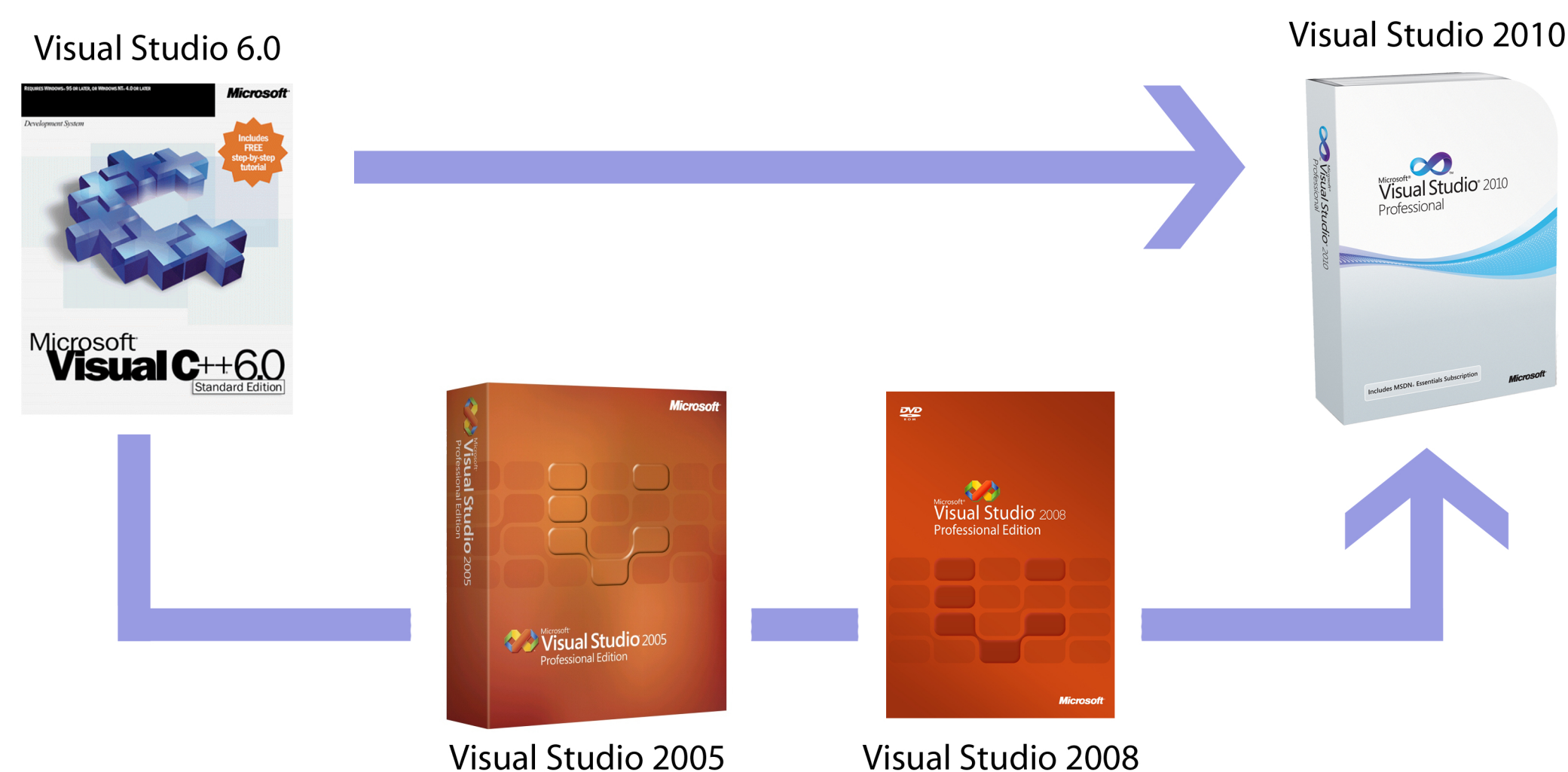


Migratiestrategieën van C++ naar C#



De eerste stappen

Het migreren van een project gemaakt in Visual Studio 6.0 naar een compileerbaar project in Visual Studio 2010, is niet vanzelfsprekend. Tussen de verschillende versies van Visual Studio zijn er talloze, subtiele verschillen die fouten kunnen veroorzaken.

Zo was het in Visual C++ 6.0 nog niet verplicht om een type identifier op te geven indien het ging over een integer. In latere versies wordt dit beschouwd als een syntax error. Met behulp van een zelfgeschreven programma iSE worden dergelijke fouten opgespoord. De fouten waarvoor een oplossing beschikbaar is binnen het programma, worden automatisch opgelost. Fouten die niet kunnen voorzien worden door het programma, moeten manueel afgehandeld worden.

Voorbeeld van een typische conversiefout:

```
const CHAR_MASK = 0xFF; // VC++ 6.0
const int CHAR_MASK = 0xFF; // VC++ 7.0 of hoger
```

De meeste fouten hebben een kenmerkende structuur. Hierdoor kunnen we met behulp van **reguliere expressies** deze fouten oplossen. Dit specifiek probleem kunnen we oplossen met de reguliere expressie: `^(.*)\\(\\w*\\s*=.*)(\\w*(.))$`. Deze vervangen we dan door `\\1int \\2`.

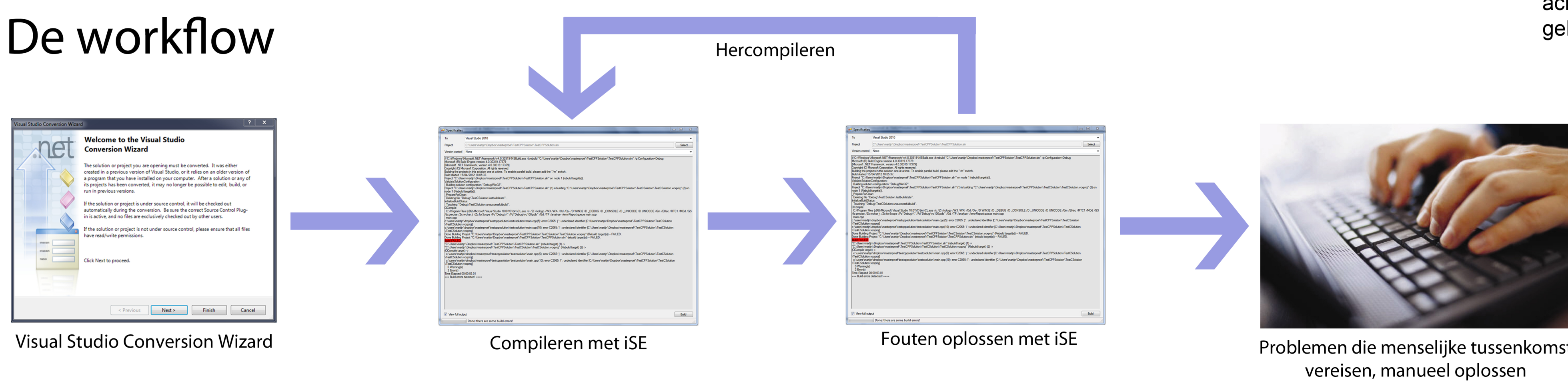
Voorbeeld van een onoplosbare conversiefout:

```
LV_ITEM* pItem= &(pDispInfo)->item;
CString hulp = _T("");
GetString(GetRecordset(), pItem->iSubItem, hulp);
tcscpy(pItem->pszText, (LPCTSTR) (hulp));
```

Sommige fouten kunnen niet programmatorisch opgelost worden. Door dergelijke fouten te documenteren, kan men een groot deel manueel oplossen. Men kan de meeste fouten indelen in twee grote groepen:

- **Linker errors:** ze worden veroorzaakt doordat de compiler bepaalde bestanden niet meer terug vindt. Dit kan meestal manueel opgelost worden door de projectinstellingen aan te passen. Vaak valt dit te wijten aan het gebruik van externe libraries.
- **Fouten door het ontbreken van informatie:** zo ontbreekt er in de voorbeeldcode de buffergrootte van `pItem->pszText` waardoor men de veilige variant, `strcpy_s`, niet kan gebruiken. De lengte valt ook niet te achterhalen met functies als `_countof()`, `sizeof()`,... omdat men gebruik maakt van pointers.

De workflow



Het upgraden van een project naar een hogere versie van Visual Studio bestaat uit drie grote stappen:

- Het project wordt eerst in de gewenste versie van Visual Studio geladen. Visual Studio start automatisch zijn **Conversion Wizard** op. Deze zorgt ervoor dat de projectbestanden compatibel gemaakt worden met de gewenste versie. Deze aanpassingen hebben enkel invloed op de instellingen en bestandsstructuur van het project, niet op de code zelf.

Projecten in Visual Studio 6.0 maken gebruik van *dsp*- (projecten) en *dsw*-bestanden (solutions). Vanaf Visual Studio 2003 en hoger wordt er gebruik gemaakt van de extensies *vcproj* (projecten) en *sln* (solutions). Vanaf Visual Studio 2010 maakt men gebruik van de *vcxproj*-extensie in plaats van de *vcproj*-extensie. De Conversion Wizard van Visual Studio converteert de bestanden naar het juiste type en past ook de nodige projectinstellingen aan. Ook zorgt het ervoor dat men gebruik kan maken van nieuwe features in Visual Studio.

- Hierna wordt het project geopend in **iSE**. Deze gebruikt de output van de compiler van Visual Studio om fouten te detecteren. Via reguliere expressies analyseert iSE de foutinformatie en koppelt indien mogelijk de juiste oplossing aan de fout. De programmeur controleert de oplossingen aangeboden door iSE en bepaalt welke ervan doorgevoerd moeten worden. Zolang iSE fouten vindt, dient er gehercompileerd te worden. Bij het hercompilieren geraakt de compiler verder en kunnen er nieuwe fouten opduiken.

Men spreekt de compiler van Visual Studio aan met de **commandline**. Men gebruikt hiervoor het commendo *VCBuild.exe*. In Visual Studio 2010 en hoger is het commando vervangen door *MSBuild.exe*. MSBuild behoort in tegenstelling tot VCBuild niet tot Visual Studio, maar tot het .NET framework. De informatie, gefilterd uit de output, wordt gebruikt om de oplossing te initialiseren met de benodigde informatie. Men krijgt bijvoorbeeld de volgende fout: `"c:\\libraries\\czam\\czamreaderwriter.cpp(211) : error C2065 : 'i': undeclared identifier"`. Reguliere expressies filteren het lijnnummer, de bestandslocatie en andere nodige informatie uit de foutmelding. Men initialiseert de oplossing met de term `"i"` en koppelt deze aan de fout. Wanneer de fout opgelost moet worden, zal iSE voor deze fout in het bestand vanaf de foutlijn, regel per regel, naar omhoog zoeken naar de term. Als men de term tegenkomt in een declaratie, gaat men na of de term in dezelfde of een hiërarchisch hogere scope zit. Indien dit het geval is, zal iSE de type identifier ophalen en deze op de juiste plek in de foutlijn invoegen.

- Sommige fouten zijn niet programmatorisch op te lossen. Voorbeelden hiervan zijn: interacties met externe libraries, projectspecifieke eigenschappen, link errors,... Deze fouten moeten **manueel** opgelost worden aangezien de code zelf geen fouten bevat.

Van C++ naar C#

De laatste stap is ook de grootste stap. Hier wordt de **syntax en de functionaliteit** van de broncode geanalyseerd. Indien er grote verschillen zijn met het C# equivalent, wordt de broncode aangepast volgens de C# standaard.

Dit is niet zo eenvoudig als het klinkt. C++ gebruikt *include* commando's, terwijl C# werkt met *using* statements. De twee talen werken met hun eigen **libraries**. Deze moeten met elkaar vergeleken worden om overeenkomsten en verschillen tussen syntax en functionaliteit te kunnen vinden. Om bijvoorbeeld in beide talen tekst naar de console te schrijven, heeft C++ het commando `std::cout` nodig en C# op zijn beurt `Console.WriteLine`.

```
#include <iostream>

int main(int argc, char **argv) {

    std::cout << "Hello world!" << std::endl;

    return 0;
}
```

→

```
using System;

static void Main(string[] args) {

    Console.WriteLine("Hello world!");
    Console.WriteLine("\n");

}
```

Men kan kiezen hoever men in de conversie gaat. Zo kan men bijvoorbeeld kiezen hoe men **pointers** converteert naar C#. Men kan ze laten staan en aanduiden met het *unsafe* keyword of trachten deze om te zetten naar reference types.

C++	C#	C# unsafe
<pre>struct Point { int x, y; }; int main() { Point p; Point *p2 = &p; p2->x = 123; std::cout << p2->x; return 0; }</pre>	<pre>public class Point { public int x, y; } static void main() { Point p = new Point(); p.x = 123; Console.WriteLine(p.x); }</pre>	<pre>struct Point { public int x, y; } unsafe static void main() { Point p = new Point(); Point *p2 = &p; p2->x = 123; Console.WriteLine(p2->x); }</pre>

Student: Martijn Saelens
Interne promotor: Leen Brouns
Externe promotor: Joris Van Maldeghem
Meer info: <http://masterproef.martenserver.com>