Software Re-engineering A Legacy Application to Create More Manageable Source Code

Jörgen Skoog
Software Re-engineering A Legacy Application to Create More Manageable Source Code

Jörgen Skoog
Dept. of Computer Science,
University of Trollhättan/Uddevalla,
P.O. Box 957, SE-46129
Trollhättan, Sweden
Jorgen.skoog@student.htu.se

Abstract

The need for maintenance and improvements of old systems has risen dramatically and the need of converting along with the need to convert old programs written in procedural language into a more manageable form. The program StålSpec 2000 from MVR (Mekaniska verkstädernas riksförbund) is a good example of such a legacy application that need to be converted. By using re-engineering strategies to help create clearer and more standardized code, as much design and structure information as possible can be gathered to make the conversion into a more maintainable object oriented form as easy as it possible. Automated tools are used to do the most part of code translation but also much manual work for structure improvements and analysis of the old source code have been done.

Re-engineering principles have been used in this work together with a more user based approach to successfully recover and “clean up” the code of StålSpec 2000 to aid rewriting procedural code into object oriented. We have only implemented some small functionality in the new StålSpec; time was the contributing factor why not more were done. Time is money is a popular expression and in re-engineering and rewriting time is needed in large amounts.

1. Introduction

Over time new software becomes old software. Most software is still maintained because it works and is a vital part of an organisation. These old systems and applications are so called legacy applications, systems and code that are business critical, meaning businesses are dependent of the daily services provided by these systems. A system failure will have serious consequences for example health information systems which stores the journals of thousands of patients. Not only does legacy code exist in old applications and other computer systems, people write legacy code even today. [1, 2]

There are number of ways of upgrading legacy applications to be more manageable; one such way is re-engineering. Re-engineering is something between a totally new development and renovation. Successful reengineering is a combination of reimplemention of the existing application and new additions of other capabilities such as, user based recovery. [3]

The legacy application we are going to work with is a program suite from Mekniska Verkstädernas Riksförbund (MVR) [4] called StålSpec 2000. The purpose of the program is to be a tool for helping out construction companies to calculate the amount of materials needed such as steel beams/sheets and man hours to fulfill a given contract. Customers can also request tenders on work, and the program provides it.

StålSpec 2000 is written in Visual Basic6 (VB6). VB6 is a structured programming language and effectively a different language from the more recent VB.NET. All versions of Visual Basic up to version 6 were backwards compatible and could be built and run on any version with minimal changes. [5, 6, 7]

The goal with this thesis is to re-engineer as much as possible of the StålSpec 2000, analyze the source code and the occurring problems and hopefully produce a more readable and cleaner code, to help us in the development of the next version of StålSpec. The old program and source code acts as requirement specification.

StålSpec 2000 is a textbook example of a legacy application with its tangled structure and cryptic variables.

In section 3 a brief summary of what legacy code is, section 4 is about software re-engineering, sections 5 discusses the benefits, costs and goals with software
re-engineering, section 6 the legacy application StålSpec 2000 and source code is analyzed, section 7 why re-engineer StålSpec 2000, section 8 the requirements specification is specified. In the section 9 to 14 the tools and coding practice used are to be found. Within the sections 15, 16 and 17 the result, conclusion and future work are to be found.

2. Background

MVR created a program to help out their members for handling of tenders, steel calculations, cutting optimizations and the amount of man hours needed, to name a few functions. The program created was named StålSpec 2000 and written in VB6 which is a structured programming language. The StålSpec 2000 application contains a lot of bugs and other related issues.

The way of using StålSpec 2000 varies from company to company and they have learned to get around the bugs or just not use the functionality that makes the program malfunction.

MVR wanted us from the beginning to bug fix StålSpec 2000, but after discussions we all agreed that future maintenance depends on the existence of a good development environment. Microsoft suite VB6 has no further support from Microsoft which is now concentrating all its efforts on the new .NET development environment. [5] The VS6 suite is also hard to get hold of because it is not sold in stores anymore. So instead of working with the old VS 6 environment that has less and less support every year. We agreed on for easier maintenance in the future the development environment as well as the StålSpec 2000 has to be changed to meet the new needs.

StålSpec 2000 has to be object oriented, written in a modern language with future support and component-based so new components can be added without writing any new code in the main StålSpec 2000 program. This will ensure StålSpec 2000 to be a versatile program, which can be custom built to fit the different customer’s needs. They could start with the basic functionality if they want and then add new component in pace with increasing need of new functionality.

The choice of programming language C-Sharp (C#) is based on the fact that C# is a modern OO language and uses the new Common Language Runtime (CLR) [8] provided by the .NET Framework [6].

The CLR manages code execution and allows execution of code created in different languages to interact and hence removes the language barrier. It also offering a new component model based upon OO concepts, and removing the distinction between program element and software component.

The .NET Framework [6] also provides automatic memory management using a mechanism called garbage collector. The CLR automatically detects if a program no longer is using the memory and recycles it and limits memory leaks. It also has support for a wide variety of languages to be used during development of software. By removing the language barrier, one could use VB.NET to add functionality to C# code, even with Visual C++ it’s all up to the developer.

Any one or a combination of these languages can be used to create ASP Web applications, XML web services, desktop applications and even mobile applications based on the .NET Framework. [6, 7, 8, 9]

3. Legacy code

Legacy code is code from the past, which is maintained because it works. The need for maintenance and improvements of old systems has risen dramatically over the past decade. Old software constitutes large assets to corporations and governments because legacy code exists in applications such as:

- Switching systems
- Banking systems
- Avionics systems
- Health information systems

These areas are of great importance in the daily life of millions of persons because they are still business critical. Business critical means that business relies on the services provided by the software and failures will have serious effects on the day-to-day running of the business. These old systems have been given the name legacy systems. Almost all legacy systems where designed before object oriented development was used in the industry. [1, 2, 3]

“Legacy applications are defined as large software applications that are difficult to cope with but are vital to an organisation.” [10]

Not only does legacy code exist in old programs, people write legacy code even today. The difference that distinguishes legacy code and non legacy code is tests, or rather the lack of tests. Making changes in large systems most of the fear bases of introducing
subtle bugs, changing things inadvertently in the source code. \[1, 2, 3\]

Legacy code is not only hard to maintain, it has several other disadvantages as well:

- The older the legacy applications are the more serious are its disadvantages.
- To maintain and extend the functionality of legacy software is difficult, especially those written in functional languages as C or FORTRAN.
- Large investments of money and human effort are required when rewriting.
- Substantial duplication of code for the same functionality where the code only differs in the data type.
- Takes no advantages of modern processor design, because most of the code was written when thread programming was in its infancy and distributed computing was non-existent
- Rarely a complete specification of the legacy system. It could be lost, but if it exist it is unlikely that all changes have been documented. Therefore there is no straightforward way of specifying a new system that has identical functionality like the system in use.
- Years of maintenance have corrupted the systems structure making it difficult to understand. Programs have been added and interfaced with other parts of the system in an ad-hoc way.

Decisions made by engineers in the development phase of these early systems were often specific to the task at hand. This resulted in systems that were not designed for direct reuse. Maintenance and upgrades over time often makes the original architectural design to degrade. Due to multiple sets of small modifications and local optimizations made over time. As a result many legacy applications and designs are costly to update because of non-uniformity and brittleness. \[3, 10, 11, 12, 13\]

Businesses with a large number of legacy software are faced with a dilemma. If they continue using legacy software and keep maintaining it, the cost will evidently increase. If they replace the old system with a new system this will be costly as well and the new system may not provide as effective business support as the old system.

Consequently many businesses are looking at software techniques that will increase the life of legacy software and reduce cost of keeping these systems in use. One of these techniques are re-engineering.

4. Software Re-engineering

Re-engineering is something between totally new development and renovation. Successful re-engineering is a combination of reimplementation of existing application and the addition of new and expanded capabilities. The re-engineering process consists of five steps.

- Data re-engineering
- Source code translation
- Reverse engineering
- Program structure improvement
- Program modularization

Software re-engineering does not necessarily require all of the above steps. If the compiler supports the program language there is no need for source code translation.

Reverse engineering to recover documentation may not be required if the re-engineering process is not completely automated tools. \[1, 2, 3\]

4.1 Data re-engineering

The process of analyzing and reorganize the data structures and in some cases the values of in a system to make it more understandable is called data re-engineering. There are a number of reasons why data re-engineering is needed in legacy systems.

- Data degradation: Data enters daily from internal systems and extranets as well as from the web and changes to the data. Resulting in data issues such as inconsistencies in the data entered which generate duplications. Entries in the wrong fields and other errors such as typos. Creates problems to get the accurate information, causing mistakes, delays and irritation.
- Inherent limits: Different platforms /operating systems have different byte length, word length, precision, registers even memory requirements differs. Even built in constraints on maximum amount of data that could be processed. All of these have to be accounted for when changing platform or operating system.
- Architectural evolution: Migration of a centralized system into a distributed
architecture it is essential that the core of that architecture should be a data management system to ensure that remote clients can access it. May require a large data re-engineering effort to move data from separate files into a server database management system. [3, 10, 12, 13]

4.2 Source code translation

By translation the legacy code into a new language an automated tool to help out the conversion is the only realistic and economic approach and in the section 9.1 Line-for-line translation, more information and analysis results are available.

4.3 Reverse engineering

The objective with the reverse engineering process is to derive the specification or design of a system from its source code. Reverse engineering may be a part of the re-engineering phase to recover program design to help engineers understand the program before reorganizing the structure. But it can also be used in re-specify a system for reimplementation. As stated before reverse engineering may not be required if the whole process relies on automated tools.[3, 14]

4.4 Program structure improvements

Structure improvements to legacy systems are needed when the systems are poorly structured. Due to lack of understanding of the software engineering principles many programmers produce tangled code. Their control structure creates a tangled system structure with many unconditional branches and unintuitive control logic.

The use of cryptic variable names in the code makes it even harder to understand, and maintenance tends to corrupt the structure as well. Programs develop this complex logic structure as they are modified and maintained by adding conditions and other actions without changing the existing control structure. This is in short term thinking quicker and less risky because it reduces the risk of adding faults. In the long run, however it unavoidable leads to incomprehensible code. [3, 14]

4.5 Program modularization

Program modularization refers to re-organization of a program so that related parts are grouped together in a single module. When grouped in modules it is easier to remove redundancies in the related components. [3, 14]

5. Software Re-engineering benefits

Re-engineering has benefits in comparison to new development; it reduces the risks of development and specification problems that might occur when developing new software. The cost of new development is increasingly higher than the cost of re-engineering the old software. [2, 3, 14]

5.1 The cost of Software Re-engineering

The cost of re-engineering legacy code depends on the extent of work that is needed. Other factors that affect the costs of re-engineering are.

- The lower quality of the software and associated documentation, the higher the cost
- Normally it is not cost effective to re-engineer a software system without using CASE tools to automate most of the program changes.
- Large volumes of data needed to be converted into a new format, significantly increases the process cost.
- The availability of the expert staff.

The main goal is to have a “good enough” quality and not perfection because no one can afford perfection when it comes to re-engineering legacy code. Because the project comes to an end not when it reaches perfection, but when the budget and time are played out. Ready or not this new system has to go into production. [3, 12]

5.2 Software Re-Engineering goals

Main goal with re-engineering is to understand the how’s, what’s and whys of the legacy system, and generate from it a new system, who will meet the new requirements, behavior, demands and performance issues needed to ensure its maintainability.

6. Visual Basic 6

Previous versions of Visual Basic up to version 6 were backwards compatible, but with the new .NET environment it is not one of Microsoft’s high-priority goals. [5]

VB.NET is based on Common Language Runtime (CLR) which is the new execution platform for all .NET languages including VB, managed C++, C#.
Some of the improvements of CLR are performance improvements, ability to easily use components and inheritance [5, 6, 7, 8]. In fact it is better to rewrite the application rather than port it, in order to really take best advantage of the new features and structures of the .NET environment. Further discussions are to be found in section 11 Conclusion.

6.1 The StålSpec 2000 application and source code

The structure of the code is more or less spaghetti code meaning that GOTO statements are used throughout the code. GOTO statements access variables and functions; those in turn call upon other global variables, which are used all over the program to make more calls. StålSpec 2000 also contains a lot of bugs and malfunctions; as a result of all the bugs and malfunctions. Some customers even have versions and bug fixes made that no one else has, because they personally contacted the developer with requests about additional functionality and bug fixes.

The new additions were made at the customer to his or hers application and not released to other license holders as updates. Even with the release of a new version only few bother downloading it. Because the new version contained bugs that older versions did not have so these users kept the old version. Instead of fixing the bug the developer works around them creating other bugs in a tangled manner.

This is a recurring problem because of the different uses of the StålSpec 2000 application. Some customers only use it to create tenders and calculations on materials needed to fill orders and other cost estimations. Others only use it to print out cutting lists created by the cutting optimization part of the software for internal use. Some use little of everything depending on the business concept. 150 licenses have been sold and all companies have different needs and requests of functionality, creating even more errors and bugs. With no record on what have been done and where, upgrading the software removes the custom functions, create faults and bugs where there haven’t been any before.

A standardized way of handling the different requests, creation of bug reports and most important of all, a standardized way of writing the code and comments in the code are all required. This should have been done from start, and now it is too late; not even the creator of StålSpec 2000 knows what the different parts of the code do. [3]

In section 7.2 we analyze the code to get a better view of the different problems and other things we can be expecting when attempting a software re-engineering.

6.1.1 Problems with the existing code. The developer of StålSpec 2000 has not used any coding standard, no naming convention for naming the variables used and the use of hundreds of global variables thru out the code. The developer also uses GOTO statements which produces spaghetti code. [15] On top of that there are almost no comments in the code explaining the different functions, making approximately 43000 rows of code to a nearly impossible task to understand and pin point the different subroutines and their tasks.

See section 10.1 Line-for-line translation result for more information about the code

6.1.2 Comments in the code. Code that is well documented makes the understanding of the different parts a lot easier. But without or very little comments makes it so much harder to understand the thoughts behind the different functions. The original code contains approximately 43000 rows of code and 2533 comments. We can remove 2296 comments that are auto generated by Visual Studio 6 (VS6), remove 213 comments used to exclude lines from the project when compiled. Six comments about a row deletion, three marks out the start of subroutines, and six comments states the value of PI. What remains are nine real comments explaining briefly functionality of the selected subroutines, which are fairly obvious, and the expected functionality of the subroutine is clear even without comments. More information about the comments is available in appendix A at the end of this paper.

7. Why re-engineer the StålSpec 2000 application.

StålSpec 2000 is one of a kind application and has no competitors yet. Approximate 150 companies’ uses this program on a regular basis to aid them in their work. As long as the StålSpec 2000 contains errors and bugs other software developers might see their chance of getting a piece of the market and create software of their own. MVR want to keep the dominant position, and in order to do so the program has to be updated and easier to maintain. The re-engineering process is cheaper then re-writing the program from scratch, and also faster.

8. Requirement specification

MVR have put together a reference group consisting of users that works with StålSpec 2000 on a
regular basis, to aid us in our work with the development of the next version of StålSpec. In meetings with the reference group we have put together a minimum requirement specification for the next version of StålSpec by removing functions that none or few uses and functionality not needed at the time. We agreed on focusing on the main functions in the old StålSpec 2000 and implement them in the new version. We agreed on a working name for the new software, StålSpec 3000 or SS3K for short.

This requirement specification is exactly what re-engineering is all about. New development starts with a written specification; here we start with the old programs graphical user interface (GUI) and source code as the requirement specification in true re-engineering spirit. [3]

8.1 Functionality

The basic functionality of SS3K will be presented very briefly.

- Creation of new StålSpec and alter existing.
- Preview and print.
- Pricelists for beams, steel sheets and other steel materials
- Pre-fabricated material prices,
- Special products and own products,
- Own profiles.
- Cutting optimization of old cutting list or create new list from existing cutting lists. Print cutting list with cutting figures.
- Suppliers register for personal use were additions to the list are made by the user.
- Create a tender for submitting to suppliers or wholesalers to get the best price for materials.

8.2 Approach

The main goal with re-engineering is to understand the how’s, what’s and whys of the legacy system. We only need to know the how’s, what’s and whys of the legacy system and cleaner and more understandable code.

Re-engineering legacy systems and applications are a challenge in its own right. There are different re-engineering approaches that can be used to tackle this challenge.

The re-engineering approaches used are line-for-line translation and structure improvement to give better structure and clearer code.

9. Automated re-engineering tools

The programmer must mentally keep track of the red thread between the different subroutines, forms and global variables used and what values they receive depending on what subroutine used at the time. An automated tool would reduce this time consuming task, so we needed a tool to do the analysis for us.

Re-engineering requires tools to capture, analyze and transform the existing code to make it more manageable. But there are still some serious problems with re-engineering tools for handling software written in multiple languages or with embedded macros.

Our source code does not contain any macros or multiple languages, only VB6 code, so we have not to worry about it. The only thing that could be troublesome is to find the appropriate tool to aid us.

10. Project Analyzer

We needed to analyze and check the state of the source code. An automated tool were needed to do the analysis for us.

After researching the Webb we found the program Project Analyzer v 7 (Demo) [16].

Project Analyzer v 7 generates call tree, problem reports and project graphs by analyzing the Visual Basic solution file and automatically adds the forms used by the solution file. Due to the limitations of the demo versions only 10 forms can be analyzed at the same time.

We wanted to do a comparison between the two latest versions of the source code of StålSpec 2000 to see if the numbers of errors increase or decrease.

The new source code of StålSpec contains 11 forms so one form have to be left out due to the demo limitation. To solve it we analyzed the code with one form out then removed another form and added the one left out from first analysis and did the analysis again. To check dependencies between forms and to verify that the problem report generates the same kind of report independently of the type of form added or removed when performing the problem analysis.

Previous version of StålSpec 2000 has 10 forms and can be used in the demo version as it is.

We found that the problem reports contents do not change and that dependencies between forms are not analyzed, only the files and its contents.

The first we extracted from the code with the help of project analyzer v 7 [16] was a problem report to pin point the different problems within the forms of the old and new source code of StålSpec 2000.
10.1 Problem Report

<table>
<thead>
<tr>
<th>The problem report summary: New source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of analyzed rows</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Optimization</td>
</tr>
<tr>
<td>Style</td>
</tr>
<tr>
<td>Functionality</td>
</tr>
<tr>
<td>Total</td>
</tr>
<tr>
<td>Average Problem/logical code line per form</td>
</tr>
</tbody>
</table>

Table 1. Problem report summary of the new source code.

<table>
<thead>
<tr>
<th>The problem report summary: Old source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of analyzed rows</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Optimization</td>
</tr>
<tr>
<td>Style</td>
</tr>
<tr>
<td>Functionality</td>
</tr>
<tr>
<td>Total</td>
</tr>
<tr>
<td>Average Problem/logical code line per form</td>
</tr>
</tbody>
</table>

Table 2. Problem report summary of the old source code.

Project Analyzer analyses the form files and generates a problem report with a summary of code declarations, GOTO statements etc. Even shows the different types of problems if it is an optimization, style or a functionality issue.

As we suspected there is a lot of problems in the source code of StälSpec 2000. Due to the extent of the problem report a short summary will be presented below. The full problem reports can be viewed in the end of this report as appendix.

Appendix C is the problem report of the new source code and appendix D is the problem report of the old source code.

The problem report summaries shown in Table 1 and Table 2. By comparing these two summaries we can see that a pattern emerges, even if we remove the new form added to the more recent source code the total number of errors only decreases with fourteen. Between these two versions, 133 new errors were added on 2686 new lines of code. It is approximate one new error on every twentieth line of new code.

Maintenance and upgrades have made the source code and architectural design to degrade. Something has to be done soon. Local optimizations and small modifications are made regularly making the source code to degrade even more.

11. Line-for-line translation

One re-engineering approach we thought would help us is a line-for-line translator to help us obtain better source code. This is the simplest form of software re-engineering. A line-for-line translator creates source code in a different language by creating a new line of code for each line of legacy code. But this approach is only realistic if there is such automatic translator available to do the main translation. [3, 14]

After researching the Webb we found the automatic translator program BabbleFisken [17]. This program seemed promising to us and hopefully it would aid us in our work. It translated directly from VB6 to C# other programs only translates from VB.NET to C#. To use a VB.NET to C# converter, the first thing to do is translate the VB6 code into VB.NET and then continue with the conversion to C# with another program. Microsoft has a wizard tool to help upgrade the VB6 code to VB.NET. [18, 19]

We decided it would be worth a try to go with the direct approach from VB6 to C# without taking the detour with an additional conversion. In many cases a completely automatic translation is impossible. Constructs in the source language may have no direct equivalent so manual changes have to be done to tune and improve the target language. [3, 14]

11.1 Line-for-line translation result

By translating the source code from VB6 to C# we thought it would really ease the task of understanding the source code as we have most experience of C#. It was not that simple, the line-for-line translator translated all the VB6 code to C#. Kept all the variable names, making subroutines to classes and all the hundreds of global variables were also just translated.

To illustrate the conversion a small example of the VB6 code will be presented along with a small sample of the new converted C# code made by the line-for-line translator BabbleFisken.

The VB6 code is actual code from our project and it is only one small subroutine used to alert the user that a change have been made and asking if the user want to save before exiting the program. See Listing 1. Its a general piece of code which also illustrates the lack of a naming convention.
Sub Villdu()
    Const IDYes = 6, IDNo = 7,
    mnuIconQuestion = 32, mnuYesNO = 4:
    Svar = 0
    Msg = "Indata är ändrad ."
    MsgBox(Msg + "Vill du spara innan du avslutar.",
    DgDef = mnuIconQuestion + mnuYesNO
    Svar = MsgBox(Msg, DgDef, "Spara...")
End Sub

Listing 1 A Small piece of VB6 code from the project.

The C# code snippet below is a direct translation of the VB6 code. The similarities between the two codes are clearly visible, not much has changed. See Listing 2.

void Villdu()
{
    const IDYes = 6, IDNo = 7,
    mnuIconQuestion = 32, mnuYesNO = 4:
    Svar = 0
    MsgBox(Msg + "Vill du spara innan du avslutar.",
    DgDef = mnuIconQuestion + mnuYesNO;
    Svar = MsgBox(Msg, DgDef, "Spara...");
}

Listing 2. The converted code from VB6 to C#.

Another tool we used is TransKing (demo) which also convert VB or VB.NET to C# but had some more powerful features then a strictly line-for-line converter as BabbleFisken. TransKing (Demo) only transforms a small amount of rows (600) but enough to get an opinion about the quality of the conversion and if it have any value to us. Before we rush away and purchase a copy.

In order to use the conversion tool Visual Studio.NET (VS.NET) has to be installed on the computer used and TransKing is run from its tool menu. It did not take long into the translation process before an error message dialog popped up.

The error message received when used the TransKing to translate the source code are as follows.

"Conversion failed, because:32Compiler failed with error32manifestdefinition

[20] Is the source code really that bad? Microsoft VS.NET has a built-in wizard for VB6 migration that we hope would work better then the previous two. Because VB6 and .NET are originated from the same company and should work fine.

The wizard is a step by step guide to help out with the upgrading. Even this one fails to upgrade the code, after only seconds the upgrade fails.

"Upgrade Failed: Exception occurred: The reference component CommonDialog is missing a design time license."

Even with the old version of the source code gives the exact same error messages both with TransKing and the Microsoft VS.NET upgrade wizard.

After researching the web for an explanation about why these errors occur. We came across a support page from Microsoft with answer to why the wizard did not work. [21]

To be able to upgrade from VB6 to VB.NET, VS6 must also be installed. We have been lent a VS6 copy from HTU to help aid us during the development and analysis of the source code. After installing the VS6 suite we once again did an attempt to convert the source code. First we tried with TransKing and hoped that the previous errors would disappear with the installation of VS6. Again the same error as before, VS6 installation did not help much. Reading the TransKing help file in order to see if it is a reoccurring error, even searching the web and on TransKings webpage with no result.

With the installation of VS6 the Microsoft upgrade wizard should work and hopefully produce some better code. The upgrade of VB6 to VB.NET worked out fine. See listing 3.

Sub Villdu()
    Dim DgDef As Object
    Dim Msg As Object
    Const IDYes As Short = 6
    Const IDNo As Short = 7
    Const mnuIconQuestion As Short =32
    Const mnuYesNO As Short = 4:Svar=0
    MsgBox(Msg + "Vill du spara innan du avslutar.",
    DgDef = mnuIconQuestion + mnuYesNO;
    Svar = MsgBox(Msg, DgDef, "Spara...");
End Sub

Listing 3. The upgraded code of VB6 to VB.NET.

Compared with Listing 1 and 2 code snippets the code in Listing 3 is much better structured and better formatted. The VB.NET wizard has done a really good job of fixing the code and uses proper indentation which increases the readability. It still has problem with the naming, global variables and GOTO
statements. The upgrade of the code also produced
issues such as error and warnings. A full report of the
different issues of the upgraded code is found in
appendix B.
This line-for-line approach with BabbelFisk would
maybe work out better if the code used in the
translation been well structured and been using a
naming and coding convention. This legacy
application has no such thing, and the line-for-line
translator gives us the same messy code in C# and
makes us not nearer a better and clearer source code.
On the other hand Microsoft VS.NET built-in
upgrade wizard compensates the lack of any coding
standard and indentation, by adding indentation and
code structure to the upgraded solution of StålSpec
2000 thus making it easier to follow.

12. Structure improvement

The structure improvements that have been done are
the identification of the GOTO codes in the StålSpec
source and restructure.
Not all the GOTO statements have been
restructured only the parts that have functions needed
in the new application. For example as we are using
the old program as the requirement specification with
some exceptions (see section 8 requirement
specification) we know functionality needed in the
new StålSpec.
With the new upgraded code to VB.NET, select the
GUI part to be used. Right click and choose “View
Code” and it bring you the code behind the button.
This approach can also be done with VS6 but with the
upgraded code VS.NET is the natural choice.
To illustrate the approach a sample piece of code is
use to show how we improved the code and made the
intended functionality more visible. As the code in
Listing 4 there is a need of extra thought and time to
dig in to the meaning and intention of the code.
The intended function of this subroutine is to save
updated prices to a file.
To begin with: if an error occurs GoTo
Errohandling show a messagebox with error
message close the file an GoTo back and exit the
subroutine. If there are no errors open the file, receive
the new values and save the file.

Sub Saveprice()
On Error GoTo Errorhandling
FileOpen(1, Filnamn,
OpenMode.Output)
For i = 0 To 7
   WriteLine(1, Kdat(i).Text)
Next
PrintLine(1, Combo2.Text)
PrintLine(1, Knappm)
z = 1 : Savedata()
If sweval(Combo2.Text) > 0 Then
   For z = 2 To
      sweval(Combo2.Text)+1
      PrintLine(1, Mtyp(z))
      Savedata()
   Next z
End If
WriteLine(1, Text4.Text)
WriteLine(1, Text6.Text)
FileClose()
Rev = 0
back:
Exit Sub
Errorhandling:
   MsgBox(ErrToString(Err.Number),
   48, "A error occurred...")
   FileClose()
Resume back
End Sub

Listing 4. Subroutine Save price

Public void Saveprice()
{
    //Try to open database.
    If error
       Show error message
    else
       fill dataset with values
       from pricelist.
       close databaseconnection
    compare dataset values with GUI
    values.
    If gui value different
       check new values for error
    if error
       show error message
    else
       update dataset with the new
       values
    Try to open database
    If error
       Show error message.
    else
       update database with new
       values from dataset to
       table pricelist
}

Listing 5. Psuedocode of code in Listing 4
This is not good coding practice with GoTo and cryptic names like Knappm and sweval. The general functionality is clearly visible and fundamental understanding of the code is not necessary. But experience of programming and software development is needed to understand the outlines of the code.

In order to make it work in the new StålSpec, modifications are needed but to begin with we are using words to support and describe the function before doing any actual code. This is called pseudocode it is a way of describe algorithm or functions.[22] See Listing 5. We are using a database instead of a text files for saving prices.

Now the GoTo statements are gone simply by using if else statements instead of the tangled way used previously. Most of the process is not as detailed as the example above. A much wider approach is used with fewer words describing the functions. In some cases pseudocode is not used at all depending on the size of the function or if a similar function has been done earlier which could be reused again.

13. User based recovery

A different approach used is a more user-centered way of recover design and functions. By actually run the program StålSpec 2000 and generate code from it to the same thing.

The most basic things were easily recovered and code generated to do the same thing. For example when a button is clicked and it calls upon another form to open. More advanced features require a combination of line-for-line conversion, structure improvements and manual effort to recover useful code if any code.

14. Coding practice

Coding practices used is Pascal Casing and Camel Casing both are specified in the two guides, C# Coding Style Guide [23] and C# Coding Standards and Best Programming Practices [24]

- Pascal Casing - First character of all words are Upper Case and other characters are lower case.
- Camel Casing - First character of all words, except the first word are Upper Case and other characters are lower case.

What makes this such a good coding practice to adopt? It brings unity to the code and distinction.

Camel parsing is used for method parameters and variables and Pascal casing for classes and method names. Meaningful and descriptive words to name the different variables are used instead of x, y or z for use as counter in while and for loops. We use variable names as arrayLengthCounter or other descriptive names, single char variables are never used. The file name always matches the classname and namespaces follow the standard pattern as well.

Indentation is made with tabs not with space, because some might like four spaces, other like two, with tab everyone can set its own indentation level.

For commenting the code we use our common sense and the guides C# Coding Style Guide [23] and C# Coding Standards and Best Programming Practices [24]. Comments in the code should express the functionality and properly state the function of the different classes. Comments should not extensively explain every little detail, just enough to explain the functionality. The comment should not either exceed the length of the code it suppose to explain by too much. This is an indication of too complicated, potential buggy code which maybe should be rewritten.

By using this practice other developer who maybe not uses this practice can easily see how the code works, with some manual effort of reading the guide of course. It also as said before brings unity to the code, using the coding practice in a development team who work with each others code. No time is needed to get acquainted with the other team member’s code.

15. Result

StålSpec 2000 contains a vast amount of functions and cryptic variables, typical for a legacy application. As time passes by new software becomes old software, most of them are still maintained because they work and are a vital part of an organisation. StålSpec is also vital to MVR as a corner stone in their organisation as a one of a kind application custom built for the steel and construction industry.

Early on we realised that the time needed and the time available to finish the project StålSpec 2000 did not even out. The code contains vast amount of functions that we never would recreate in time. We had to narrow it down and focus on the main functionality and more or less go back to basics. Meetings with MVR [4] and the reference group ended up in a minimum specification with only the main functionality in focus. MVR and the reference group knew from the beginning that we would never finish in time.
The amount of time we spent on the development of StålSpec 3000 is about 900 hours divided between three persons; the rest of the time was used in writing this thesis. StålSpec is such a custom application that people like ourselves have little understanding of the technical language and jargon used in the steel industry. Most of the time was spent on analyzing the old StålSpec 2000, studying the specific industry terms used within, and in to every meeting we had questions to the reference group about StålSpec 2000 and other related issues.

During the development of SS3K we have been using a coding and naming practice specified in the guide C# Coding Style Guide [23] and C# Coding Standards and Best Programming Practices [24] which also give examples of good coding practice. As discussed in section 14 Coding practice.

The easiest task during the development phase was the GUI; by using the StålSpec 2000 as template and copying the layout to the new SS3K. The hard part was the code behind it all. Analyzing the old code and learning how to work with the StålSpec 2000 consumed most of the time. In programming the new SS3K scarce functionality was added such as calls for opening different GUIs similar to the GUI of StålSpec 2000. In the StålSpec 2000 text files were used to store information, we instead created a database file in Microsoft Access to do the same. Cryptic names used when storing information in the text files used by StålSpec 2000 slowed down the process of recreating it to a database. We manage to make the database work with some tables. But the goal was to make it work, to get values from the database, to alter the values, add rows get the general functionality of a database. The database was implemented in the function SupplierFiles which is used as an address book with addresses, phone numbers etc stored in the database. By only implement the database and make it fully work in one function it is easier to expand it to include other parts of the project, instead of adding it all over and not make work at all.

The time needed and the time available was the contributing factor why not more were made to the SS3K application.

The new version of StålSpec has a lot more work to be done before it is ready to be released but the achievements so far have only got positive feedback from MVR.

"It is a shame StålSpec 2000 was so poorly designed that you have not been able to reach further with your work. What you have done looks great and we are hoping you are willing to continue the development even after your exam work, but I do not believe this new version of StålSpec is ready yet to meet the public eye. It would have been exciting to be able to bring a demonstration of the new, functional StålSpec to our congress. Oh well, you will impress the audience on our next congress instead." [25]

The intention is to continue working with the StålSpec and hopefully in the future release a new version to the reference group for testing before the public release.

16. Conclusion

When using software re-engineering to create more manageable code, automated tools are of great help in converting and standardizing code. Microsoft upgrade wizard did a good job of standardize the code and cleaning cosmetic things like indentations of the legacy source code. The upgrade wizard took some time to finish the task but the result was better then expected.

Structure improvements were done to parts of the code that contained tangled structures as GOTO statements. Only parts useful for development of SS3K were re-structured and re-written with psuedocode to make the implementation easier.

The task at hand was to make StålSpec 2000 better, remove bugs and other related issues which make StålSpec 2000 malfunction during certain circumstances. MVR, the reference group and the project team, we all agreed that to ensure easier maintenance in the future the program has to be changed to meet the new needs. SS3K has to be object oriented and in a modern language. To fully take advantage of the new features and structures the new VS.NET environment has to offer the application has to be rewritten.

The process of rewriting obsolete software systems is the most costly approach and the most risky. But it eliminates the legacy application in which code and structures are of such a poor quality that migration is not an option. The new software may not even function exactly as the old one. The new software could even contain new malfunctions, bugs and other issues.

We use the old StålSpec 2000 GUI and its programs source code as the requirement specification with some exceptions of functionality which is left out. The focus is “back to basics”, what are the functions that most people use. Recover how they work and implement them in StålSpec 3000. Well aware of the risk of rewriting.

For Companies looking to preserve and extend the functionality of their legacy system the re-engineering is the best approach. But where the goal is to produce a
similar system in an OO manner, parts from reengineering, combined with use based recovery can aid the new development, as has been shown in the case of SS3K.

The most suitable approach to do just that is a combination of use based recovery and the reengineering principles of line-for-line translation and structure improvements for recovering design and functionality. This is not a silver bullet with the solution of the problem with legacy systems. But in this case it certainly helped ease the somewhat overwhelming task StålSpec 2000 gave. A considerable amount of time and manual work is needed to produce working code, as showed in the earlier sections 11 line-for-line translation, 12 structure improvements and 13 User based recovery. These approaches do not provide all the answers needed to fully re-write a legacy application. They work as guidance giving the presumable functionality, then it up to us to make it work.

17. Future Work

The .NET Framework offers a number of benefits; for starters it offers an easy migration path to a web based application that might be needed in the future. It could end up as a client-server system with most of the computer intensive functions on a powerful server, and the customers with a relatively light client to access the system. Updates to the system are made to the server and the clients are automatically updated the next time they access the server.

To create such a client-server system, ASP.NET would be a suitable choice because of its integration with VS.NET providing a GUI designer, a toolbox and a fully integrated debugger. Allowing development of application in a What You See is What You Get (WYSIWYG) manner. [7, 9]

18. References


Appendix

A. Comment details in files used in StålSpec 2000.

A.1. The form frmSpecin

Contains 5719 rows of code, 319 comments are auto generated by Visual Studio 6. On 16 occasions comments is used to exclude lines from the project when debugged. 4 real comments in the source code that give useful information about the subroutine used

A.2. The form frmStart

Contains 13413 rows of code, 831 comments are auto generated by Visual Studio 6. On 88 occasions comments are used to exclude lines from the project when debugged. 5 real comments in the source code that give useful information about the subroutine used

A.3. The module file Module11

Contains 5646 rows of code, three are auto generated by Visual Studio 6. On 21 occasions comments are used to exclude lines from the project when debugged. 5 comments are “note to self” of row deletion. 6 comments states the value of PI.

A.4. The form frmNypa

Contains 1610 rows of code, 52 comments are auto generated by Visual Studio 6. On 4 occasions comments are used to exclude lines from the project when debugged.

A.5. The form frmOpp5

Contains 2038 rows of code, 32 comments are auto generated by Visual Studio 6. On 9 occasions comments are used to exclude lines from the project when debugged.

A.6. The form frmOpp7

Contains 4321 rows of code, 31 comments are auto generated by Visual Studio 6. On 3 occasions comments are used to exclude lines from the project when debugged.

A.7. The form frmRes2

Contains 7499 rows of code, 529 comments are auto generated by Visual Studio 6. On 37 occasions comments are used to exclude lines from the project when debugged. 1 comment is “note to self” of row deletion 1 comment shows the start of a subroutine.

A.8. The form frmApris

Contains 2024 rows of code, 187 comments are auto generated by Visual Studio 6. On 17 occasions comments are used to exclude lines from the project when debugged. 1 comment shows the start of a subroutine.

A.9. The form frmEgen2

Contains 1643 rows of code, 87 comments are auto generated by Visual Studio 6. On 9 occasions comments are used to exclude lines from the project when debugged.
A.10. The form frmGranska

Contains 372 rows of code, 19 comments are auto generated by Visual Studio 6. 1 comment shows the start of a subroutine.

A.11. The form frmKappris

Contains 2250 rows of code, 206 comments are auto generated by Visual Studio 6. On 9 occasions comments are used to exclude lines from the project when debugged
B. Problem report by description - StålSpec2000 New Source Code

Form frmApris

Constant available

A constant is available in place of a function call. Use a string constant instead of Chr/ChrW. The available string constants and their ASCII values are: vbNullChar (0), vbBack (8), vbTab (9), vbLf (10), vbVerticalTab (11), vbFormFeed (12), vbCr (13), vbCrLf or vbCrLf (13 & 10). vbCrLf is faster than vbCrLf. Successive Chr(13) & Chr(10) should be replaced by vbCrLf, not vbCrLf & vbCrLf. Instead of a call such as Asc("A"), use a numeric constant such as Const ascA = 65. These rules apply to VB 4-6. In VB .NET the compiler takes care of optimizing the use of these functions. Optimization. Severity: Info.

Count: 3.

4 Sub frmApris.Visa
   vbTab

4 Sub frmApris.Läsainfil
   vbLf

4 Sub frmApris.Läsainfil
   vbCr

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning.

Count: 2.

frmApris
TextBox txtEdit

frmApris
PictureBox ÖppnaSpara

Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning.

Count: 1.

Sub frmApris.Sfram10
**Fixed file number found**

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 17.

```vba
21 Sub frmApris.Form_Activate
28 Sub frmApris.Form_Activate
36 Sub frmApris.Form_Activate
43 Sub frmApris.Form_Activate
 3 Sub frmApris. Hämtaapriss
 3 Sub frmApris. Sparaapriss
15 Sub frmApris. Skärmspec
36 Sub frmApris. Skärmspec
 2 Sub frmApris. Lasainfil
18 Sub frmApris. Pris88
43 Sub frmApris. Pris88
22 Sub frmApris. Pris109
46 Sub frmApris. Pris109
 2 Sub frmApris. Sparaskruv
 8 Sub frmApris. Sparaskruv
14 Sub frmApris. Sparaskruv
20 Sub frmApris. Sparaskruv
```

**Goto statement found**

Use of Goto is bad programming practice and should be avoided when possible. Jumping around with Goto easily leads to unstructured control flow structure and spaghetti code. Style. Severity: Info. Count: 2.

```vba
28 Sub frmApris. Koll8
12 Sub frmApris. Koll9
```

**Hotkey missing**


```vba
frmApris
CommandButton Command1
 "Pris skruv 10.9"

frmApris
Frame Frame4
 "PRISLISTA STÅLBYGNGADSSKRUV"
```

**Option Explicit missing**

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

```vba
frmApris
```
The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant available</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>17</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>2</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>2</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

Type               Count
Optimization        6 ****
Style               20 **************
Functionality       3 **
Total               29
Problems/logical code line 0,03

Form frmegen2

Click event missing


Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If ..And.. condition as two nested Ifs. Short-circuiting If ..Or.. yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/ AndAlso and Or/OrElse. Optimization. Severity: Info. Count: 1.
**Constant without type specification**

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants.


```
11 Sub frmegen2.MB_Bortkval_Click
   MB_IconQuestion
11 Sub frmegen2.MB_Bortkval_Click
   IDYes
11 Sub frmegen2.MB_Bortkval_Click
   IDNo
```

**Control not enabled**

A control’s Enabled property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are never enabled. You should consider removing disabled controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control’s properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where a disabled control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. Applies to VB 3-6. Optimization. Severity: Warning. Count: 2.

```
frmegen2
   Menu mnuArtikel
frmegen2
   Menu mnuRevArt
```

**Control not visible**

A control’s Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control’s properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

```
frmegen2
   Menu mnuRevArt
```

**Dead constant**

A variable or constant is not used. You may remove it if you are sure you won’t need it later. The removal doesn’t affect the functionality of your program.


```
11 Sub frmegen2.MB_Bortkval_Click
   IDNo
```
Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program.

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 14.

Goto statement found

Use of Goto is bad programming practice and should be avoided when possible. Jumping around with Goto easily leads to unstructured control flow structure and spaghetti code. Style. Severity: Info. Count: 5.

Hotkey conflict

Two or more controls or menu items share a hotkey. For example, there are options &Save and &Search in the same menu. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.
Hotkey missing


```vbnet
frmegen2
Menu Kr(2)
"Prisjustering i procent"
```

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

```vbnet
frmegen2
```

Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

```vbnet
frmegen2
Picture3 doesn't precede children, txtEdit->CoolBar1
```

Unicode function is faster

The wide functions AscW and ChrW/ChrW$ are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you're handling ASCII characters from 0 to 127, you're safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 4.

```vbnet
14 Sub frmegen2.Text1_KeyPress
    AscW
```

```vbnet
14 Sub frmegen2.Text1_KeyPress
    ChrW$
```
Variable without type specification

A variable does not have a defined data type. By default, the type is Variant. Variant needs more memory than other types. Decide what type you need and write it to the variable declaration. Besides, upgrading to VB.NET will be easier if you use explicit data types. Fix recommended before upgrade. Optimization. Severity: Warning. Count: 1.

```
frmgen2.(declarations)
Maxrad
```

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>2</td>
</tr>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Control not enabled</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead variable/parameter</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>14</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>5</td>
</tr>
<tr>
<td>Hotkey conflict</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>2</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Unicode function is faster</td>
<td>Optim.</td>
<td>4</td>
</tr>
<tr>
<td>Variable without type specification</td>
<td>Optim.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>15</td>
</tr>
<tr>
<td>Style</td>
<td>20</td>
</tr>
<tr>
<td>Functionality</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td>41</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,04</td>
</tr>
</tbody>
</table>

Form frmGranska

Click event missing


```
frmGranska
Menu MB_Spara

frmGranska
Menu MB_Skrivarval
```
Constant available

A constant is available in place of a function call. Use a string constant instead of Chr/ChrW. The available string constants and their ASCII values are: vbNullChar (0), vbBack (8), vbTab (9), vbLf (10), vbVerticalTab (11), vbFormFeed (12), vbCr (13), vbCrLf or vbNewline (13 & 10). vbNewline is faster than vbCrLf. Successive Chr(13) & Chr(10) should be replaced by vbNewline, not vbCr & vbLf. - Instead of a call such as Asc("A"), use a numeric constant such as Const ascA = 65. - These rules apply to VB 4-6. In VB.NET the compiler takes care of optimizing the use of these functions. Optimization. Severity: Info. Count: 2.

4 Sub frmGranska.Läsainfil
   vbCr
4 Sub frmGranska.Läsainfil
   vbLf

Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won’t need it later. The removal doesn’t affect the functionality of your program. - Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 1.

Sub frmGranska.Huvuda

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 3.

15 Sub frmGranska.Skärmspec
36 Sub frmGranska.Skärmspec
 2 Sub frmGranska.Läsainfil

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

frmGranska
Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

frmGranska
Command4->CoolBar1

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don’t respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

frmGranska

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>5</td>
</tr>
<tr>
<td>Constant available</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>3</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

Type                Count
Optimization        3 ****
Style               4 *****
Functionality       7 *********
Total               14
Problems/logical code line 0,12

Form frmKappris

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 16.

6 Sub frmKappris.Sparak
13 Sub frmKappris.Sparak
34 Sub frmKappris.Sparak
44 Sub frmKappris.Sparak
Hotkey missing


frmKappris
Frame Frame10
"KOSTNAD FÖR GASSKÄRNING, kr/skärmeter"

frmKappris
Frame Frame5
"STÄLLKOSTNAD, Kr PER DIMENSION"

frmKappris
Frame Frame7
"Kostnad för bockning och vändning av plåt, kr/bock"

frmKappris
Frame Frame3(0)
"MÅNGDFAKTOR, SUMMA YTA PER ARTIKELRAD"

frmKappris
Frame Frame1(0)
"GRUNDPRIS, kr/m²"

frmKappris
Frame Frame2(1)
"PRISFAKTOR"

frmKappris
Frame Frame3(1)
"MATERIALFAKTOR"

frmKappris
Frame Frame6
"LÄNGDFAKTOR"

frmKappris
Frame Frame1(1)
"PRISER FÖR KAPNING AV BALK OCH PROFILER I HELA KRONOR"

frmKappris
Frame Frame3(3)
Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>16</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>12</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

Type                Count
Style               17  ***********
Functionality       13  ***********
Total               30
Problems/logical code line 0,04

Form frmNypa

Click event missing

Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If ...And.. condition as two nested Ifs. Short-circuiting If ..Or.. yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/AndAlso and Or/OrElse. Optimization. Severity: Info. Count: 2.

42 Sub frmNypa.Strip_Click
43 Sub frmNypa.Strip_Click

Constant available

A constant is available in place of a function call. Use a string constant instead of Chr/ChrW. The available string constants and their ASCII values are: vbNullChar (0), vbBack (8), vbTab (9), vbCrLf (10), vbVerticalTab (11), vbFormFeed (12), vbCr (13), vbCrLf or vbCrLf (13 & 10). vbCrLf is faster than vbCrLf. Successive Chr(13) & Chr(10) should be replaced by vbCrLf, not vbCrLf & vbCrLf. - Instead of a call such as Asc("A"), use a numeric constant such as Const ascA = 65. - These rules apply to VB 4-6. In VB.NET the compiler takes care of optimizing the use of these functions. Optimization. Severity: Info. Count: 1.

11 Sub frmNypa.Läsainpris
   vbTab

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 3.

frmNypa
Label Label10
frmNypa
TextBox txtEdit
frmNypa
Label Label9

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 6.

Hotkey conflict
Two or more controls or menu items share a hotkey. For example, there are options &Save and &Search in the same menu. Applies to VB 3–6. Functionality. Severity: Warning. Count: 1.

Hotkey missing

Option Explicit missing
A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.
Resizable Form missing Form_Resize

The Form.Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

frmNypa

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Constant available</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>6</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Hotkey conflict</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>3</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>6 *****</td>
</tr>
<tr>
<td>Style</td>
<td>8 *****</td>
</tr>
<tr>
<td>Functionality</td>
<td>6 *****</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,02</td>
</tr>
</tbody>
</table>

Form frmOpp5

Click event missing


frmOpp5
CommandButton Command5

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

frmOpp5
PictureBox Picture2
Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. - Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 3.

Sub frmOpp5.spillb
Sub frmOpp5.visamer
Sub frmOpp5.Lulle

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 5.

11 Sub frmOpp5.Hämta
13 Sub frmOpp5.Hämta
6 Sub frmOpp5.Form Activate
2 Sub frmOpp5.Hämtakapning
7 Sub frmOpp5.Hämtakapning

Goto statement found

Use of Goto is bad programming practice and should be avoided when possible. Jumping around with Goto easily leads to unstructured control flow structure and spaghetti code. Style. Severity: Info. Count: 2.

20 Sub frmOpp5.Hämta
38 Sub frmOpp5.Hämta

Hotkey missing


frmOpp5
CommandButton Command5
"Flytta"

frmOpp5
CommandButton Command4
"OPTIMERA"

frmOpp5
CheckBox Check2
"1 Längd/balk"

frmOpp5
CommandButton Command1
"EGET LAGER"

frmOpp5
CommandButton Command7
Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.
Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>5</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>2</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>9</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>4 ***</td>
</tr>
<tr>
<td>Style</td>
<td>8 *********</td>
</tr>
<tr>
<td>Functionality</td>
<td>12 *********</td>
</tr>
<tr>
<td>Total</td>
<td>24</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,02</td>
</tr>
</tbody>
</table>

Form frmOpp7

Hotkey missing


```vbnet
frmOpp7
CommandButton Command1
"FLER KAPFIGURER"
```

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

```vbnet
frmOpp7
```

Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time

```
frmOpp7
Picture2(0) doesn't precede children
```

**Resizable Form missing Form_Resize**

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

```
frmOpp7
```

**Problem summary**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style</td>
<td>1         ****</td>
</tr>
<tr>
<td>Functionality</td>
<td>3         ***************</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
</tr>
</tbody>
</table>

Problems/logical code line 0,03

**Form frmRes2**

**Click event missing**


```
frmRes2
CommandButton Command1(1)
```

**Consider using Image control**

A PictureBox uses significantly more memory and system resources than an Image control. While PictureBox has more features, this PictureBox is a potential candidate to be replaced by an Image control. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

```
frmRes2
PictureBox Picture2
```

**Constant available**

A constant is available in place of a function call. Use a string constant instead of Chr/ChrW. The available string constants and their ASCII values are: vbNullChar (0), vbBack (8), vbTab (9), vbLf (10), vbVerticalTab (11), vbFormFeed (12), vbCr (13), vbCrLf or vbNewline (13 & 10). vbNewline is faster than vbCrLf. Successive Chr(13) & Chr(10) should be replaced by vbCrLf, not vbCrLf & vbCrLf. - Instead of a call such as Asc("A"), use a numeric constant such as Const ascA = 65. - These rules apply to VB 4-6. In VB.NET the compiler takes
The care of optimizing the use of these functions. Optimization. Severity: Info. Count: 44.

9 Sub frmRes2.Visa
  vbTab

21 Sub frmRes2.Visa
  vbTab

23 Sub frmRes2.Visa
  vbTab

28 Sub frmRes2.Visa
  vbTab

31 Sub frmRes2.Visa
  vbTab

33 Sub frmRes2.Visa
  vbTab

37 Sub frmRes2.Visa
  vbTab

39 Sub frmRes2.Visa
  vbTab

40 Sub frmRes2.Visa
  vbTab

48 Sub frmRes2.Visa
  vbTab

49 Sub frmRes2.Visa
  vbTab

50 Sub frmRes2.Visa
  vbTab

51 Sub frmRes2.Visa
  vbTab

11 Sub frmRes2.Visam
  vbTab

21 Sub frmRes2.Visam
  vbTab

21 Sub frmRes2.Visam
  vbTab

21 Sub frmRes2.Visam
  vbTab
21 Sub frmRes2.Visam
    vbCrLf
21 Sub frmRes2.Visam
    vbCrLf
21 Sub frmRes2.Visam
    vbCrLf
21 Sub frmRes2.Visam
    vbCrLf
21 Sub frmRes2.Visam
    vbCrLf
21 Sub frmRes2.Visam
    vbCrLf
27 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
33 Sub frmRes2.TabStrip1_Click
    vbCrLf
87 Sub frmRes2.TabStrip1_Click
    vbCrLf
91 Sub frmRes2.TabStrip1_Click
    vbCrLf
91 Sub frmRes2.TabStrip1_Click
    vbCrLf
92 Sub frmRes2.TabStrip1_Click
    vbCrLf
92 Sub frmRes2.TabStrip1_Click
    vbCrLf
93 Sub frmRes2.TabStrip1_Click
vbTab
93 Sub frmRes2.TabStrip1_Click
vbTab
94 Sub frmRes2.TabStrip1_Click
vbTab
94 Sub frmRes2.TabStrip1_Click
vbTab
9 Sub frmRes2.visaegna
vbTab
12 Sub frmRes2.visaegna
vbTab
13 Sub frmRes2.visaegna
vbTab

**Constant without type specification**

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants.


2 Sub frmRes2.Lagersaldo
   IDNo
2 Sub frmRes2.Lagersaldo
   mnuIconQuestion
2 Sub frmRes2.Lagersaldo
   IDYes

**Control not visible**

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 2.

frmRes2
PictureBox Picture6
frmRes2
TextBox txtedit
Dead constant

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 1.

2 Sub frmRes2.Lagersaldo
IDNo

Sub frmRes2.KrProcent

Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. - Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 1.

2 Sub frmRes2.Lagersaldo
IDNo

Sub frmRes2.KrProcent

Dead procedure/declaration/event (called by dead only)

A procedure is not executed and it is effectively dead. Calls to the procedure exist, but the callers don't execute. You should remove this procedure along with its callers, provided that you are sure you won't need any of the callers later. Optimization. Severity: Warning. Count: 1.

Sub frmRes2.KrProcent

Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 3.

frmRes2.(declarations)
Ställ

frmRes2.(declarations)
Kpos

4 frmRes2.(declarations)
Krabb

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 32.

12 Sub frmRes2.Bocka
25 Sub frmRes2.FBMåla
13 Sub frmRes2.Grundpris
26 Sub frmRes2.Grundpris
38 Sub frmRes2.Grundpris
50 Sub frmRes2.Grundpris
103 Sub frmRes2.Grundpris
Goto statement found


Hotkey missing

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

Variable without type specification

A variable does not have a defined data type. By default, the type is Variant. Variant needs more memory than other types. Decide what type you need and write it to the variable declaration. Besides, upgrading to VB.NET will be easier if you use explicit data types. Fix recommended before upgrade. Optimization. Severity: Warning. Count: 11.
Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Consider using Image control</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Constant available</td>
<td>Optim.</td>
<td>44</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead procedure/declaration/event (called by dead only)</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead variable/parameter</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>32</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>6</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Variable without type specification</td>
<td>Optim.</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>67 ************</td>
</tr>
<tr>
<td>Style</td>
<td>34 ****</td>
</tr>
<tr>
<td>Functionality</td>
<td>8 *</td>
</tr>
<tr>
<td>Total</td>
<td>109</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,03</td>
</tr>
</tbody>
</table>

Form frmSpecin

Click event missing


```vbnet
frmSpecin
CommandButton Command3(2)

frmSpecin
CommandButton Command1(0)

frmSpecin
CommandButton Command6

frmSpecin
CommandButton Knapp(14)
```
Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If ..And.. condition as two nested Ifs. Short-circuiting If ..Or.. yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/ AndAlso and Or/OrElse. Optimization. Severity: Info. Count: 1.

3 Sub frmSpecin.pGrid_MouseUp

Consider using Image control

A PictureBox uses significantly more memory and system resources than an Image control. While PictureBox has more features, this PictureBox is a potential candidate to be replaced by an Image control. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

frmSpecin
PictureBox Picture5

Constant without type specification

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants. Optimization. Severity: Info. Count: 3.

2 Sub frmSpecin.Villdukval
IDNo

2 Sub frmSpecin.Villdukval
mnuIconQuestion

2 Sub frmSpecin.Villdukval
IDYes

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or

frmSpecin
PictureBox Picture6

frmSpecin
FileList Box File1

Dead constant
A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 1.

2 Sub frmSpecin.Villdukval
   IDNo

Dead procedure/declaration/event
A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. - Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 5.

Sub frmSpecin.totviktyta
Sub frmSpecin.Visaextra
Sub frmSpecin.Radkoll
Sub frmSpecin.Rad572
Sub frmSpecin.Skruvar

Dead variable/parameter
A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 2.

2 frmSpecin.(declarations)
   KKK

3 frmSpecin.(declarations)
   Mark

Fixed file number found
Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 23.

65 Sub frmSpecin.viktber
67 Sub frmSpecin.viktber
16 Sub frmSpecin.Visalev
12 Sub frmSpecin.Visaval
23 Sub frmSpecin.Command11_Click
29 Sub frmSpecin.Command11_Click
14 Sub frmSpecin.Command2_Click
34 Sub frmSpecin.Command2_Click
4 Sub frmSpecin.Command4_MouseDown
18 Sub frmSpecin.Command9_Click
35 Sub frmSpecin.Command9_Click
11 Sub frmSpecin.Form_Activate
77 Sub frmSpecin.Grid_MouseUp
91 Sub frmSpecin.Grid_MouseUp
7 Sub frmSpecin.mnuBortModul_Click
29 Sub frmSpecin.mnuSorterar_Click
34 Sub frmSpecin.mnuSorterar_Click
5 Sub frmSpecin.mnuSäkersaldo_Click
8 Sub frmSpecin.mnuÅtersaldo_Click
148 Sub frmSpecin.txtedit_KeyPress
19 Sub frmSpecin.Exx
9 Sub frmSpecin.Moduler
4 Sub frmSpecin.Visanr

Goto statement found


58 Sub frmSpecin.Kollktyp
61 Sub frmSpecin.Kollktyp
6 Sub frmSpecin.Grid_KeyDown
15 Sub frmSpecin.Grid_KeyDown

Hotkey conflict

Two or more controls or menu items share a hotkey. For example, there are options &Save and &Search in the same menu. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

frmSpecin
Conflict in frmSpecin
SO: "Ta bort m&odul" "&Sortera" "&Säkerhetskopie" "Kap&optimera"

Hotkey missing


frmSpecin
CommandButton Command8(1)
"mm - grader"

frmSpecin
CommandButton Knapp(2)
"1600"

frmSpecin
CommandButton Knapp(1)
"1234567890123"

frmSpecin
CommandButton Knapp(7)
"PLÅT"

frmSpecin CommandButton Command5 "AVSYNING"

frmSpecin CommandButton Command12(1) "Rostfritt"

frmSpecin CommandButton Knapp(3) "IPE"

frmSpecin CommandButton Command3(1) "EGNA PRODUKTER"

frmSpecin CommandButton Command3(2) "EGNA PROFILER"

frmSpecin CommandButton Command11 "Uppdatera lager"

frmSpecin CommandButton Knapp(6) "KKR"

frmSpecin CommandButton Command7 "Tillbaka"

frmSpecin CommandButton Knapp(4) "UPE"

frmSpecin CommandButton Knapp(5) "VKR"

frmSpecin Menu mnuSök(4) "Ta bort sökmarkeringen"

frmSpecin CommandButton Command12(2) "Aluminium"

frmSpecin Menu mnuSök(2) "Profil och dimension"

frmSpecin Menu mnuSök(0) "Littera"
frmSpecin
Menu mnuSök(1)
"Profil"

frmSpecin
Menu mnuSöka(0)
"Sök"

frmSpecin
Frame Frame4
"SKRUVLÄNGDER VID GIVEN KLÄMLÄNGD"

frmSpecin
CommandButton Command12(0)
"Stål"

frmSpecin
CommandButton Command2
"SKRUV 8.8"

frmSpecin
CommandButton Command9
"SKRUV 10.9"

frmSpecin
CommandButton Knapp(14)
"1234567890123"

frmSpecin
CommandButton Knapp(12)
"1234567890123"

frmSpecin
CommandButton Knapp(8)
"PLS"

frmSpecin
CommandButton Knapp(13)
"1234567890123"

frmSpecin
CommandButton Command4
"LEVERANTÖRER"

frmSpecin
CommandButton Command6
"EXTRA"

frmSpecin
CommandButton Command8(0)
"Grader - mm"

frmSpecin
CommandButton Command20
"Avbryt"
frmSpecin
CommandButton Knapp(11)
"1234567890123"

frmSpecin
Menu mnuSök(3)
"Profil, dimension och stålsort"

frmSpecin
CommandButton Knapp(9)
"UNI"

frmSpecin
CommandButton Knapp(10)
"1234567890123"

frmSpecin
CheckBox Check1(2)
"Klippling"

frmSpecin
Frame Frame3
"Plåtbearbetning"

frmSpecin
CheckBox Check1(1)
"Skärning"

frmSpecin
Frame Frame2
"Sök profil"

frmSpecin
Frame Frame1
"Bockning"

frmSpecin
CommandButton Command1(0)
"ROSTFRITT"

frmSpecin
CheckBox Check1(3)
"Nibbling"

frmSpecin
CommandButton Command15
"Visa allt"

frmSpecin
CommandButton Command1(1)
"ALUMINIUM"

frmSpecin
Menu BYT
"Byt profil"
Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

```
frmSpecin
```

Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

```
frmSpecin
Picture8->Picture7, Iinfo(3)->Iinfo(2), Picture7->Picture5, ...
```

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

```
frmSpecin
```

Unicode function is faster

The wide functions AscW and ChrW/ChrW$ are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you’re handling ASCII characters from 0 to 127, you’re safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 2.

```
3 Sub frmSpecin.txtedit_KeyPress
AscW

3 Sub frmSpecin.txtedit_KeyPress
ChrW$ 
```

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>6</td>
</tr>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>1</td>
</tr>
</tbody>
</table>
Consider using Image control
Constant without type specification
Control not visible
Dead constant
Dead procedure/declaration/event
Dead variable/parameter
Fixed file number found
Goto statement found
Hotkey conflict
Hotkey missing
Option Explicit missing
Possibly twisted tab order
Resizable Form missing Form_Resize
Unicode function is faster

Type | Count
--- | ---
Optimization | 17 ***
Style | 28 *****
Functionality | 55 ***********
Total | 100
Problems/logical code line | 0,02

Form frmStart

Click event missing


```vbnet
frmStart
CommandButton Command1(0)

frmStart
CommandButton Command14(13)

frmStart
Menu mnuBortkval

frmStart
CommandButton Commandb1
```

Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If...And... condition as two nested Ifs. Short-circuiting If...Or... yields more complex code case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/ AndAlso and Or/ OrElse. Optimization. Severity: Info. Count: 4.

```vbnet
26 Sub frmStart.Command16_Click
100 Sub frmStart.Sparaindata
31 Sub frmStart.Bakåt
```
Consider using Image control

A PictureBox uses significantly more memory and system resources than an Image control. While PictureBox has more features, this PictureBox is a potential candidate to be replaced by an Image control. Applies to VB 3-6. Optimization.


```
frmStart
PictureBox Picture8
```

Constant available

A constant is available in place of a function call. Use a string constant instead of Chr/ChrW. The available string constants and their ASCII values are: vbNullChar (0), vbBack (8), vbTab (9), vbLf (10), vbVerticalTab (11), vbFormFeed (12), vbCr (13), vbCrLf or vbNewline (13 & 10). vbCrLf is faster than vbCrLf. Successive Chr(13) & Chr(10) should be replaced by vbCrLf, not vbCrLf or vbCrLf. - Instead of a call such as Asc("A"), use a numeric constant such as Const ascA = 65. - These rules apply to VB 4-6. In VB.NET the compiler takes care of optimizing the use of these functions. Optimization. Severity: Info. Count: 8.

```
31 Sub frmStart.TabStrip1_MouseUp
   vbCrLf
64 Sub frmStart.TabStrip1_MouseUp
   vbCrLf
125 Sub frmStart.TabStrip1_MouseUp
   vbCrLf
190 Sub frmStart.TabStrip1_MouseUp
   vbCrLf
2 Sub frmStart.VisaRuta1
   vbCrLf
18 Sub frmStart.Visas
   vbCrLf
52 Sub frmStart.Visaregister
   vbCrLf
71 Sub frmStart.Visaregister
   vbCrLf
```

Constant without type specification

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants. Optimization. Severity: Info. Count: 8.

```
5 Sub frmStart.mnu_Bort_Click
   mnuYesNO
```
Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

frmStart
TextBox txtedit

Dead constant

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 4.
Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won’t need it later. The removal doesn’t affect the functionality of your program. Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 2.

Sub frmStart.Slutgross
Sub frmStart.LNysid

Empty procedure/module


Sub frmStart.Textb2_KeyDown
Sub frmStart.Textb2_LostFocus

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 79.

12 Sub frmStart.Command16_Click
2 Sub frmStart.Command17_Click
20 Sub frmStart.Command2_Click
87 Sub frmStart.Command2_Click
38 Sub frmStart.Grid4_DblClick
7 Sub frmStart.List1_MouseUp
31 Sub frmStart.mnuAvsluta_Click
387 Sub frmStart.Stree_NodeClick
405 Sub frmStart.Stree_NodeClick
43 Sub frmStart.TabStrip1_MouseUp
65 Sub frmStart.TabStrip1_MouseUp
75 Sub frmStart.TabStrip1_MouseUp
126 Sub frmStart.TabStrip1_MouseUp
127 Sub frmStart.TabStrip1_MouseUp
128 Sub frmStart.TabStrip1_MouseUp
129 Sub frmStart.TabStrip1_MouseUp
156 Sub frmStart.TabStrip1_MouseUp
166 Sub frmStart.TabStrip1_MouseUp
18 Sub frmStart.Text11_LostFocus
17 Sub frmStart.Toolbar3_ButtonClick
6 Sub frmStart.Sparaindata
11 Sub frmStart.Sparaindata
16 Sub frmStart.Sparaindata
12 Sub frmStart.Sparaindata
28 Sub frmStart.Sparaindata
34 Sub frmStart.Sparaindata
35 Sub frmStart.Sparaindata
36 Sub frmStart.Sparaindata
37 Sub frmStart.Sparaindata
49 Sub frmStart.Sparaindata
59 Sub frmStart.Sparaindata
Hotkey conflict

Two or more controls or menu items share a hotkey. For example, there are options &Save and &Search in the same menu. Applies to VB 3-6. Functionality.

```
frmStart
Conflict with controls
TR: "&Registrera" "Öppna &tom spec" "&Ta bort datafi" "Ta bort &rad"
```

Hotkey missing

A control or menu item is missing a hotkey. Checked controls: CommandButton, CheckBox, OptionButton, Frame, Menu. Applies to VB 3-6. Functionality.

```
frmStart
Frame Frame1(0)
"Litterring"

frmStart
CommandButton Prog(6)
"Rostfria produkter"

frmStart
CommandButton Prog(10)
"Rabatter"

frmStart
OptionButton Option5(0)
"

frmStart
CommandButton Prog(8)
"Fraktpriser"

frmStart
Frame Frame16
"Kopiera från"

frmStart
OptionButton Option4(0)
"

frmStart
CommandButton Command9
"Visa föregående"

frmStart
CheckBox Check1
"Kaplista"

frmStart
CommandButton Prog(4)
"Kallformade stålprofiler"
```
frmStart OptionButton KKap(3) "Manuell inmatning"

frmStart OptionButton KKap(5) "Kapning"

frmStart OptionButton KKap(4) "Ingen kapning"

frmStart CommandButton Prog(7) "Aluminiumprodukter"

frmStart CommandButton Prog(0) "Profilspecifika rabatter"

frmStart CommandButton Prog(11) "Uppdatera priser från grossist"

frmStart CommandButton Command18 "Avbryt"

frmStart Frame Frame14 "Standardknappar för inmatning av stålspec"

frmStart CommandButton Command16 "STARTA KOPIERINGEN"

frmStart CommandButton Command17 "Spara och avsluta"

frmStart Frame Frame17 "Kopiera till"

frmStart CheckBox Check6(0) "FB kapning"

frmStart CommandButton Prog(3) "Stålrör"

frmStart CommandButton Prog(5) "Stålplåt"
frmStart
Frame Frame19(2)
"Aluminiumprofiler"

frmStart
CommandButton Prog(2)
"Hålprofiler av stål"

frmStart
Frame Frame20
"Prisuppdatering"

frmStart
Frame Frame19(1)
"Rostfria profiler"

frmStart
Frame Frame19(0)
"Stålprofiler"

frmStart
CommandButton Avbryt
"Avbryt"

frmStart
CheckBox Check6(1)
"FB klipp"

frmStart
CheckBox Check5
"Stålpris kr/m"

frmStart
OptionButton Option1(2)
"Nibbling"

frmStart
Frame Frame2
"Plåtbearbetning"

frmStart
OptionButton Option1(1)
"Klipning"

frmStart
CommandButton Stål(3)
"ALUMINIUM"

frmStart
Frame Frame1(1)
"HÅLPROFILER"

frmStart
CommandButton Stål(1)
"STÅL"
frmStart
Menu mnuskrivaadr
"Skriva ut adresslistor"

frmStart
Menu mnuPrintstandard(2)
"Printerutskrift allt"

frmStart
CheckBox Check2
"Snedkapning anges i grader"

frmStart
CommandButton Stål(2)
"ROSTFRITT"

frmStart
Menu mnuBytaMapp
"Byta mapp"

frmStart
Menu MB_Slutreg
"Avsluta registrering"

frmStart
OptionButton Option1(0)
"Skärning"

frmStart
OptionButton Option3(0)
"Standard"

frmStart
OptionButton Option3(1)
"Lokalt"

frmStart
CheckBox Ikb
"Ingen kvantrabatt"

frmStart
OptionButton Kmål(3)
"Ingen målning"

frmStart
Frame Frame7(1)
"Prislista"

frmStart
OptionButton KKap(2)
"Snedkapning standard"

frmStart
CheckBox Ipt
"Inget pos. tillägg"
frmStart
CheckBox Check6(2)
"FB skärning"

frmStart
OptionButton KKap(1)
"Rak kapning standard"

frmStart
Frame Frame15
"Prisfiler"

frmStart
OptionButton Option3(2)
"Centralt"

frmStart
Frame Frame4
"Kapning"

frmStart
CommandButton Command12
"Kopiera prisfiler"

frmStart
Frame Frame13
"Standardkatalog för lagring av indatafiler"

frmStart
CommandButton Command15
"Visa nästa"

frmStart
Menu mnuData(1)
"Data till fil"

frmStart
CommandButton Prog(1)
"Valsade stålprofiler"

frmStart
CommandButton Prog(13)
"Påslag lagerförsäljning"

frmStart
OptionButton ii(4)
"4"

frmStart
CheckBox Check3
"Inget målningsindex"

frmStart
Frame Frame5
"Spillberäkning"
frmStart
Frame Frame3
"Prisberäkning"

frmStart
Frame Frame7(0)
"Montage"

frmStart
OptionButton ii(1)
"1"

frmStart
OptionButton ii(3)
"3"

frmStart
CommandButton Commandb1
"Sök"

frmStart
OptionButton Kspill(3)
"Skrotkostnad beräknas inte"

frmStart
OptionButton Direkt(3)
"Direkt till resultat"

frmStart
OptionButton ii(2)
"2"

frmStart
CheckBox Checkb1(2)
"Kontrollintyg"

frmStart
Frame Frame8
"Prisindex"

frmStart
OptionButton Kspill(1)
"Skrotkostnad beräknas av programmet"

frmStart
OptionButton Kspill(2)
"Skrot"

frmStart
CommandButton Commandb1(0)
"Egna kostnader"

frmStart
OptionButton Klitt(1)
"Manuellt"
frmStart
OptionButton Kmont(1)
"Ja"

frmStart
CheckBox Checkb1(1)
"SBS-material"

frmStart
OptionButton Klitt(2)
"Ingen littering"

frmStart
CheckBox Check4
"Utan prisuppdatering"

frmStart
OptionButton Klitt(3)
"Auto"

frmStart
CommandButton Command1(2)
"Aktiviteter"

frmStart
OptionButton Kpris(2)
"Nej"

frmStart
CommandButton Command1(1)
"Å-priser"

frmStart
OptionButton Kpris(1)
"Ja"

frmStart
Frame Frame9
"StålSpec 2000"

frmStart
OptionButton Kmont(2)
"Nej"

frmStart
Frame Frame18
"Standard för kapoptimering"

frmStart
CommandButton Command3
"Avbryt"

frmStart
CommandButton Command7(1)
"TA BORT FIL"
frmStart
CheckBox CheckBox1(3)
"Certifikat"

frmStart
CommandButton Command6
"Utskrift av beräkning"

frmStart
Frame Frame1(2)
"KALLFORMADE PROFILER"

frmStart
Frame Frame21(6)
"Legeringstillägg rostfritt"

frmStart
Frame Frame1(3)
"PLÅT"

frmStart
Frame Frame1(6)
"STÅNG"

frmStart
OptionButton Kmål(2)
"Manuell inmatning"

frmStart
CommandButton Prog(12)
"Diverse produkter"

frmStart
Frame Frame1(4)
"ÄMNESRÖR"

frmStart
CommandButton Prog(14)
"Legeringstillägg"

frmStart
CommandButton Prog(9)
"Kvantitetstillägg"

frmStart
Frame Frame6
"Målning"

frmStart
Frame Frame1(5)
"ÖVRIGA RÖR"

frmStart
OptionButton Option2(1)
"grader"
frmStart
Frame Frame12
"Rostskydd"

frmStart
OptionButton Kmål(1)
"Målningssystem 1 som standard"

frmStart
CommandButton Command8
"Command8"

frmStart
CommandButton Command10
"Command10"

frmStart
OptionButton Direkt(2)
"Direkt till resultat"

frmStart
CommandButton Command7(0)
"SKAPA NY FIL"

frmStart
CommandButton Command2
"Öppna"

frmStart
CommandButton Stål(0)
"STÅLBALKAR"

frmStart
OptionButton Direkt(0)
"Direkt till stålspec"

frmStart
Frame Frame11
"Kapvinkel"

frmStart
CommandButton Command5
"ÖPPNA FIL"

frmStart
OptionButton Option2(0)
"mm"

frmStart
OptionButton Direkt(1)
"Direkt till resultat"

frmStart
Frame Frame10
"Hur vill du öppna filen"
frmStart
Menu mnuData(2)
"Data med e-post"

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

frmStart

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

frmStart

Unicode function is faster

The wide functions AscW and ChrW/ChrW$ are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you're handling ASCII characters from 0 to 127, you're safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 4.

```
15 Sub frmStart.Form_Load
   AscW
21 Sub frmStart.Form_Load
   AscW
4 Sub frmStart.txtedit_KeyPress
   ChrW$
4 Sub frmStart.txtedit_KeyPress
   AscW
```

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>4</td>
</tr>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>4</td>
</tr>
<tr>
<td>Consider using Image control</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Constant available</td>
<td>Optim.</td>
<td>8</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>8</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>4</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Empty procedure/module</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>79</td>
</tr>
</tbody>
</table>
Module11

Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If ..And.. condition as two nested Ifs. Short-circuiting If ..Or.. yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/ AndAlso and Or/ OrElse. Optimization. Severity: Info. Count: 2.

124 Sub Module11.Öppnakap
18 Sub Module11.Sortb

Constant without type specification

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants. Optimization. Severity: Info. Count: 5.

2 Module11.(declarations)
  Pi

2 Sub Module11.Villdu
  mnuIconQuestion

2 Sub Module11.Villdu
  IDYes

2 Sub Module11.Villdu
  IDNo

2 Sub Module11.Villdu
  mnuYesNO
Dead constant

A variable or constant is not used. You may remove it if you are sure you won’t need it later. The removal doesn’t affect the functionality of your program.

```
2 Sub Module11.Villdu
   IDNo
2 Sub Module11.Villdu
   IDYes
```

Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won’t need it later. The removal doesn’t affect the functionality of your program. - Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 2.

```
Sub Module11.Utskrifte
Sub Module11.Utresultat
```

Dead procedure/declaration/event (called by dead only)

A procedure is not executed and it is effectively dead. Calls to the procedure exist, but the callers don’t execute. You should remove this procedure along with its callers, provided that you are sure you won’t need any of the callers later. Optimization. Severity: Warning. Count: 2.

```
Sub Module11.RubrikEgna
Sub Module11.RubrikEgna2
```

Dead return value

A function’s return value is not used by any its callers. Review the function and the callers to determine whether the return value should be used or whether the function should be rewritten as a Sub. This problem does not apply to VB3 where return values must be used at all times. See also the style issue Return value discarded. Optimization. Severity: Warning. Count: 5.

```
Function Module11.Prodime
Function Module11.Printu
Function Module11.Knappfärg
Function Module11.Combo
Function Module11.Exprofil
```

Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won’t need it later. The removal doesn’t affect the functionality of your program.

```
50 Module11.(declarations)
BL
```
Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 58.
Goto statement found

Use of Goto is bad programming practice and should be avoided when possible. Jumping around with Goto easily leads to unstructured control flow structure and spaghetti code. Style. Severity: Info. Count: 5.

8 Sub Module11.Öppnakap
23 Sub Module11.Spara
34 Sub Module11.Printfigur
42 Sub Module11.Printfigur
16 Sub Module11.Utkostnad

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which

Module11

**Unicode function is faster**

The wide functions AscW and ChrW/ChrW$ are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you're handling ASCII characters from 0 to 127, you're safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 3.

11 Sub Module11.Startdata
    AscW
4 Sub Module11.radängd
    AscW
5 Sub Module11.radängd
    AscW

**Variable without type specification**

A variable does not have a defined data type. By default, the type is Variant. Variant needs more memory than other types. Decide what type you need and write it to the variable declaration. Besides, upgrading to VB.NET will be easier if you use explicit data types. Fix recommended before upgrade. Optimization. Severity: Warning. Count: 48.

7 Module11.(declarations)
    xxx
7 Module11.(declarations)
    Revk
18 Module11.(declarations)
    spg
26 Module11.(declarations)
    FFrakt
60 Module11.(declarations)
    qv
70 Module11.(declarations)
    Marg
76 Module11.(declarations)
    zinkv
76 Module11.(declarations)
    gross
111 Module11.(declarations)
    Bild
Module11.(declarations)

xl

Dvo

h

Dhoo

Dhuu

Dhu

dxx

Dhooo

ddy

Rkk

Dvoo

dx

Dvu

Dvu

Dhoo

PB

g

Fris2

dy
mmet
Prisl
Skala
d
rm
t
vm
rvm
vv
vmet
Tvikt
rvv
b
Mtyp
sss
Qvv
DELSUMMF
ställm
Mpri
**Variable written only**

A variable is given a value but the value is never read. Check the location(s) where the variable gets a value. You can remove the assignments and remove the variable declaration if you are sure you won’t need the value later.


<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>5</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>(called by dead only)</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead return value</td>
<td>Optim.</td>
<td>5</td>
</tr>
<tr>
<td>Dead variable/parameter</td>
<td>Optim.</td>
<td>11</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>58</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>5</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Unicode function is faster</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Variable without type specification</td>
<td>Optim.</td>
<td>48</td>
</tr>
<tr>
<td>Variable written only</td>
<td>Optim.</td>
<td>5</td>
</tr>
</tbody>
</table>

**Type**

<table>
<thead>
<tr>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
</tr>
</tbody>
</table>

**Total**

<table>
<thead>
<tr>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
</tr>
</tbody>
</table>

**Problems/logical code line**  

0,03
C. Problem report by description - StålSpec2000 Old Source Code

Form frmApris

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 2.

frmApris
TextBox txtEdit

frmApris
PictureBox ÖppnaSpara

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 2.

3 Sub frmApris.Hämtaapris
3 Sub frmApris.Sparaapris

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

frmApris

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

frmApris

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>2</td>
</tr>
</tbody>
</table>
Fixed file number found          Style          2
Option Explicit missing         Style          1
Resizable Form missing Form_Resize   Funct.         1

Type                     Count
Optimization              2 ******
Style                     3 **********
Functionality             1 ***
Total                     6
Problems/logical code line 0,01

Form frmOpp7

Hotkey missing

frmOpp7
CommandButton Command1
"FLER KAPFIGURER"

Option Explicit missing
A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

frmOpp7

Possibly twisted tab order
The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

frmOpp7
Picture2(0) doesn't precede children

Resizable Form missing Form_Resize
The Form_Resize event is missing from a Form that users can resize at run-time.

frmOppo7

## Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style</td>
<td>1 ******</td>
</tr>
<tr>
<td>Functionality</td>
<td>3 ***********</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,03</td>
</tr>
</tbody>
</table>

### Form frmKappris

#### Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 16.

```vbnet
6 Sub frmKappris.Sparak
13 Sub frmKappris.Sparak
34 Sub frmKappris.Sparak
44 Sub frmKappris.Sparak
56 Sub frmKappris.Sparak
66 Sub frmKappris.Sparak
73 Sub frmKappris.Sparak
5 Sub frmKappris.Visasida
15 Sub frmKappris.Visasida
99 Sub frmKappris.Visasida
106 Sub frmKappris.Visasida
187 Sub frmKappris.Visasida
234 Sub frmKappris.Visasida
313 Sub frmKappris.Visasida
333 Sub frmKappris.Visasida
361 Sub frmKappris.Visasida
```

#### Hotkey missing


```vbnet
frmKappris
Frame Frame1(0)
"GRUNDFRIS, kr/m²"
```

```vbnet
frmKappris
Frame Frame8
```
"KOSTNAD FÖR STANSNING/NIBBLING"

frmKappris
Frame Frame10
"KOSTNAD FÖR GASSKÄRNING, kr/skärmeter"

frmKappris
Frame Frame6
"LÄNGDFAKTOR"

frmKappris
Frame Frame3(1)
"MATERIALFAKTOR"

frmKappris
Frame Frame7
"Kostnad för bockning och vändning av plåt, kr/bock"

frmKappris
Frame Frame2(1)
"PRISFAKTOR"

frmKappris
Frame Frame9
"Klippkostnader, kr/kg"

frmKappris
Frame Frame3(0)
"MÄNGDFAKTOR, SUMMA YTA PER ARTIKELRAD"

frmKappris
Frame Frame5
"STÄLLKOSTNAD, Kr PER DIMENSION"

frmKappris
Frame Frame3(3)
"GERFAKTOR"

frmKappris
Frame Frame1(1)
"PRISER FÖR KAPNING AV BALK OCH PROFILER I HELA KRONOR"

**Option Explicit missing**

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style: Critical. Count: 1.

frmKappris

**Resizable Form missing Form_Resize**

The Form_Resize event is missing from a Form that users can resize at run-time.

frmKappris

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>16</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>12</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Style</td>
<td>17</td>
<td>*******</td>
</tr>
<tr>
<td>Functionality</td>
<td>13</td>
<td>*******</td>
</tr>
<tr>
<td>Total</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,03</td>
<td></td>
</tr>
</tbody>
</table>

Form frmNyp

Click event missing


frmNyp
CommandButton Visk(6)

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 3.

frmNyp
Label Label9

frmNyp
TextBox txtEdit

frmNyp
Label Label10

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile

3 Sub frmNyp.Sparalager
2 Sub frmNyp.Spararabb
11 Sub frmNyp.Form_Activate
3 Sub frmNyp.Lager
3 Sub frmNyp.Läsainpris
2 Sub frmNyp.PrisSpara

Goto statement found


31 Sub frmNyp.MB_till_Click

Hotkey conflict

Two or more controls or menu items share a hotkey. For example, there are options &Save and &Search in the same menu. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

frmNyp
Conflict with controls
PKA: "&Arkiv" "&Priser" "Pris kr/&kg" "&Prisjustering" "&Kvalitet"
"&Avsluta revide"

Hotkey missing


frmNyp
Frame Frame1
"FÖLJANDE KVALITETER FINNS ATT VÄLJA MELLAN"

frmNyp
Frame Frame3
"$"frmNyp.frx":058A"

frmNyp
Menu MB_till
"Justering i förhållande till"

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

frmNyp
Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

frmNyp

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>6</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Hotkey conflict</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>3</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>3 ****</td>
</tr>
<tr>
<td>Style</td>
<td>8 ******</td>
</tr>
<tr>
<td>Functionality</td>
<td>6 *******</td>
</tr>
<tr>
<td>Total</td>
<td>17</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,02</td>
</tr>
</tbody>
</table>

Form frmOpp5

Click event missing


frmOpp5
CommandButton Command5

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

frmOpp5
PictureBox Picture2
Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 5.

- Sub frmOpp5.Ingentext
- Sub frmOpp5.spillb
- Sub frmOpp5.Vänster_Click
- Sub frmOpp5.visamer
- Sub frmOpp5.Lulle

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 5.

- 11 Sub frmOpp5.Hämta
- 13 Sub frmOpp5.Hämta
- 5 Sub frmOpp5.Form_Activate
- 2 Sub frmOpp5.Hämtakapning
- 7 Sub frmOpp5.Hämtakapning

Goto statement found

Use of Goto is bad programming practice and should be avoided when possible. Jumping around with Goto easily leads to unstructured control flow structure and spaghetti code. Style. Severity: Info. Count: 2.

- 20 Sub frmOpp5.Hämta
- 38 Sub frmOpp5.Hämta

Hotkey missing


- frmOpp5 CommandButton Commandg "GROSSISTLAGER"
- frmOpp5 CheckBox Check2 "1 Längd/balk"
- frmOpp5 Menu Visar(3) "Visa profilkapning på skrivare"
- frmOpp5 CommandButton Command5 "Flytta"
frmOpp5
CheckBox Check1(1)
"Check1"

frmOpp5
CommandButton Command7
"MARKERA ALLA"

frmOpp5
CommandButton Command1
"EGET LAGER"

frmOpp5
CommandButton Command4
"OPTIMERA"

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.
Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>5</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>5</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>2</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>8</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>6 *****</td>
</tr>
<tr>
<td>Style</td>
<td>8 ******</td>
</tr>
<tr>
<td>Functionality</td>
<td>11 *******</td>
</tr>
<tr>
<td>Total</td>
<td>25</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,02</td>
</tr>
</tbody>
</table>

Form frmRes2

Click event missing


    frmRes2
    CommandButton Command1(1)

Consider using Image control

A PictureBox uses significantly more memory and system resources than an Image control. While PictureBox has more features, this PictureBox is a potential candidate to be replaced by an Image control. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

    frmRes2
    PictureBox Picture2

Constant without type specification

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants. Optimization. Severity: Info. Count: 3.

    2 Sub frmRes2.Lagersaldo IDYes
    2 Sub frmRes2.Lagersaldo IDNo
    2 Sub frmRes2.Lagersaldo mnuIconQuestion
Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 2.

```vbnet
frmRes2
TextBox txtedit
frmRes2
PictureBox Picture6
```

Dead constant

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 1.

```vbnet
2 Sub frmRes2.Lagersaldo
IDNo
```

Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 1.

```vbnet
4 frmRes2.(declarations)
Krabb
```

Empty procedure/module


```vbnet
Sub frmRes2.txtEdit_LostFocus
```

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 28.

```vbnet
12 Sub frmRes2.Bocka
3 Sub frmRes2.FBMåla
47 Sub frmRes2.Grundpris
49 Sub frmRes2.Grundpris
9 Sub frmRes2.Lagersaldo
3 Sub frmRes2.Nibbla
105 Sub frmRes2.Form_Activate
```
Hotkey missing


```vbnet
frmRes2
CommandButton Command1(3)
"Aktiviteter"

frmRes2
Frame Bild41
"FB-målning hos grossist"

frmRes2
Frame Frame3
"Fri text"

frmRes2
Frame Frame1
"Kund"

frmRes2
CommandButton Command1(1)
"Egna kostnader"

frmRes2
CommandButton Command1(2)
"À-priser"
```

Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some
nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

frmRes2

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

frmRes2

Variable without type specification

A variable does not have a defined data type. By default, the type is Variant. Variant needs more memory than other types. Decide what type you need and write it to the variable declaration. Besides, upgrading to VB.NET will be easier if you use explicit data types. Fix recommended before upgrade. Optimization. Severity: Warning. Count: 12.
Variable written only

A variable is given a value but the value is never read. Check the location(s) where the variable gets a value. You can remove the assignments and remove the variable declaration if you are sure you won’t need the value later.


```
frmRes2.(declarations)
Ställ

frmRes2.(declarations)
Kpos
```

Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Consider using Image control</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead variable/parameter</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Empty procedure/module</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>28</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>6</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Variable without type specification</td>
<td>Optim.</td>
<td>12</td>
</tr>
<tr>
<td>Variable written only</td>
<td>Optim.</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>23</td>
</tr>
<tr>
<td>Style</td>
<td>29</td>
</tr>
<tr>
<td>Functionality</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,02</td>
</tr>
</tbody>
</table>

Form frmegen

Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If ..And.. condition as two nested Ifs. Short-circuiting If ..Or.. yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/ AndAlso and Or/ OrElse. Optimization. Severity: Info. Count: 1.

```
8 Sub frmegen.txtedit_KeyPress
```

Constant without type specification
A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants.

```
10 Sub frmegen.MB_Bortkval_Click
    IDYes
10 Sub frmegen.MB_Bortkval_Click
    IDNo
10 Sub frmegen.MB_Bortkval_Click
    MB_IconQuestion
11 Sub frmegen.MB_Bortkval_Click
    MB_YesNO
```

Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

```
    frmegen
    TextBox txtEdit
```

Dead constant

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program.

```
10 Sub frmegen.MB_Bortkval_Click
    IDYes
10 Sub frmegen.MB_Bortkval_Click
    IDNo
```

Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program.

```
    6 Sub frmegen.MB_Byta_Click
        Revka
```

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time

60 Sub frmegen.MB_Ny_Click
71 Sub frmegen.MB_Ny_Click
79 Sub frmegen.MB_Ny_Click
15 Sub frmegen.MB_Profildata_Click
52 Sub frmegen.MB_Radbort_Click
60 Sub frmegen.MB_Radbort_Click
6 Sub frmegen.Sparap
2 Sub frmegen.SparaPdata
2 Sub frmegen.OppnaPdata

Goto statement found

Use of Goto is bad programming practice and should be avoided when possible. Jumping around with Goto easily leads to unstructured control flow structure and spaghetti code. Style. Severity: Info. Count: 11.

39 Sub frmegen.kr_Click
41 Sub frmegen.kr_Click
45 Sub frmegen.kr_Click
19 Sub frmegen.MB_Noll_Click
21 Sub frmegen.MB_Noll_Click
23 Sub frmegen.MB_Noll_Click
96 Sub frmegen.MB_Ny_Click
45 Sub frmegen.MB_Profildata_Click
34 Sub frmegen.Procent_Click
36 Sub frmegen.Procent_Click
39 Sub frmegen.Procent_Click

Hotkey conflict

Two or more controls or menu items share a hotkey. For example, there are options &Save and &Search in the same menu. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

frmegen
Conflict with controls
A: "&Arkiv" "/&Automatisk pri"

Hotkey missing


frmegen
Menu Procent
"Prisjustering i procent"

frmegen
Frame Frame1
"Profiltyper"
Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

```
frmege
```

Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed if move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

```
frmege
txtEdit->Grid1
```

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

```
frmege
```

Unicode function is faster

The wide functions AscW and ChrW/ChrW$ are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you're handling ASCII characters from 0 to 127, you're safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 4.

```
13 Sub frmege.Text1_KeyPress
    AscW
13 Sub frmege.Text1_KeyPress
    ChrW$
10 Sub frmege.Text2_KeyPress
    ChrW$
10 Sub frmege.Text2_KeyPress
    AscW
```
Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>4</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead variable/parameter</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>9</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>11</td>
</tr>
<tr>
<td>Hotkey conflict</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>2</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Unicode function is faster</td>
<td>Optim.</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>13</td>
</tr>
<tr>
<td>Style</td>
<td>21</td>
</tr>
<tr>
<td>Functionality</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>39</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,04</td>
</tr>
</tbody>
</table>

Module1

Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If ..And.. condition as two nested Ifs. Short-circuiting If ..Or.. yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/AndAlso and Or/OrElse. Optimization. Severity: Info. Count: 2.

130 Sub Module1.Öppnakap
18 Sub Module1.Sortb

Constant without type specification

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants. Optimization. Severity: Info. Count: 4.

2 Sub Module1.Villdu
    IDNo

2 Sub Module1.Villdu
    IDYes

2 Sub Module1.Villdu
    mnuYesNO
Dead constant

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 2.

Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. - Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 1.

Dead return value

A function's return value is not used by any its callers. Review the function and the callers to determine whether the return value should be used or whether the function should be rewritten as a Sub. This problem does not apply to VB3 where return values must be used at all times. See also the style issue Return value discarded. Optimization. Severity: Warning. Count: 2.

Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 21.
Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 56.
**Goto statement found**


```
  8 Sub Module1.Öppnakap
  22 Sub Module1.Spara
  138 Sub Module1.Utskrift
  140 Sub Module1.Utskrift
  198 Sub Module1.Utskrift
  25 Sub Module1.Sparae
  3 Sub Module1.Stålsortk
  35 Sub Module1.Printfigur
  43 Sub Module1.Printfigur
```

**Option Explicit missing**

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

```
Module1
```

**Unicode function is faster**

The wide functions AscW and ChrW/ChrW$ are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you're handling ASCII characters from 0 to 127, you're safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 2.

```
  4 Sub Module1.radlängd
    AscW
  5 Sub Module1.radlängd
    AscW
```

**Variable without type specification**

A variable does not have a defined data type. By default, the type is Variant. Variant needs more memory than other types. Decide what type you need and write it to the variable declaration. Besides, upgrading to VB.NET will be easier if you use explicit data types. Fix recommended before upgrade. Optimization. Severity: Warning. Count: 46.

```
  9 Module1.(declarations)
    xxx
  9 Module1.(declarations)
    Revk
  20 Module1.(declarations)
    spg
  28 Module1.(declarations)
```
FFrakt

30 Module1.(declarations)
  soo

60 Module1.(declarations)
  qv

70 Module1.(declarations)
  Marg

76 Module1.(declarations)
  gross

76 Module1.(declarations)
  zinkv

111 Module1.(declarations)
  Bild

118 Module1.(declarations)
  xl

122 Module1.(declarations)
  Dvuuu

122 Module1.(declarations)
  ddy

122 Module1.(declarations)
  Rkk

122 Module1.(declarations)
  Dvu

122 Module1.(declarations)
  Dvo

122 Module1.(declarations)
  dxx

122 Module1.(declarations)
  Dvoo

122 Module1.(declarations)
  Dhooo

122 Module1.(declarations)
  dx

122 Module1.(declarations)
  Dhoo

122 Module1.(declarations)
  Dho
Module1.(declarations)
  h
Module1.(declarations)
  Dhuu
Module1.(declarations)
  Dhu
Module1.(declarations)
  t
Module1.(declarations)
  vmet
Module1.(declarations)
  FB
Module1.(declarations)
  rvv
Module1.(declarations)
  vv
Module1.(declarations)
  rvm
Module1.(declarations)
  vm
Module1.(declarations)
  Pris2
Module1.(declarations)
  Prisl
Module1.(declarations)
  Tviktn
Module1.(declarations)
  mmetn
Module1.(declarations)
  g
Module1.(declarations)
  d
Module1.(declarations)
  dy
Module1.(declarations)
  Skalan
Module1.(declarations)
  rmetm
Variable written only

A variable is given a value but the value is never read. Check the location(s) where the variable gets a value. You can remove the assignments and remove the variable declaration if you are sure you won't need the value later.


Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>4</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead return value</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Dead variable/parameter</td>
<td>Optim.</td>
<td>21</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>56</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>9</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Unicode function is faster</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Variable without type specification</td>
<td>Optim.</td>
<td>46</td>
</tr>
<tr>
<td>Variable written only</td>
<td>Optim.</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>85</td>
</tr>
<tr>
<td>Style</td>
<td>66</td>
</tr>
</tbody>
</table>
Form frmSpecin

Click event missing


```vbnet
frmSpecin
    CommandButton Command2
frmSpecin
    CommandButton Command3
frmSpecin
    CommandButton Knapp(9)
frmSpecin
    CommandButton Command1
frmSpecin
    CommandButton Command4
frmSpecin
    CommandButton Command5
frmSpecin
    CommandButton Command6
frmSpecin
    CommandButton Command10
```

Consider short-circuited logic

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If...And... condition as two nested Ifs. Short-circuiting If...Or... yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the second operand calls a function, this call may not execute. Read VB help for differences between And/AndAlso and Or/OrElse. Optimization. Severity: Info. Count: 1.

```vbnet
3 Sub frmSpecin.pGrid_MouseUp
```

Constant without type specification

A constant has no defined data type. VB will pick an appropriate data type for it. The chosen type may not be optimal where the constant value is actually used. You can use this rule to require explicit typing of constants. Optimization. Severity: Info. Count: 3.

```vbnet
2 Sub frmSpecin.Villdukval
    IDYes
```
Control not visible

A control's Visible property is set to False and it is not set to True by code. Most user interface controls are not very useful if they are not visible. The control was possibly set invisible by a developer who thought it was not required any more but was uncertain about removing it. You should consider removing invisible controls as they may unnecessarily consume some resources. Events related to the control are possibly not executed. Code that reads or sets the control's properties and calls its methods is potentially unnecessary for the operation of the program. There are some cases where an invisible control can be useful, so you have to be careful about removing the control and related code. This rule does not work with control arrays. It does not detect all kinds of invisibility, as controls may be hidden behind other controls or set invisible by code. Applies to VB 3-6. Optimization. Severity: Warning. Count: 1.

Dead constant

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 1.

Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the project. It is not called by the code nor executed by any other means. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Event declarations are reported dead only if they are not raised nor handled. See the problem Event not raised for events that would be handled but that are not fired. Optimization. Severity: Warning. Count: 3.

Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 2.
**Fixed file number found**

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 16.

71 Sub frmSpecin.viktber
73 Sub frmSpecin.viktber
17 Sub frmSpecin.Visadimval
11 Sub frmSpecin.Visalev
12 Sub frmSpecin.Visaval
31 Sub frmSpecin.Command11_Click
36 Sub frmSpecin.Command11_Click
9 Sub frmSpecin.Form_Activate
110 Sub frmSpecin.Form_Load
120 Sub frmSpecin.Form_Load
85 Sub frmSpecin.Grid_MouseUp
10 Sub frmSpecin.mnuBortModul_Click
30 Sub frmSpecin.mnuSorterar_Click
35 Sub frmSpecin.mnuSorterar_Click
97 Sub frmSpecin.txtedit_KeyPress
10 Sub frmSpecin.Moduler

**Goto statement found**


49 Sub frmSpecin.Kollktyp
52 Sub frmSpecin.Kollktyp
6 Sub frmSpecin.Grid_KeyDown
15 Sub frmSpecin.Grid_KeyDown

**Hotkey missing**


frmSpecin
CommandButton Command12(2)
"Aluminium"

frmSpecin
CommandButton Command12(0)
"Stål"

frmSpecin
Frame Frame3
"Plåtbearbetning"

frmSpecin
CommandButton Knapp(9)
"UNI"
frmSpecin
CommandButton Command15
"Visa allt"

frmSpecin
Frame Frame2
"Sök profil"

frmSpecin
CommandButton Command2
"ALUMINIUM"

frmSpecin
CommandButton Command1
"ROSTFRITT"

frmSpecin
CommandButton Command6
"EXTRA"

frmSpecin
CheckBox Check1(2)
"Klippning"

frmSpecin
CommandButton Command8(0)
"Grader - mm"

frmSpecin
CheckBox Check1(1)
"Skärning"

frmSpecin
CommandButton Command8(1)
"mm - grader"

frmSpecin
CommandButton Knapp(7)
"PLÅT"

frmSpecin
CommandButton Knapp(8)
"PLS"

frmSpecin
CommandButton Command11
"Uppdatera lager"

frmSpecin
CommandButton Knapp(1)
"HEA"

frmSpecin
CommandButton Knapp(3)
"IPE"

frmSpecin
Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.
Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. A back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

Picture8->Picture7, Picture5 doesn't precede children, Iinfo(3)->Iinfo(2), ...

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

Unicode function is faster

The wide functions AscW and ChrW/ChrW$ are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you're handling ASCII characters from 0 to 127, you're safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 4.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>8</td>
</tr>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>1</td>
</tr>
</tbody>
</table>
Dead procedure/declaration/event          Optim.         3  
Dead variable/parameter                   Optim.         2  
Fixed file number found                   Style         16  
Goto statement found                      Style          4  
Hotkey missing                            Funct.         30  
Option Explicit missing                   Style          1  
Possibly twisted tab order                 Funct.          1  
Resizable Form missing Form_Resize         Funct.          1  
Unicode function is faster                 Optim.          4  

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>15 ****</td>
</tr>
<tr>
<td>Style</td>
<td>21 *****</td>
</tr>
<tr>
<td>Functionality</td>
<td>40 ***********</td>
</tr>
<tr>
<td>Total</td>
<td>76</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,02</td>
</tr>
</tbody>
</table>

**Form frmSpecin**

**Click event missing**


```vbnet
frmSpecin
    CommandButton Command2

frmSpecin
    CommandButton Command3

frmSpecin
    CommandButton Knapp(9)

frmSpecin
    CommandButton Command1

frmSpecin
    CommandButton Command4

frmSpecin
    CommandButton Command5

frmSpecin
    CommandButton Command6

frmSpecin
    CommandButton Command10
```

**Consider short-circuited logic**

In the expressions (x And y), (x Or y), both operands (x, y) are evaluated. Short-circuiting means rewriting this so that when x=False in (x And y), y is not evaluated. The same goes for x=True in (x Or y). This saves CPU cycles, especially if y is a complex expression. In VB.NET, consider replacing And with AndAlso, and Or with OrElse. In VB Classic, consider splitting an If ..And.. condition as two nested Ifs. Short-circuiting If ..Or.. yields more complex code, usable case by case. Risks: Short-circuiting changes the logic. If the
second operand calls a function, this call may not execute. Read VB help for
differences between And/AndAlso and Or/OrElse. Optimization. Severity: Info.
Count: 1.

3 Sub frmSpecin.pGrid_MouseUp

Constant without type specification

A constant has no defined data type. VB will pick an appropriate data type for
it. The chosen type may not be optimal where the constant value is actually
used. You can use this rule to require explicit typing of constants.

2 Sub frmSpecin.Villdukval
   IDYes
2 Sub frmSpecin.Villdukval
   mnuIconQuestion
2 Sub frmSpecin.Villdukval
   IDNo

Control not visible

A control's Visible property is set to False and it is not set to True by code.
Most user interface controls are not very useful if they are not visible. The
control was possibly set invisible by a developer who thought it was not
required any more but was uncertain about removing it. You should consider
removing invisible controls as they may unnecessarily consume some resources.
Events related to the control are possibly not executed. Code that reads or
sets the control's properties and calls its methods is potentially unnecessary
for the operation of the program. There are some cases where an invisible
control can be useful, so you have to be careful about removing the control and
related code. This rule does not work with control arrays. It does not detect
all kinds of invisibility, as controls may be hidden behind other controls or
Count: 1.

   frmSpecin
   PictureBox Picture6

Dead constant

A variable or constant is not used. You may remove it if you are sure you won't
need it later. The removal doesn't affect the functionality of your program.

2 Sub frmSpecin.Villdukval
   IDNo

Dead procedure/declaration/event

A procedure, a DLL declaration or an Event declaration is not used by the
project. It is not called by the code nor executed by any other means. You may
remove it if you are sure you won't need it later. The removal doesn't affect
the functionality of your program. - Event declarations are reported dead only
if they are not raised nor handled. See the problem Event not raised for events
that would be handled but that are not fired. Optimization. Severity: Warning.
Count: 3.

Sub frmSpecin.totviktyta
Dead variable/parameter

A variable or constant is not used. You may remove it if you are sure you won't need it later. The removal doesn't affect the functionality of your program. Optimization. Severity: Warning. Count: 2.

2 frmSpecin.(declarations)
KKK

4 frmSpecin.(declarations)
Mark

Fixed file number found

Using fixed file numbers in file handling is a potential cause of run-time errors. It is better to use a variable and initiate it with the FreeFile function, like this: FileNum = FreeFile. Style. Severity: Warning. Count: 16.

71 Sub frmSpecin.vikter
73 Sub frmSpecin.vikter
17 Sub frmSpecin.Visadimal
11 Sub frmSpecin.Visalev
12 Sub frmSpecin.Visaval
31 Sub frmSpecin.Command11_Click
36 Sub frmSpecin.Command11_Click
9 Sub frmSpecin.Form_Activate
110 Sub frmSpecin.Form_Load
120 Sub frmSpecin.Form_Load
85 Sub frmSpecin.Grid_MouseUp
10 Sub frmSpecin.mnuBortModul_Click
30 Sub frmSpecin.mnuSorterar_Click
35 Sub frmSpecin.mnuSorterar_Click
97 Sub frmSpecin.txtedit_KeyPress
10 Sub frmSpecin.Moduler

Goto statement found


49 Sub frmSpecin.Kollktyp
52 Sub frmSpecin.Kollktyp
6 Sub frmSpecin.Grid_KeyDown
15 Sub frmSpecin.Grid_KeyDown

Hotkey missing


frmSpecin
CommandButton Command12(2)
"Aluminium"
frmSpecin
CommandButton Command12(0)
"Stål"

frmSpecin
Frame Frame3
"Plåtbearbetning"

frmSpecin
CommandButton Knapp(9)
"UNI"

frmSpecin
CommandButton Command15
"Visa allt"

frmSpecin
Frame Frame2
"Sök profil"

frmSpecin
CommandButton Command2
"ALUMINIUM"

frmSpecin
CommandButton Command1
"ROSTFRITT"

frmSpecin
CommandButton Command6
"EXTRA"

frmSpecin
CheckBox Check1(2)
"Klippning"

frmSpecin
CommandButton Command8(0)
"Grader - mm"

frmSpecin
CheckBox Check1(1)
"Skärning"

frmSpecin
CommandButton Command8(1)
"mm - grader"

frmSpecin
CommandButton Knapp(7)
"PLÅT"

frmSpecin
CommandButton Knapp(8)
"PLS"

frmSpecin
CommandButton Command11
"Uppdatera lager"

frmSpecin
CommandButton Knapp(1)
"HEA"

frmSpecin
CommandButton Knapp(3)
"IPE"

frmSpecin
CommandButton Knapp(2)
"HEB"

frmSpecin
Frame Framel
"Bockning"

frmSpecin
CheckBox Check1(3)
"Nibbling"

frmSpecin
CommandButton Command3
"EGNA PRODUKTER"

frmSpecin
CommandButton Knapp(4)
"UPE"

frmSpecin
CommandButton Knapp(6)
"KKR"

frmSpecin
CommandButton Command5
"AVSYNING"

frmSpecin
CommandButton Knapp(5)
"VKR"

frmSpecin
CommandButton Command7
"Tillbaka"

frmSpecin
CommandButton Command10
"EGNA PROFILER"

frmSpecin
CommandButton Command12(1)
"Rostfritt"

frmSpecin
Option Explicit missing

A file is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names. The use of Option Explicit will also help in upgrading code to VB.NET. By forcing explicit variable declaration you prevent accidental use of late binding, which is harder to upgrade. Fix recommended before upgrade. Style. Severity: Critical. Count: 1.

Possibly twisted tab order

The tab order of some controls on a form looks questionable. Take a look at the TabIndex properties of the mentioned controls. The best way to do this is to open the form in VB, select the first control and press the Tab key repeatedly. Generally, the focus should move either right or down. Labels with an access key (& in the caption) should precede the control next to them. A container control should precede any child controls placed within it. An back jump is allowed to move the focus from the bottom of the form to the top-left corner. If the form's RightToLeft property is True, Project Analyzer uses a mirrored rule to require moves either left or down. The twisted tab order detection rules are not foolproof. In some cases, a twisted-looking tab order might be all right. Forms that hide, show or move their controls at run-time are exceptional in that their design-time layout is different from the run-time layout. In that case, this rule may produce a false warning and you should ignore it. Applies to VB 3-6. Functionality. Severity: Warning. Count: 1.

Resizable Form missing Form_Resize

The Form_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events. Applies to VB 3-6. Functionality. Severity: Critical. Count: 1.

Unicode function is faster

The wide functions AscW and ChrW$/ChrW are faster than the Asc/Chr/Chr$ alternatives. VB works internally in Unicode, so the unicode versions run faster. They are not the same functions though. If you're handling ASCII characters from 0 to 127, you're safe to replace Asc with AscW and Chr with ChrW/ChrW$. Applies to VB4 and later. Optimization. Severity: Info. Count: 4.
### Problem summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click event missing</td>
<td>Funct.</td>
<td>8</td>
</tr>
<tr>
<td>Consider short-circuited logic</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Constant without type specification</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Control not visible</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead constant</td>
<td>Optim.</td>
<td>1</td>
</tr>
<tr>
<td>Dead procedure/declaration/event</td>
<td>Optim.</td>
<td>3</td>
</tr>
<tr>
<td>Dead variable/parameter</td>
<td>Optim.</td>
<td>2</td>
</tr>
<tr>
<td>Fixed file number found</td>
<td>Style</td>
<td>16</td>
</tr>
<tr>
<td>Goto statement found</td>
<td>Style</td>
<td>4</td>
</tr>
<tr>
<td>Hotkey missing</td>
<td>Funct.</td>
<td>30</td>
</tr>
<tr>
<td>Option Explicit missing</td>
<td>Style</td>
<td>1</td>
</tr>
<tr>
<td>Possibly twisted tab order</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Resizable Form missing Form_Resize</td>
<td>Funct.</td>
<td>1</td>
</tr>
<tr>
<td>Unicode function is faster</td>
<td>Optim.</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>15 *****</td>
</tr>
<tr>
<td>Style</td>
<td>21 *****</td>
</tr>
<tr>
<td>Functionality</td>
<td>40 **********</td>
</tr>
<tr>
<td>Total</td>
<td>76</td>
</tr>
<tr>
<td>Problems/logical code line</td>
<td>0,02</td>
</tr>
</tbody>
</table>
## D. Upgrading issues from VB6 to VB.NET

<table>
<thead>
<tr>
<th>New Filename</th>
<th>Original Filename</th>
<th>File Type</th>
<th>Status</th>
<th>Errors</th>
<th>Warnings</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmApris.vb</td>
<td>frmApris.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>22</td>
<td>25</td>
<td>47</td>
</tr>
<tr>
<td>Frmegen2.vb</td>
<td>frmegen2.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>4</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>frmGranska.vb</td>
<td>frmGranska.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>3</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>frmKappris.vb</td>
<td>frmKappris.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>frmNypa.vb</td>
<td>frmNypa.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>1</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>frmOpp5.vb</td>
<td>frmOpp5.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>29</td>
<td>52</td>
<td>81</td>
</tr>
<tr>
<td>frmOpp7.vb</td>
<td>frmOpp7.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>3</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>frmRes2.vb</td>
<td>frmRes2.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>13</td>
<td>156</td>
<td>169</td>
</tr>
<tr>
<td>frmSpecin.vb</td>
<td>frmSpecin.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>13</td>
<td>114</td>
<td>127</td>
</tr>
<tr>
<td>frmStart.vb</td>
<td>frmStart.frm</td>
<td>Form</td>
<td>Uppgraded with issues</td>
<td>35</td>
<td>120</td>
<td>155</td>
</tr>
<tr>
<td>Module11.vb</td>
<td>Modul11.bas</td>
<td>Module</td>
<td>Uppgraded with issues</td>
<td>158</td>
<td>345</td>
<td>503</td>
</tr>
<tr>
<td>11 files</td>
<td></td>
<td></td>
<td>Forms: 10 Uppgraded: 11</td>
<td>283</td>
<td>887</td>
<td>1170</td>
</tr>
</tbody>
</table>