

# Vorlesung Softwareentwicklung 2021

<https://github.com/SebastianZug/CsharpCourse>

André Dietrich	Christoph Pooch	Fabian Bär	Fritz Apelt	Galina Rudolf
JohannaKlinke	Jonas Treumer	KoKoKotlin	Lesestein	LinaTeumer
MMachel	Sebastian Zug	Snikker123	Yannik Höll	Florian2501
		fb89zila		DEVensiv

## Dokumentation und Build-Tools

Parameter	Kursinformationen
<b>Veranstaltung</b>	Vorlesung Softwareentwicklung
<b>Semester</b>	Sommersemester 2021
<b>Hochschule:</b>	Technische Universität Freiberg
<b>Inhalte:</b>	Realisierung von Dokumentationen, Anwendung von Build-Tools
<b>Link auf den</b>	<a href="https://github.com/TUBAF-Iff-LiaScript/VL_Softwareentwicklung/blob/master/17_Dokumentation_BuildTools.md">https://github.com/TUBAF-Iff-LiaScript/VL_Softwareentwicklung/blob/master/17_Dokumentation_BuildTools.md</a>
<b>GitHub:</b>	
<b>Autoren</b>	@author

## Dokumentation

### Wer braucht schon eine Doku?

*Eine Softwaredokumentation ist mangelhaft, wenn in ihr in nennenswertem Umfang Bildschirmdialoge nicht (mehr) aktuell sind, nicht mit den im Programm vorhandenen Dialogen übereinstimmen oder gar nicht dokumentiert sind. ... Eine Softwaredokumentation ist mangelhaft, wenn sie den Anwender nicht in die Lage versetzt, die Software im Bedarfsfalle erneut oder auf einer anderen Anlage zu installieren. [LG Bonn, 19.12.2003]*

Als Softwaredokumentation bezeichnet man die Beschreibung einer Software für Entwickler, Anwender und Benutzer. Entsprechend den unterschiedlichen Rollen, wird erläutert, wie die Software funktioniert, was sie erzeugt und verarbeitet (z. B. Daten), wie sie zu benutzen ist, was zu ihrem Betrieb erforderlich ist und auf welchen Grundlagen sie entwickelt wurde.

*Klassifikation 1 - Intern/Extern ...* bezieht sich dabei auf die Frage, ob das Ganze für den internen Gebrauch oder den externen Gebrauch, also zur Weitergabe an Kunden, realisiert werden muss. Letztgenannte Variante unterliegt einer Vielzahl von rechtlichen Normierungen und Standards.

### *Klassifikation 2 - Inhalt*

Art der Dokumentation	Beschreibung
Installationsdokumentation	Beschreibung der erforderlichen Hardware und Software, mögliche Betriebssysteme und -Versionen, vorausgesetzte Software-Umgebung wie etwa Standardbibliotheken und Laufzeitsysteme. Erläuterung der Prozeduren zur Installation, außerdem zur Pflege (Updates) und De-Installation, bei kleinen Produkten eine Readme-Datei.

---

Art  
der  
Dokumentation

---

**Benutzerschnittstellen** Dokumentation des Material für die tatsächlichen Endbenutzer, etwa über die Benutzerschnittstelle. Den Anwendern kann auch die Methodendokumentation zugänglich gemacht werden, um Hintergrundinformationen und ein allgemeines Verständnis für die Funktionen der Software zu vermitteln.

**Datendokumentation** ist über Beschreibungen zu den Daten erforderlich. Es sind die Interpretation der Informationen in der realen Welt, Formate, Datentypen, Beschränkungen (Wertebereich, Größe) zu benennen. Die Datendokumentation kann oft in zwei Bereiche aufgeteilt werden: Innere Datenstrukturen, wie sie nur für Programmierer sichtbar sind und Äußere Datendokumentation für solche Datenelemente, die für Anwender sichtbar sind – von Endbenutzern einzugebende und von der Software ausgegebene Informationen. Dazu gehört auch die detaillierte Beschreibung möglicher Import-/Exportschnittstellen.

**Testdokumentation** von Testfällen, mit denen die ordnungsgemäße Funktion jeder Version des Produkts getestet werden können, sowie Verfahren und Szenarien, mit denen in der Vergangenheit erfolgreich die Richtigkeit überprüft wurde.

**Entwicklungsdokumentation** von Versionen auf Grund von Veränderungen, der jeweils zugrundegelegten Ziele und Anforderungen und der als Vorgaben benutzten Konzepte (z. B. in Lastenheften und Pflichtenheften); beteiligte Personen und Organisationseinheiten; erfolgreiche und erfolglose Entwicklungsrichtungen; Planungs- und Entscheidungsunterlagen etc.

---

Häufig fasst ein Projekt alle Arten der Dokumentation gleichermaßen zusammen. Im folgenden soll zum Beispiel die Implementierung der avrlibc für Mikrocontroller der AtTiny, AtMega und XMega Familie auf die entsprechenden Beiträge hin untersucht werden.

<https://www.nongnu.org/avr-libc/>

### *Klassifikation 3 - Autoren*

Entwickler:

- empfindet die Softwaredokumentation oft als lästiges Übel
- generiert ggf. sehr spezifische Dokumentationen ohne Anspruch auf Allgemeinverständlichkeit
- ist aber der unmittelbare Experte!

Technischer Redakteur:

- fehlendes technisches Detailwissen, dichter am Wissensstand des Kunden
- geeignetes Abstraktionsvermögen
- erfahren im Dokumentenmanagement

### **Programmiererdokumentation**

“Code is like humor. When you have to explain it, it’s bad.”

“Warum soll ich dokumentieren, es ist doch mein Code!”

“Bei einem gut geschriebenen und formatierten Code braucht man weniger zu dokumentieren.”

Denken Sie in Zielgruppen, wenn Sie die Dokumentation erstellen. Welche Hilfestellung erwartet welcher Nutzer der Implementierung? Welche Voraussetzungen können Sie annehmen?

Entsprechend differenzieren wir Zielgruppen und fragen uns welche Personenkreise wir davon bedienen wollen. Aus dieser Fragestellung ergeben sich die Schwerpunkte der Dokumentationsarbeit:

- Praktiker ... starker Bezug zur Umsetzung, benötigt Code-Kommentare, Code-Beispiele
- Systematiker ... liest zuerst einmal die Grundlagen, bemüht sich alle Hintergrundinfo zu API/Framework, zu erfassen. benötigt Architekturbeschreibung, Konzepte allgemeiner Programmieraufgaben (Error handling, Lokalisierung & Co.)
- Bedarfsleser ... situationsgetriebene Auswertung der Dokumentation, erwartet Antworten auf spezifische Fragen, benötigt Code-Kommentare, Hintergrundinformationen und Code-Beispiele

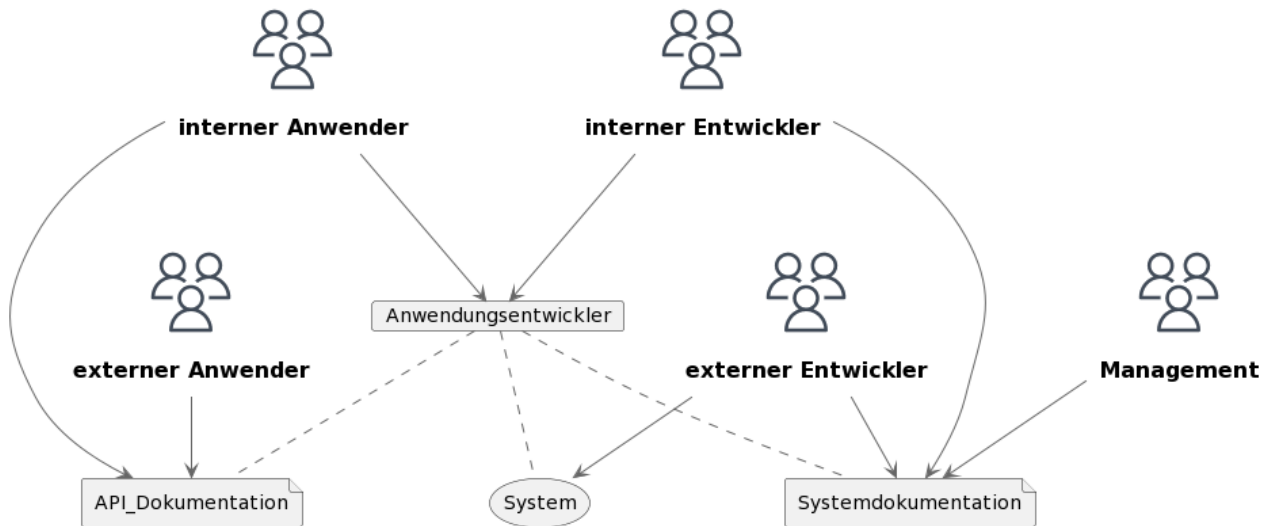


Abbildung motiviert aus <sup>1</sup>

Entsprechend ergeben sich vielfältige Dokumentationstypen, die ggf. erfasst werden sollten:

- Programmier Kochbuch
- Wiki
- Code Kommentare
- API Dokumentation

die in unterschiedlichen Formaten (online, offline, html, pdf, doc, usw.) realisiert werden können.

## Benutzerdokumentation

Auch die Benutzerdokumentation muss einer starken Zielgruppenorientierung unterliegen sowie unterschiedliche Konzepte der Handhabung einer Software beschreiben. Für Handbücher lassen sich zum Beispiel folgende Typen unterscheiden:

- Trainings-Handbuch
- Referenz-Handbuch
- Referenzkarte (auch als Cheat-Sheets bezeichnet)
- Benutzer-Leitfaden

vgl. zum Beispiel Python Pandas [Cheat Sheet](#)

## Realisierung der Dokumentation in Csharp

Merke: Anhand einer Semantik werden aus formlosen Kommentaren automatisch auswertbare Elemente einer Benutzerdokumentation!

Gliederungselemente für die Dokumentationsgenerierung sind dabei:

- Zuordnungen von Informationen zu Klassen, Methoden, Variablen
- Erläuterung von Methodensignaturen (Input/Outputs)
- Beschreibung der Funktion von Variablen, Properties usw.
- Integration von Beispielcode

Unter C# wird hinsichtlich der Darstellung zusätzlich zwischen Kommentaren mit `//` oder `/* */` und Dokumentationsinhalten unterschieden, die mit `///` eingeleitet werden.

```
using System;
```

```
// Eine zweielementige Vektorklasse ohne Methoden
public class Vector {
    public double X;
    public double Y;
    public Vector (double x, double y){
```

<sup>1</sup>Uwe Friedrichsen, Optimale Systemdokumentation mit agilen Prinzipien, 06/11, <https://www.codecentric.de/publikation/optimale-systemdokumentation-mit-agilen-prinzipien/>

```

    this.X = x;
    this.Y = y;
}
public static bool operator !=(Vector p1, Vector p2){
    // hier hatte ich keine Lust mehr
    // TODO die Methode müsste noch implementiert werden.
    return true;
}
}

/// <summary>
/// Klasse mit dem Einsprungspunkt für die Main zu Testzwecken.
/// </summary>
public class Program
{
    /// <summary>
    /// Main Funktion mit exemplarischer Initialisierung zweier Vektoren.
    /// </summary>
    public static void Main(string[] args)
    {
        Vector a = new Vector (3,4);
        Vector b = new Vector (9,6);
        Console.WriteLine (a == b);
    }
}

```

Über entsprechende Tags lassen sich den Dokumentationsfragmenten Bedeutungen geben, die eine entsprechende Gliederung und Zuordnung erlaubt:

Tag	Erläuterung
<summary>	Umfasst kurze Informationen über einen Typ oder Member.
<remarks>	Ergänzt weiterführende Informationen zu Typen und Membern.
<returns>	Beschreibt den Rückgabewert einer Methode
<value>	Beschreibt Bedeutung einer Eigenschaft
<example>	Ermöglicht mit <code> die Einbettung von (Code-)Beispielen.
<para>	Ermöglicht die Beschreibung der Eingabeparameter einer Methode
<c>	Indikator für Inline-Codefragmente
<exception>	Erlaubt die Beschreibung möglicher Exceptions, die in einer Methode auftreten können.
<see>	Klickbare Links in Verbindung mit <cref>

Was lässt sich damit umsetzen?

```

// Divides an integer by another and returns the result
// Codebeispiel aus der Microsoft Dokumentation
// siehe https://docs.microsoft.com/de-de/dotnet/csharp/codedoc

/// <summary>
/// Divides an integer <paramref name="a"/> by another integer <paramref name="b"/> and returns the result
/// </summary>
/// <returns>
/// The quotient of two integers.
/// </returns>
/// <example>
/// <code>
/// int c = Math.Divide(4, 5);
/// if (c > 1)
/// {
///     Console.WriteLine(c);
/// }
/// </code>

```

```

/// </example>
/// <exception cref="System.DivideByZeroException">Thrown when <paramref name="b"/> is equal to 0.</e
/// See <see cref="Math.Divide(double, double)"/> to divide doubles.
/// <seealso cref="Math.Add(int, int)"/>
/// <seealso cref="Math.Subtract(int, int)"/>
/// <seealso cref="Math.Multiply(int, int)"/>
/// <param name="a">An integer dividend.</param>
/// <param name="b">An integer divisor.</param>
public static int Divide(int a, int b)
{
    return a / b;
}

```

Offensichtlich bläht diese Struktur den Code, der im Beispiel aus insgesamt 4 Zeilen besteht unschön auf. Die Lesbarkeit, die ja eigentlich gesteigert werden sollte leidet darunter. Welche Lösungsmöglichkeit sehen Sie?

### Separate Dokumentationsdateien

Mit dem `<include>`-Tag lassen sich externe Dokumentationen während des Generierungsprozesses referenzieren. Damit umfasst die Dokumentation lediglich einen entsprechenden Link in Kombination mit einem einfachen Kommentar.

```

<docs>
    <members name="math">
        <Math>
            <summary>
                The main <c>Math</c> class.
                Contains all methods for performing basic math functions.
            </summary>
            <remarks>
                <para>This class can add, subtract, multiply and divide.</para>
                <para>These operations can be performed on both integers and doubles.</para>
            </remarks>
        </Math>
        <AddInt>
            <summary>
                Adds two integers <paramref name="a"/> and <paramref name="b"/> and returns the result.
            </summary>
            <returns>
                The sum of two integers.
            </returns>
        </AddInt>
        <DivideInt>
            <summary>
                Divides an integer <paramref name="a"/> by another integer <paramref name="b"/> and returns
            </summary>
            <returns>
                The quotient of two integers.
            </returns>
        </DivideInt>
    </members>
</docs>

// Adds two integers and returns the result
/// <include file='docs.xml' path='docs/members[@name="math"]/AddInt/*' />
public static int Add(int a, int b)
{
    // If any parameter is equal to the max value of an integer
    // and the other is greater than zero
    if ((a == int.MaxValue && b > 0) || (b == int.MaxValue && a > 0))
        throw new System.OverflowException();

    return a + b;
}

```

```
}
```

Leider funktioniert dieser Mechanismus unter dem gleich vorzustellen Tool Doxygen nicht.

### Konkrete Umsetzung mit C#

**Anwendung 1:** Generierung separater XML Dateien zur Verwendung in Visual Studio Code oder Visual Studio. Um die entsprechende XML Datei sollte den gleichen Namen tragen wie das Assembly und sich im gleichen Ordner befinden.

```
csc -out:MyAssembly.exe File.cs -doc:MyAssembly.xml
```

Aufbauend auf den Inhalten der XML Datei ist IntelliSense in Visual Studio dann in der Lage, die Textinformationen zu Klassen und Mitgliedern bei der Eingabe anzuzeigen.

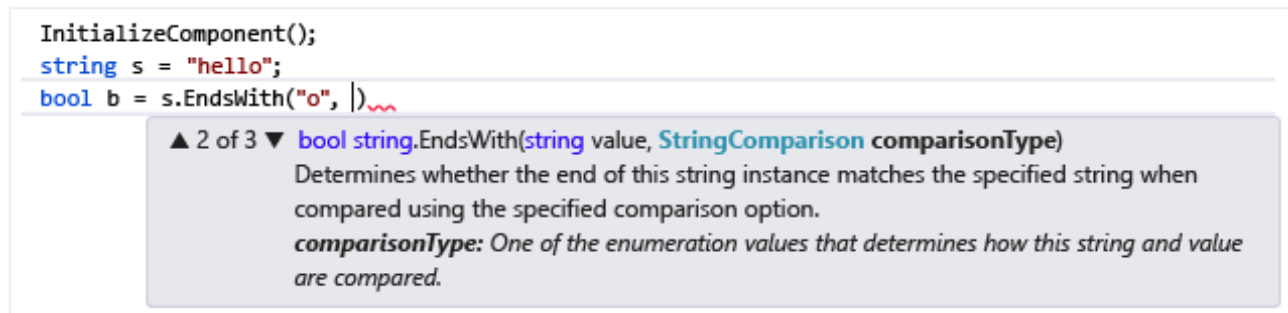


Figure 1: IntelliSense

Sie können die Parameterinformation manuell aufrufen, indem Sie STRG+UMSCHALT+LEERTASTE drücken (die alternativen Methoden mit der Maus braucht ohnehin niemand).

**Anwendung 2:** Die XML basierten Dokumentationsinhalte können in html oder pdf Dokumente transformiert werden, um eine losgelöste Dokumentation darzustellen. Hierfür können externe Tools herangezogen werden. Beispiele dafür sind [Javadoc](#), [Sphinx](#) oder [Doxygen](#). Ursprünglich bot Microsoft eine eigene Toolchain für die Code-Generierung an, diese wird gegenwärtig unter dem Projektnamen Sandcastle als Open-Source Projekt weitergeführt.

<https://github.com/EWSoftware/SHFB>

Im folgenden soll beispielhaft auf die Anwendung von Doxygen eingegangen werden.

!/?doxygenmovie

Die Anwendung von Doxygen wird im folgenden Foliensatz anhand eines Beispielprojektes gezeigt. Dabei werden zwei Verbesserungen der Darstellung realisiert.

1. Auswertung der Doxygen Ausgaben im Hinblick auf die Abdeckung der Dokumentation.
2. Einbindung des Quellcodes über das SOURCE\_BROWSER Flag.

Anpassung des entsprechenden Eintrages von NO auf YES.

```
# If the SOURCE_BROWSER tag is set to YES then a list of source files will be
# generated. Documented entities will be cross-referenced with these sources.
#
# Note: To get rid of all source code in the generated output, make sure that
# also VERBATIM_HEADERS is set to NO.
# The default value is: NO.
```

```
SOURCE_BROWSER          = NO
```

3. Integration des Projektfiles README.md in die Dokumentation.

```
INPUT                   += README.md
USE_MDFILE_AS_MAINPAGE = README.md
```

## Paketmanagement

**Merke:** Erfinde das Rad nicht neu!

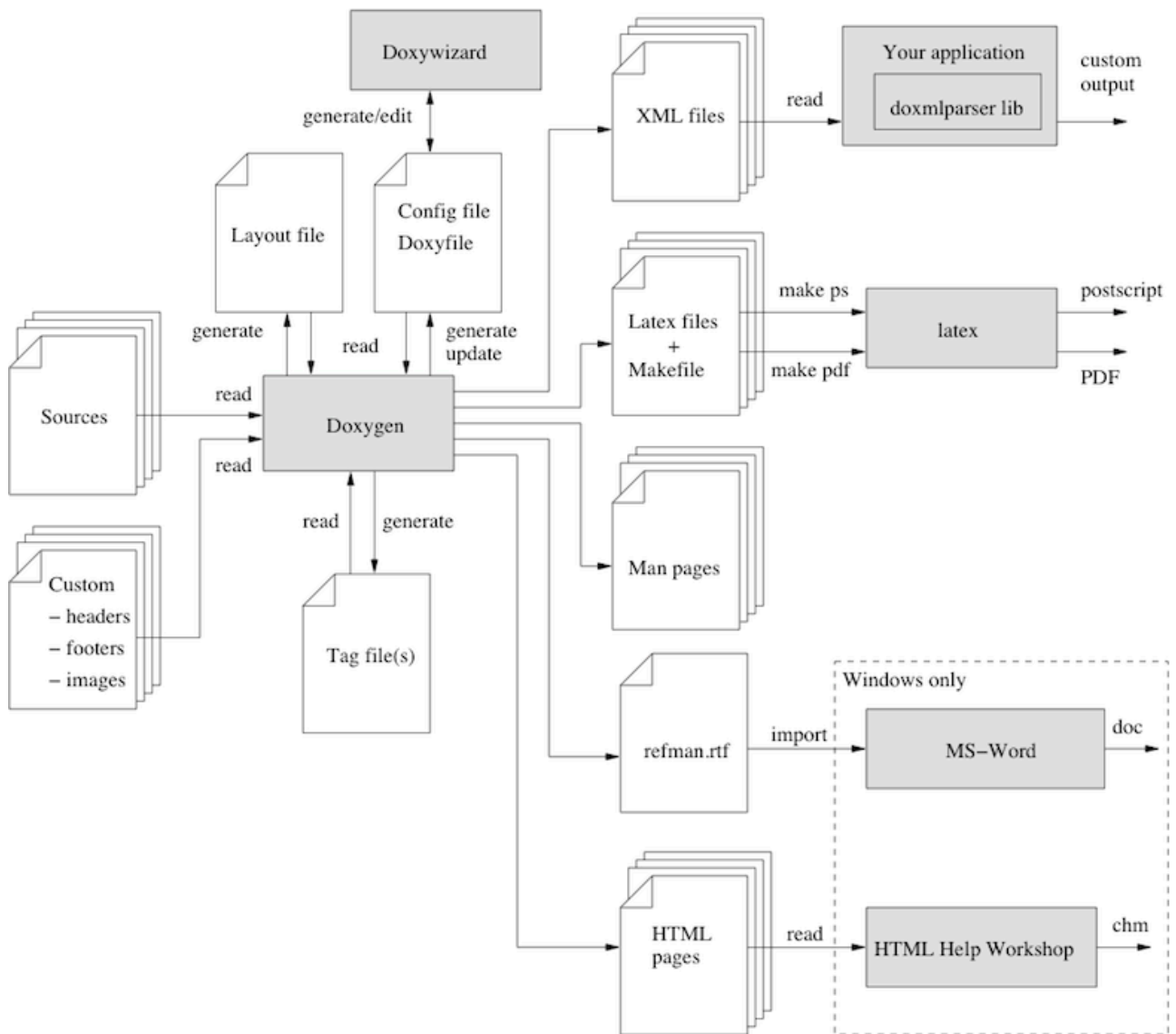


Figure 2: doxygen

Wie schaffen es erfahrene Entwickler innerhalb kürzester Zeit Prototypen mit beeindruckender Funktionalität zu entwerfen? Sicher, die Erfahrung spielt hier eine Große Rolle aber auch die Wiederverwendung von existierendem Code. Häufig wiederkehrende Aufgaben wie zum Beispiel:

- das Logging
- der Zugriff auf Datenquellen
- mathematische Operationen
- Datenkapselung und Abstraktion
- ...

werden bereits durch umfangreiche Bibliotheken implementiert und werden entsprechend nicht neu geschrieben.

Ok, dann ziehe ich mir eben die zugehörigen Repositories in mein Projekt und kann die Bibliotheken nutzen. In individuell genutzten Implementierungen mag das ein gangbarer Weg sein, aber das Wissen um die zugehörigen Abhängigkeiten - Welche Subbibliotheken und welches .NET Framework werden vorausgesetzt? - liegt so nur implizit vor.

Entsprechend brauchen wir ein Tool, mit dem wir die Abhängigkeiten UND den eigentlichen Code kombinieren und einem Projekt hinzufügen können. NuGet löst diese Aufgabe für .NET und schließt auch gleich die Mechanismen zur Freigabe von Code ein. NuGet definiert dabei, wie Pakete für .NET erstellt, gehostet und verarbeitet werden.

Ein NuGet-Paket ist eine gepackte Datei mit der Erweiterung `.nupkg` die: + den kompilierten Code (DLLs), + ein beschreibendes Manifest, in dem Informationen wie die Versionsnummer des Pakets, ggf. der Speicherort des Source Codes oder die Projektwebseite enthalten sind sowie + die Abhängigkeiten von anderen Paketen und dessen Versionen enthalten sind. Ein Entwickler, der seinen Code veröffentlichen möchte generiert die zugehörige Struktur und lädt diese auf einen NuGet Server. Unter dem [Link](#) kann dieser durchsucht werden.

### Anwendungsbeispiel: Symbolisches Lösen von Mathematischen Gleichungen

Eine entsprechende Bibliothek steht unter [Projektwebseite](#). Das Ganze wird als Nuget Paket gehostet [MathNet](#).

Unter der Annahme, dass wir `dotnet` als Buildtool benutzen ist die Einbindung denkbar einfach.

```
dotnet new console -o SymbolicMath
cd SymbolicMath
dotnet add package MathNet.Symbolics
Determining projects to restore...
Writing /tmp/tmpNsaYtc.tmp
info : Adding PackageReference for package 'MathNet.Symbolics' into project '/home/zug/Desktop/Vorlesun
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/mathnet.symbolics/index.json
...
```

Danach findet sich in unserer Projektdatei `.csproj` ein entsprechender Eintrag

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="MathNet.Symbolics" Version="0.24.0" />
  </ItemGroup>
</Project>

using System;
using System.Collections.Generic;
using MathNet.Symbolics;
using Expr = MathNet.Symbolics.SymbolicExpression; // Platzhalter für verkürzte Schreibweise

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Beispiele für die Verwendung des MathNet.Symbolics Paketes");
    }
}
```



```

var x = Expr.Variable("x");
var y = Expr.Variable("y");
var a = Expr.Variable("a");
var b = Expr.Variable("b");
var c = Expr.Variable("c");
var d = Expr.Variable("d");
Console.WriteLine("a+a =" + (a + a + a).ToString());
Console.WriteLine("(2 + 1 / x - 1) =" + (2 + 1 / x - 1).ToString());
Console.WriteLine("((a / b / (c * a)) * (c * d / a) / d) =" + ((a / b / (c * a)) * (c * d / a) / d).ToString());
Console.WriteLine("Der zugehörige Latex Code lautet " + ((a / b / (c * a)) * (c * d / a) / d).ToString());
}
}

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="MathNet.Symbolics" Version="0.24.0" />
  </ItemGroup>
</Project>

```

## Build Tools

### Konzepte

Wir haben bisher über das Compilieren des Codes und die Realisierung von Tests gesprochen, nun kommt auch noch die Erstellung einer Dokumentation hinzu ... und für all diese Teilaspekte gibt es jeweils eigne Tools. Das möchte doch niemand manuell angehen!

```

mcs program.cs
mono program.exe
doxygen Doxyfile
...

```

Wie kann man den Codeerstellungsprozess automatisieren und organisieren ohne jedes mal über eine Vielzahl von CLI-Parametern nachdenken zu müssen? Welche Aspekte sollte diese Straffung des Entwicklungsflusses abdecken:

- Auflösung der Abhängigkeiten von anderen Paketen
- Kompilierung
- Anwendung von Qualitätsmetriken
- Programmausführung
- Tests
- Generierung der Dokumentation

Dabei wäre es sinnvoll, wenn ausgehend von einer tatsächlichen Veränderung der Eingabedateien eine Realisierung des gewünschten Targets erfolgt.

Vorgang	Kompilierung	Tests	Qualitätscheck	Doku
Änderung in einer Code Datei	X	X	X	X
Hinzufügen eines Tests		X		
Anpassen einer Dokumentationsdatei				X

Um diese Idee abzubilden müssen wir offenbar Abhängigkeiten beschreiben, die ausgehend von einer Veränderung, eine bestimmte Folge von Aktionen auslösen. Ausgangspunkt für diese Aktionen können unterschiedliche Quellen sein (vgl. Generierung der Dokumentation in obiger Tabelle).

### dotnet

**dotnet** ist ein Tool für das Verwalten von .NET-Quellcode und Binärdateien. Das Programm stellt Befehle zur Verfügung, die bestimmte Aufgaben erfüllen, die zudem jeweils über eigene Argumente verfügen.

Befehl	Bedeutung
dotnet new	Initialisiert ein C#- oder F#-Projekt für eine bestimmte Vorlage.
dotnet restore	Stellt die Abhängigkeiten für eine bestimmte Anwendung wieder her.
dotnet build	Erstellt eine .NET Core-Anwendung.
dotnet clean	Bereinigen von Buildausgaben.
dotnet test	Ausführen der entsprechenden Testanwendungen

Beispielanwendung: Entwerfen Sie eine C# Programm, dass Excel-Files erstellt, für die bestimmte Felder vorinitialisiert sind.

```
export DOTNET_SKIP_FIRST_TIME_EXPERIENCE=1
dotnet --help
dotnet new console -n MyExcelGenerator
cd MyExcelGenerator
cat MyExcelGenerator.csproj
dotnet add package EPPlus
cat MyExcelGenerator.csproj
... Program.cs anpassen ... siehe Codebeispiel im Codeordner
dotnet build
dotnet run
soffice -calc myworkbook.xlsx
```

dotnet ermöglicht keine Definition von Abhängigkeiten (ohne auf MSBuild zurückzugreifen) standardisiert aber den Erstellungs- und Testprozess, sowie das Pakethandling!

## MSBuild

MSBuild ist Build-Tool, das insbesondere für das Erstellen von .NET-basierten Anwendungen genutzt wird. Microsofts Visual Studio ist in wesentlichem Maße von MSBuild abhängig; MSBuild selbst ist aber nicht von Visual Studio abhängig. Dadurch lassen sich mit MSBuild auch Visual-Studio-Projekte ohne den Einsatz von Visual Studio bauen.

Im Wesentlichen besteht MSBuild aus der Datei `msbuild.exe` und dll-Dateien, die auch im .NET Framework enthalten sind, und XML-Schemas, nach deren Vorgaben die von `msbuild.exe` verwendeten Projektdateien aufgebaut sind. Wegen der XML-Basiertheit wird MSBuild auch als Auszeichnungssprache eingeordnet.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="A">
    <Message Text="Hello World!"/>
  </Target>
</Project>
```

Innerhalb der xml-Struktur definieren Sie sogenannte Targets als Einsprungpunkte für den Erstellungsprozess. Im Beispiel sind dies zunächst nur HelloWorld-Ausgaben, im Weiteren wurde dies auf konkrete Kompilierungsvorgänge ausgeweitet.

Ein spezifisches Target kann mit `msbuild filename /t:targetname` aufgerufen werden.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="A">
    <Message Text="Hello World! A"/>
  </Target>
  <Target Name="B" DependsOnTargets="A">
    <Message Text="Hello World! B"/>
  </Target>
</Project>
```

Ein minimales Konfigurationsfile für die Build-Prozess einer einzelnen C# Datei könnte folgende Konfiguration haben:

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup>
    <Compile Include="helloworld.cs" />
  </ItemGroup>
```

```

    <Target Name="Build">
      <Csc Sources="@ (Compile)"/>
    </Target>
  </Project>

```

Ein Beispiel für eigene Experimente findet sich unter 'code/17\_ToolChain/msbuildProject'

Die zuvor besprochenen dotnet Befehle bauen auf MSBuild auf und kapseln diese. Die Ausführung von **dotnet build** entspricht **dotnet msbuild -restore -target:Build**.

Arbeiten Sie auch die Dokumentation von MSBuild durch, diese stellt auch das umfangreiche Featureset (vordefinierte Targets, integrierte Tools, die Möglichkeit externe Anwendungen einzubetten) vor, das deutlich über die Beispiele hinausgeht.

<https://docs.microsoft.com/de-de/visualstudio/msbuild/msbuild?view=vs-2019>

## Make

**make** wird beispielsweise, um in Projekten, die aus vielen verschiedenen Dateien mit Quellcode bestehen, automatisiert alle Arbeitsschritte (Übersetzung, Linken, Dateien kopieren etc.) zu steuern, bis hin zum fertigen, ausführbaren Programm.

**make** liest ein sogenanntes *Makefile* (ACHTUNG Großschreibung erforderlich) und realisiert den beschriebenen Übersetzungsprozesses entsprechend der Abhängigkeiten.

```

A: B
   generate_A

```

```

B: D E
   generate_B

```

Daneben können Sie entsprechende Makros `$(MAKRO_NAME)`, `wildcard` und Platzhalter verwenden, um die Beschreibung effizienter und kompakter zu gestalten. Ein typisches Makefile für eine eingebettetes C Projekt (Arduino) stellt sich wie folgt dar:

```

# Source, Executable, Includes, Library Defines
INCL  = loop.h defs.h
SRC    = a.c b.c d.c      # c Code-Dateien
OBJ    = $(SRC:.c=.o)
LIBS   = -lgen
EXE    = program

# Compiler, Linker Defines
CC      = /usr/bin/gcc
CFLAGS  = -ansi -pedantic -Wall -O2
LIBPATH = -L.
LD_FLAGS = -o $(EXE) $(LIBPATH) $(LIBS)
RM       = /bin/rm -f

# Compile and Assemble C Source Files into Object Files
%.o: %.c
    $(CC) -c $(CFLAGS) $*.c

# Link all Object Files with external Libraries into Binaries
$(EXE): $(OBJ)
    $(CC) $(LD_FLAGS) $(OBJ)

# Objects depend on these Libraries
$(OBJ): $(INCL)

# Clean Up Objects, Executables, Dumps out of source directory
clean:
    $(RM) $(OBJ) $(EXE) core a.out

```

MERKE: Die Einschübe im MAKEFILE sind keine Leerzeichen, sondern Tabulatorshifts!  
Das Hilfsprogramm make ist Teil des POSIX-Standards.