquidity:** It is advisable to only trade companies where enough volume of trades occur on a daily basis, so that the buy and sell actions do not themselves manipulate the market price. The purchase volume should not exceed 1% of the overall daily trade volume. This is important since backtests do not model changes in price that are caused by the strategies own actions. Also this filters out companies that are not very popular for investors, since other investors are the drivers who cause the price to rise when they buy the company.
- **Long Term Debt:** This should filter out companies that have a long term debt that cannot be paid back within three years by positive free cash flow. This prevents the strategy to buy companies that have a high risk of going bankrupt by not being able to pay off their debt.
- **Stable Growth:** This filter could apply a threshold on the growth grade (a school grade system for stable growth as seen in [Stock2Own, 2012]). This will filter out companies that are not stable enough in their growth, which again is a prerequisite for the intrinsic value calculation used here.

Other things the strategy developer could experiment with might be the relation between price to tangible book value, various other ways to determine stable growth or playing around with weighted averages to find companies that have a rising historical growth rate instead of a declining one. Here again these filters can be included in the variability model of the trading strategy and each strategy combination can be automatically tested for robustness and profitability.

The strategy itself could run on each company market simultaneously to test the rules over the whole portfolio of companies. This is in fact required for the test, since these rules represent the portfolio management strategy.

4.3.4 Value Investing Parameters

Every rule described here can be chosen inside a variability model as design parameters to find out which combinations of rules are the best. Additionally, every of these rules has some sort of threshold constant in it, which could be modeled as an optimization parameter. This way the threshold value would not simply be based on a good guess or some recommendation, but instead based on statistical evaluation. This might as well help in improving the profitability of the strategy. Also this could improve robustness by making the strategy adapt itself to changing market conditions. Examples might be the threshold for the long term debt or the discount between sticker price and safety price. Maybe in some market conditions it shows itself to be advisable to become a bit more risky to do some big catches. On the other hand it might be interesting to see what negative effects can be observed when some of these safety rules are not followed rigorously.

Using these strategy building blocks that implement rules for the value investing strategy, it should be possible to run backtests on an automated version of this style of investing. Since this thesis introduced some concepts which have not been implemented yet in the platform and implementing those will take quite a while, only time will tell if this strategy actually works when it is automated.

This concludes the concept chapter and with it the main content of this thesis. In the following, a conclusion will be made based on the knowledge that was gained while working on this thesis.

5 Conclusion

The first goal of this thesis was to automatically generate strategies from building blocks and testing them. This was achieved by researching strategies in depth and developing a modular strategy API for reuse of rules.

The second goal of this thesis was to help in, or automate the decision processes of choosing value investments and strategies to trade them with.

Automating the choosing process of value investments showed itself not to be possible in all aspects, thus a decision was made during the analysis, that a few tradeoffs needed to be made for this to work. Using these tradeoffs and combining the already existing database of value investing calculations with the modular strategy API, it shows itself that this goal can be achieved partially. Though the actual trading might result in some sort of semi-automation which also respects the subjective steps of the decision process.

The decision process of choosing strategies to trade those value investments with, showed itself to be similar in nature of drawbacks due to automation. This includes the requirement of the developer having a good understanding about the variability model of the strategies and the automated design process being very time consuming when operating on the whole variability model. Nevertheless, using intelligent mechanisms to generate strategy combinations and choosing good combinations can compensate these problems. In that regard, this objective seems to be achieved from the current point of view.

Overall, this thesis thus succeeded in defining a concept for a platform that realizes these objectives.

5.1 Categorization

The development process of this concept can be categorized as a new approach to strategy development that fits between a manual approach (as defined by the simple and scientific strategy development process) and a fully automated approach (as seen in TradingSystemLab using a genetic programming AI). This is illustrated by the following figure[36]:

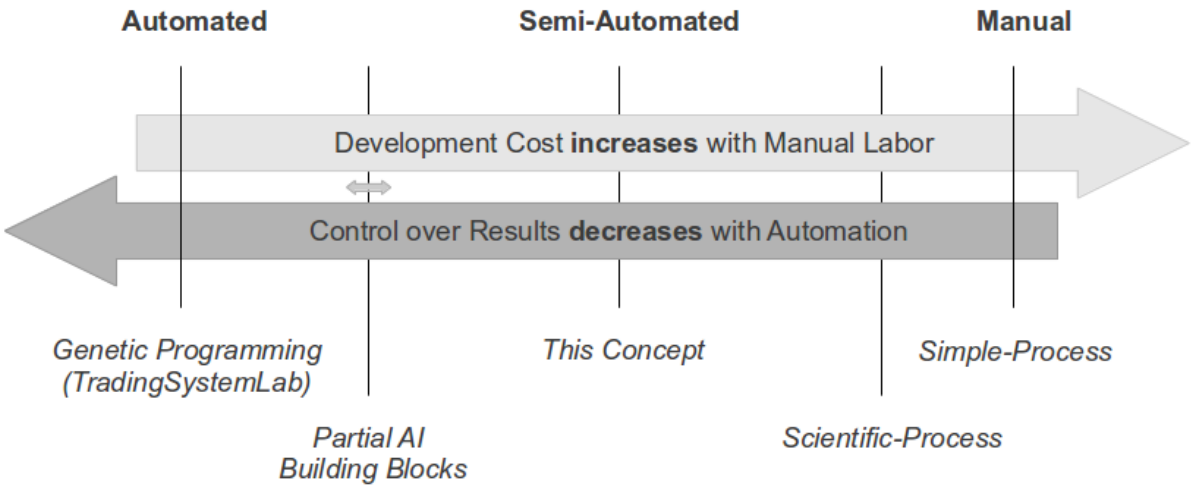


Figure 19: Categorization of this Concept between a Manual and Automated Strategy Development Approach

The relation of cost and control in this spectrum is similar to the relation of cost and flexibility in the spectrum known from software product lines as it can be derived from [Northrop and Clements, 2007]. That spectrum compares standard software with custom software. There the cost increases with customization and flexibility decreases with standardization. The software product line finds a spot in between these two extremes to share benefits and reduce drawbacks of both sides.

One could conclude that the development process of this concept is some sort of software product line for trading strategies. This concept is able to share benefits and reduce drawbacks of both sides of the described approaches to strategy development. This is explained in the following chapter by describing the benefits that can be seen when comparing this approach to each side individually.

[36] The concept described here can also use partial AI building blocks to vary the degree of automation to come nearer to the position of TradingSystemLab inside the spectrum.

5.2 Benefits

Using the concept described here to develop automated trading strategies should give the following benefits in comparison to a manual strategy development process:

- *Less development effort due to reusability of rules.* Classical strategies often reinvent the wheel when implementing most of the rules. This is unlikely to happen with strategies here, because a design for reuse is enforced by the modular strategy API.
- *Automated design and validation of strategies.* This enables the developer to have more time to think about his ideas instead of doing menial labor. The platform finds out for him whether the idea has potential or not. Classical strategies take a very long time and a lot of thinking to try a lot of rule combinations manually until a profitable combination is found. This involves many iterations of modifying the code, configuring the backtest, waiting for it to finish, determining improvements and doing the cycle again. Using the platform presented here, the developer once defines every possible combination by creating a variability model, lets the platform validate that and then just looks at the results after validation is finished.
- *Thorough and statistically based strategy development.* Problems of classical strategy development are prevented by the process. Examples are: Producing strategies that are overfitted, had insufficient testing or are not robust enough for live trading. The platform automatically applies best practices to avoid common pitfalls.
- *Faster analysis of potential strategies.* Testing strategies in a live environment is impossible, since it would take years when doing this in real time. Instead, the platform runs backtests, a lot of backtests in parallel, to validate the strategies. Compared to most other platforms, this saves a lot of time when not having to wait too long for desired results. The platform achieves this by harnessing the computing power of a cluster of machines.

In comparison to a fully automated process like TradingSystemLab offers, this concept should provide the following benefits:

- *Harnessing the intelligence of the developer.* The developer still has control about which rules are used in the strategy and how they can be combined. So he has the potential of using his experience to influence the generation of the strategy combinations. The developer also profits from this, because he increases his experience further while doing that.
- *Easier reverse engineering of strategies.* Since the developer writes the rules and defines the possible combinations, he can easily understand the test reports to get a clue why a specific strategy works or works not. When the strategy is generated as a whole like done using the genetic programming approach of TradingSystemLab, the developer might not be able to understand why something works or works not, even though there is generated code he can look at. Though the developer looses this advantage partially if he decides to have AI strategy building blocks, which are again harder to reverse engineer.
- *Being able to implement a strategy based on specific rules.* With the genetic programming approach, TradingSystemLab comes up with its own rules to trade a given market. This makes it hard for a developer to implement a strategy based on a functional specification of a strategy that includes fixed rules and expects the developer to implement these as it is normally expected from the manual strategy development process. TradingSystemLab also does not support development of semi-automated strategies. In comparison, the concept provided here is compatible to the manual approach and allows to develop semi-automated strategies. It even makes this more efficient than the manual approach.
- *Higher trust in the strategies.* The strategies developed here can be understood easily, because they follow known and specified rules. This allows the developer to have trust in his own strategies. If he developed a strategy for a client, the client can also trust the strategies, because they behave as he requested them to be implemented.

This makes the platform attractive to be used as a new approach for developing strategies.

5.3 Comparison

Despite looking at the sides of the spectrum, it is also interesting to look at how this concept compares to various other popular platforms available. For this, the following table considers a few functional aspects that are oriented to the strategy development process. This does not compare subjective criteria like usability and complexity of the platforms, which depends on the target audience. Also this does not go into detail about how the specific aspects are implemented or to what degree. It only notes if the aspect is somehow available.

Platform	**API**	**MS**	**PO**	**WFA**	**ASD**	**PFD**	**Process**
This Concept	X	X	X	X	X	X	Variable*2
TradingSystemLab				X	X	X*3	Automated
TradeStation[37]	X		X	X	X*1	X	Scientific,Automated*1
MetaStock[38]	X		X	X*1	X*1	X	Simple*4,Automated*1
M4[39]	X		X		X	X	Simple,Automated
MetaTrader[40]	X		X	X*1			Simple,Scientific*1
TradingBlox	X	X*5	X	X			Scientific
SmartQuant[41]	X	X*6	X	X		X	Scientific
AmiBroker[42]	X		X	X		X	Scientific
NinjaTrader[43]	X		X	X			Scientific
MultiCharts[44]	X		X	X			Scientific
TradeNavigator[45]	X	X*7	X				Simple
JForex	X		X				Simple
WealthLab[46]	X		X			X	Simple
XTrader[47]	X	X*8					Simple
EclipseTrader[48]	X						Simple

Table 1: Platform Comparison

Legend:
API = Strategy API | MS = Modular Strategies | X = Available
PO = Parameter Optimization | WFA = Walk-Forward-Analysis
ASD = Automated Strategy Design | PFD = Provided/Pluggable Fundamental Data

Markers:
*1 By purchasing an extension.
*2 Towards simple by not using the modular strategy API or more automated by using partial AI building blocks.
*3 In TradingSystemLab, the fundamental data needs to be preprocessed to be usable.
*4 In MetaStock, the WFA is only available for the genetic programming strategies and not the normal strategy API.
*5 TradingBlox has a coarse granularity and strict isolation for the strategy building blocks.
*6 The new frontend for SmartQuant, called VisualQuant, provides a graphical programming platform.
*7 TradeNavigator is based on a more mathematical or physics oriented approach instead of object oriented.
*8 XTrader has a graphical programming environment that is based on strategy building blocks.

Neither XTrader, TradeNavigator, SmartQuant, nor TradingBlox with their modular strategies provide building blocks that are suitable for variability models with arbitrary constraints among rules to generate strategy combinations.

It would be possible to implement automated value investing with some of the other platforms that are compared here. The fundamental data must either be provided by the platform, or the strategy API should allow this to be added by the developer himself. The lack of this rules out the platforms that have not been marked on that aspect in the previous comparison table.

The platform should also be able to trade stocks via a built in broker connector. The lack of this rules out the forex platforms MetaTrader and JForex. Even if TradingBlox provided fundamental data, it would be dismissed here, since it can only output trades to files without being able to connect itself to brokers. TradingSystemLab needs some enhancement to be able to facilitate portfolio management required for value investing, thus this also will not be usable. Though it could also be ruled out because the developer can only provide the preprocessed data on which basis the platform develops its own rules. Thus it can not be reliably said that the resulting strategy really implements value investing. Since the scientific strategy development process was determined to be an important ingredient for developing robust strategies, this rules out WealthLab, M4 and MetaStock as candidates as well.

This leaves AmiBroker, SmartQuant, TradeStation and this concept on the list of suitable candidates. They are valid platforms with which a proof of concept for an automated value investing strategy can be developed.

Though the modular strategy API, in combination with the use of variability models to respect constraints among rules, showed that an innovation in the strategy development platform market is possible that saves development time in the long run and can provide a benefit in developing various other strategies. This makes it interesting to develop a new platform based on that concept to validate these estimations. What is left to see is whether this platform is feasible or not.

[37] [TradeStationGroup, 2013] with the extension [AdaptradeSoftware, 2013]
[38] [MetaStock, 2013] with the extension [SecurityGenetics, 2013] or [TrendMedium, 2013]
[39] [ModulusFinancialEngineering, 2013]
[40] [MetaQuotesSoftwareCorp, 2013] with the extension [EasyExpertForex, 2013]
[41] [SmartQuant, 2013]
[42] [AmiBroker, 2013]
[43] [NinjaTrader, 2013]
[44] [MultiCharts, 2013]
[45] [GenesisFinancialTechnologies, 2013]
[46] [MS123, 2013]
[47] [TradingTechnologiesInternational, 2013]
[48] [EclipseTrader, 2013]

5.4 Feasibility

The question about feasibility of this platform can be answered by looking at what efforts and hurdles are required to be overcome to implement this. This is structured in the following aspects:

- **Monetary:** Since the chosen frameworks and tools here incur no running costs to the research project, it seems that development does not underlie any time constraints and thus can go on at the same pace it has been done until now.
- **Motivational:** Motivation is the driver that keeps one to stick to an idea, which itself is driven by the benefits one can get from this. Spending time developing this increases knowledge in many interesting subjects and has the potential of paying itself back at some time in the future. So as long as this does not change, feasibility does not seem to be endangered by this. Even if value investing cannot be successfully automated with this, surely some other strategy can be.
- **Timely:** Working on this subject in a team could reduce the development time by a manifold. Though this is also tightly bound to the monetary subject, where low risk is given by not incurring any running costs. So for now, time has to be put as a low priority on the feasibility, since the aim of this thesis was also to be visionary about a few years into the future, when counting by the current pace of development. The time horizon is not expanded too much by the elements of the concept and seems achievable within a few years of development.
- **Technological:** Even though the strategy testing seems to be quite computing intensive, it still seems that todays technology is far enough to make this concept possible and practicable for every day use. There are many pieces of software that already solve large pieces of the problems described here, which should boost development time as well. The ideas here also seem to be technologically possible to be realized.

Even though this is not a complete feasibility analysis (which also was not the goal), this still gives a positive outlook about the platform being feasible.

Thus let the time and continuing effort work its magic on this one...

Bibliography

a) Documents and Books

Amazon.
Amazon Elastic MapReduce Developer Guide, 4 2012.
URL `http://amazon.com/o/ASIN/B007US6CIO/`.

Heinz-Dieter Assmann and Uwe H. Schneider, editors.
Wertpapierhandelsgesetz.
Schmidt (Otto), Köln, 6. neu bearbeitete und erweiterte Auflage 2012. edition, 2 2012.
ISBN 9783504400880.
URL `http://amazon.de/o/ASIN/3504400889/`.

Rogers Cadenhead.
Sams Teach Yourself Java in 21 Days (Covering Java 7 and Android) (6th Edition).
Sams Publishing, 6 edition, 8 2012.
ISBN 9780672335747.
URL `http://amazon.com/o/ASIN/0672335743/`.

Robert W. Colby.
The Encyclopedia Of Technical Market Indicators, Second Edition.
McGraw-Hill, 2 edition, 10 2002.
ISBN 9780070120570.
URL `http://amazon.com/o/ASIN/0070120579/`.

L.M. Deschaine and F.D. Francone.
Comparison of Discipulus™ Linear Genetic Programming Software with Support Vector Machines, Classification Trees, Neural Networks and Human Experts.
White Paper, RML Technologies, Inc, 2004.
URL `http://jcpldih.rmltech.com/doclink/Comparison.White.Paper.pdf`.

Martin Fowler.
UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition).

Addison-Wesley Professional, 3 edition, 9 2003.
ISBN 9780321193681.
URL http://amazon.com/o/ASIN/0321193687/.

Benjamin Graham and David Dodd.
Security Analysis: Sixth Edition, Foreword by Warren Buffett (Security Analysis Prior Editions).
McGraw-Hill, 6 edition, 9 2008.
ISBN 9780071592536.
URL http://amazon.com/o/ASIN/0071592539/.

Benjamin Graham and Jason Zweig.
The Intelligent Investor: The Definitive Book on Value Investing. A Book of Practical Counsel (Revised Edition).
Collins Business, revised edition, 7 2003.
ISBN 9780060555665.
URL http://amazon.com/o/ASIN/0060555661/.

Michael Griffis and Lita Epstein.
Trading For Dummies.
For Dummies, 2 edition, 6 2009.
ISBN 9780470438404.
URL http://amazon.com/o/ASIN/0470438401/.

K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson.
Feature-oriented domain analysis (FODA) feasibility study.
Technical report, DTIC Document, 1990.
URL http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA235785.

W. Chan Kim and Renee Mauborgne.
Blue Ocean Strategy: How to Create Uncontested Market Space and Make Competition Irrelevant.
Harvard Business Review Press, 1 edition, 2 2005.
ISBN 9781591396192.
URL http://amazon.com/o/ASIN/1591396190/.

Matt Krantz.
Fundamental Analysis For Dummies.

For Dummies, 1 edition, 11 2009.
ISBN 9780470506455.
URL http://amazon.com/o/ASIN/0470506458/.

Andrew W. Lo.
Efficient markets hypothesis.
THE NEW PALGRAVE: A DICTIONARY OF ECONOMICS, 2007.
URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=991509.

B.G. Malkiel.
The efficient market hypothesis and its critics.
Journal of Economic Perspectives, pages 59–82, 2003.
URL http://www.jstor.org/discover/10.2307/3216840?uid=3737864&uid=2&uid=4&sid=21101768524431.

Frederic S. Mishkin and Stanley Eakins.
Financial Markets and Institutions (6th Edition).
Prentice Hall, 6 edition, 1 2008.
ISBN 9780321374219.
URL http://amazon.com/o/ASIN/0321374215/.

Robert Pardo.
The Evaluation and Optimization of Trading Strategies (Wiley Trading).
Wiley, 2 edition, 2 2008.
ISBN 9780470128015.
URL http://amazon.com/o/ASIN/0470128011/.

Marco Antonio Penteado.
Efficient Markets, Behavioral Finance and a Statistical Evidence of the Validity of Technical Asnalysis.
arXiv preprint arXiv:1302.1228, 2013.

Michael E. Porter.
Competitive Advantage: Creating and Sustaining Superior Performance.
Free Press, 1 edition, 6 1998a.
ISBN 9780684841465.
URL http://amazon.com/o/ASIN/0684841460/.

Michael E. Porter.
Competitive Advantage: Creating and Sustaining Superior Performance.

Free Press, 1 edition, 6 1998b.
ISBN 9780684841465.
URL http://amazon.com/o/ASIN/0684841460/.

Murray A. Ruggiero.
Cybernetic Trading Strategies: Developing a Profitable Trading System with State-of-the-Art Technologies.
Wiley, 1 edition, 6 1997.
ISBN 9780471149200.
URL http://amazon.com/o/ASIN/0471149209/.

Shushobhickae Singh, Chitrangada Nemani, et al.
Fluent interfaces.
ACEEE International Journal on Information Technology, 1(2), 2011.
URL http://hal.archives-ouvertes.fr/hal-00753271/.

Phil Town.
Rule #1: The Simple Strategy for Successful Investing in Only 15 Minutes a Week!
Crown Business, 2007.
ISBN 9780307336842.
URL http://amazon.com/o/ASIN/0307336840/.

Phil Town.
Payback Time: Making Big Money Is the Best Revenge!
Crown Business, 3 2010.
ISBN 9780307461865.
URL http://amazon.com/o/ASIN/0307461866/.

Michael Voigt.
Das große Buch der Markttechnik: Auf der Suche nach der Qualität im Trading.
FinanzBuch Verlag, 2012.
URL http://amazon.com/o/ASIN/B0084ZUWRO/.

Tom White.
Hadoop: The Definitive Guide.
O'Reilly Media, third edition edition, 5 2012.
ISBN 9781449311520.
URL http://amazon.com/o/ASIN/1449311520/.

Charlie F. Wright.
Trading as a Business.
Charlie F. Wright, 1998.
URL http://amazon.com/o/ASIN/B0006ROD1O/.

b) Online Publications

AdaptradeSoftware.
Adaptrade Software : Trading Software : Day Trade Stock : Forex Day Trading System : Automated Trading : Day Trader : Trading Systems : Trading Strategies, 2013.
URL http://www.adaptrade.com/.

AmiBroker.
AmiBroker - Technical Analysis Software. Charting, Backtesting, Scanning of stocks, futures, mutual funds, forex (currencies). Alerts. Free quotes., 2013.
URL http://www.amibroker.com/.

Interactive Brokers.
Interactive Brokers website, 2013.
URL http://www.interactivebrokers.com/en/main.php.

Michael R. Bryant.
Building Trading Systems Using Automatic Code Generation, 2010.
URL http://www.adaptrade.com/Articles/WhitePaper-GP-Content.htm.

Dukascopy.
JForex Strategy API, 2013.
URL http://www.dukascopy.com/wiki/#Strategy_API.

EasyExpertForex.
Walk Forward Analyzer Software for MetaTrader :: Easy Expert Forex, 2013.
URL http://www.easyexpertforex.com/walk-forward-metatrader.html.

EclipseTrader.
EclipseTrader, 2013.
URL http://www.eclipsetrader.org/.

GenesisFinancialTechnologies.
www.tradenavigator.com - Index, 2013.
URL http://www.tradenavigator.com/.

Steve Johns.
Walk-Forward-Analysis, 2011.
URL http://codefortraders.com/Walk-Forward_Analysis/WFA_Introduction.htm.

MetaQuotesSoftwareCorp.
MetaTrader 5 Trading Platform / MetaQuotes Software Corp., 2013.
URL http://www.metaquotes.net/.

MetaStock.
MetaStock | trading analysis software | over 25 years | serious traders - MetaStock.com, 2013.
URL http://www.metastock.com/.

ModulusFinancialEngineering.
Trading Platform Source Code with Charting, Technical Anlaysis, Scanning, Backtesting and more., 2013.
URL http://www.modulusfe.com/m4/.

MS123.
Wealth Lab: Technical Analysis Software Trading Platform - Portfolio backtesting software, stock charting, strategy scripts and much more..., 2013.
URL http://www.wealth-lab.com/.

MultiCharts.
Trading Software for Automated Trading and Backtesting | Multicharts Trading Platform, 2013.
URL http://www.multicharts.com/.

NinjaTrader.
NinjaTrader stock, futures and forex charting software and online trading platform., 2013.
URL http://www.ninjatrader.com/.

Linda Northrop and Paul Clements.
A Framework for Software Product Line Practice: Version 5.0.

Carnegie Mellon University, Software Engineering Institute, 2007.
URL http://www.sei.cmu.edu/productlines/frame_report/index.html.

Prescience.
ABAT Final Report, 2011.
URL http://presciencefunds.com/documents/ABATFinalReport.pdf.

Resin.
Java On Raspberry Pi Performance - Resin 4.0 Wiki, 2013.
URL http://wiki4.caucho.com/Java_On_Raspberry_Pi_Performance.

SecurityGenetics.
Genetic System Search for Technical Analysis - Genetic System Search for Technical Analysis, 2013.
URL http://www.gssta.com/.

SmartQuant.
Automated Trading Strategies Development Platform, 2013.
URL http://www.smartquant.com.

University of Southampton.
Raspberry Pi at Southampton, 2013.
URL http://www.southampton.ac.uk/~sjc/raspberrypi/.

Stock2Own.
Value Investing Analysis for Microsoft, 2012.
URL http://www.stock2own.com/StockAnalyzer.aspx?s=US:MSFT.

Structured Software Systems.
An Overview of TREE: TREE structure: the big picture, 1997.
URL http://www.strsoft.com/intro/treeintro-2.html.

TradeStationGroup.
Online Trading | Trade Stocks, Options, Futures Forex Online | Trading Software, 2013.
URL http://www.tradestation.com/.

TradingBlox.
Trading Blox - System Development And Portfolio Backtesting, 2013.
URL http://www.tradingblox.com/.

TradingSystemLab.
Trading System Lab, 2013.
URL http://www.tradingsystemlab.com/.

TradingTechnologiesInternational.
X_TRADER | Trading Technologies International Inc, 2013.
URL https://www.tradingtechnologies.com/xtrader/.

TrendMedium.
TrendMedium Intelligent Stock Profiling, 2013.
URL http://www.trendmedium.com/.

VisualPatternDesignerProfessional.
Visual Pattern Designer Professional: Getting Started Manual, 1992.
URL http://www.scrigroup.com/limba/engleza/106/Visual-Pattern-Designer-Profes33917.php.

Voofie.
Value investing techniques for return of multiple times, 2013.
URL http://www.voofie.com/content/27/value-investing-techniques-for\-return-of-multiple-times/.

Declaration of Honor

The author hereby ensures that the supplied master thesis has been created without external help or the use of sources and aid other than referenced. Also that the by-word or by-meaning extracted parts of used sources are stated or identifiable as such. This or a similar work has not been submitted to an examinig authority yet.

Bad Lippspringe, March 22, 2013

Signature