



Media Engineering Programming Tools

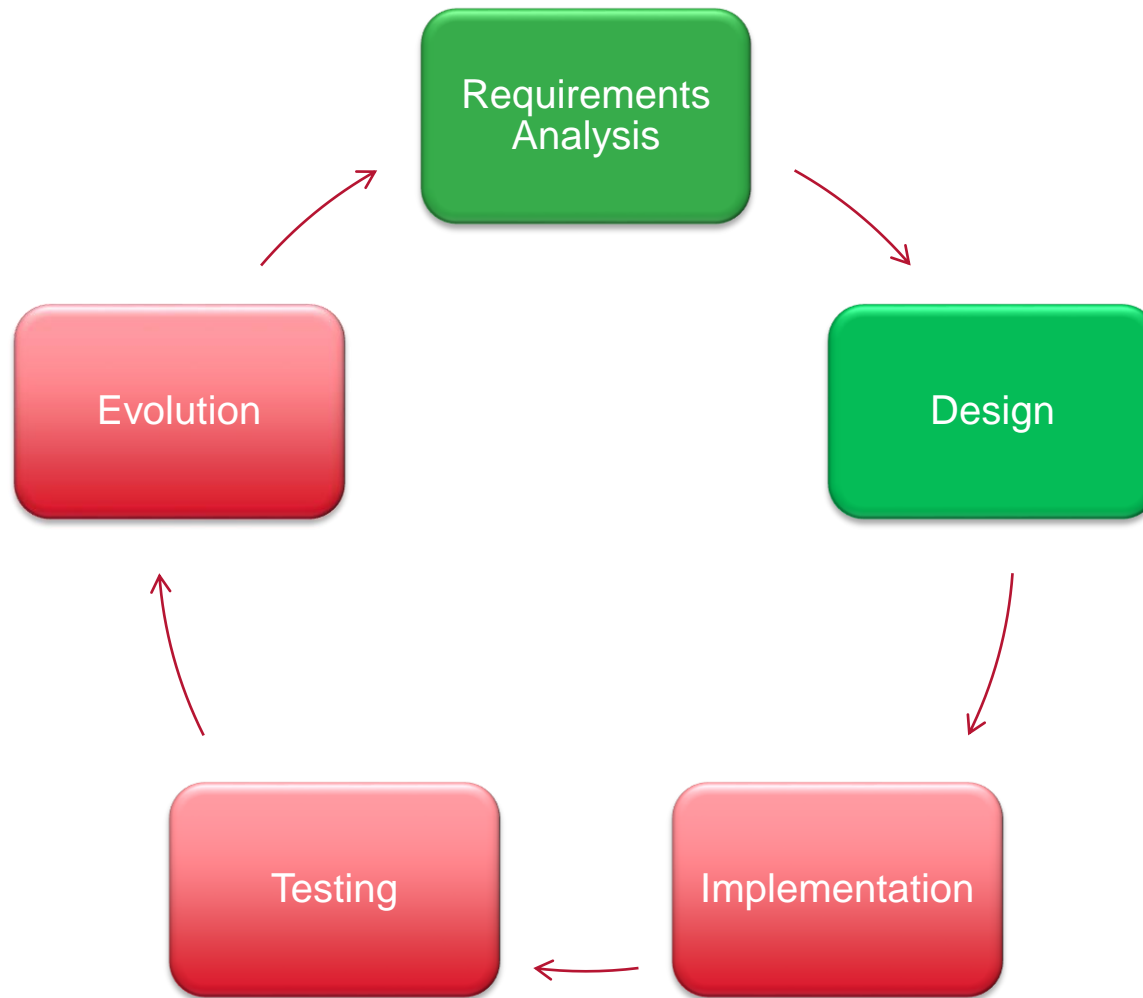


R. Weller

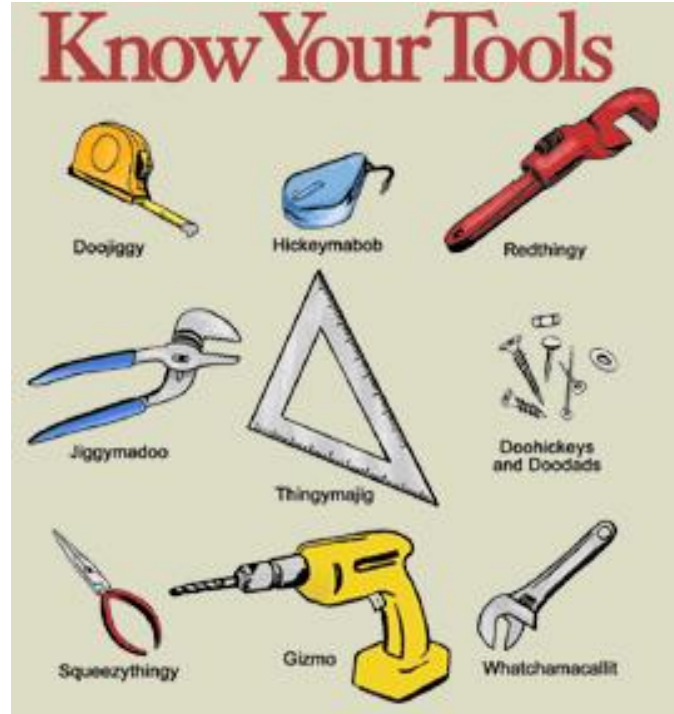
University of Bremen, Germany

cgvr.cs.uni-bremen.de

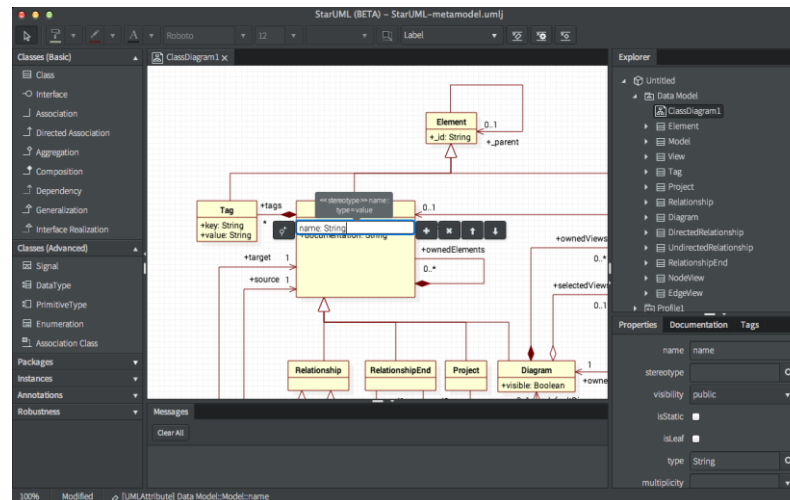
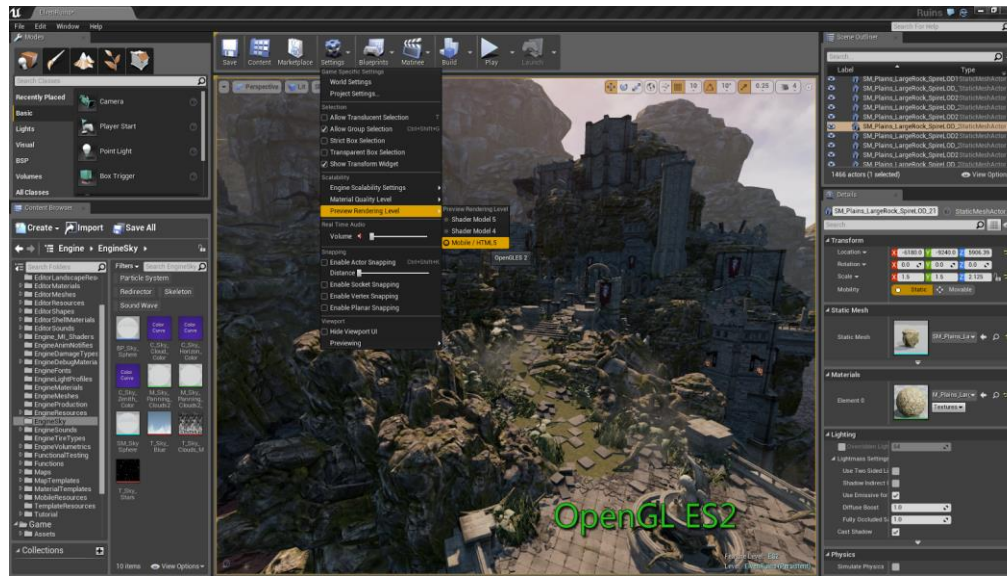
Der Software Development-Lifecycle



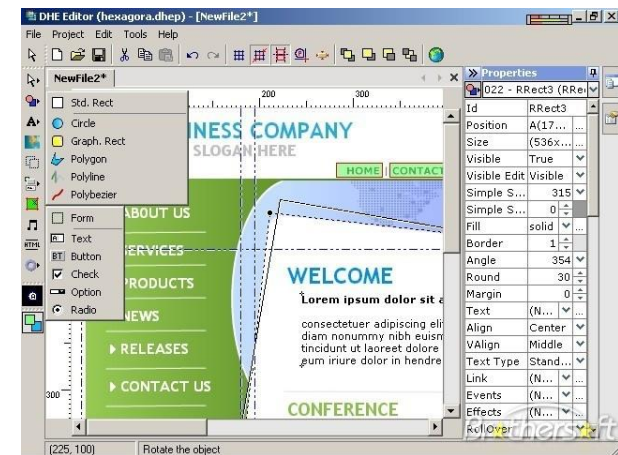
Also endlich?



Einige Tools kennen wir schon



Viele Werkzeuge für digitale Medien



■ In dieser Vorlesung beschränken wir uns auf **Programmierwerkzeuge**

Programmierung – Die Anfänge

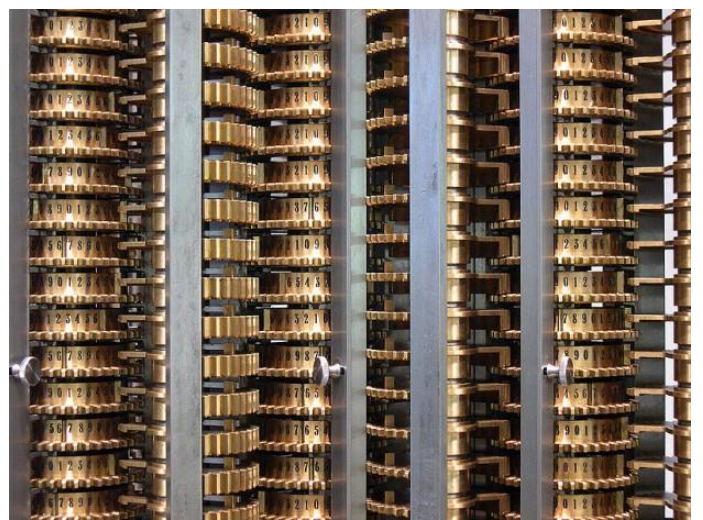
Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 222 of seq.)

Number of Operations	Statement of Results	Definition of changes to the value of any Variable	Initial	Working Variables										Result Variables					
				V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	R_1	R_2	R_3	R_4		
1	$V_1 = 1, V_2 = 0, V_3 = 0, V_4 = 0, V_5 = 0, V_6 = 0, V_7 = 0, V_8 = 0, V_9 = 0, V_{10} = 0$		1																
2	$V_1 = 1, V_2 = 1, V_3 = 0, V_4 = 0, V_5 = 0, V_6 = 0, V_7 = 0, V_8 = 0, V_9 = 0, V_{10} = 0$	$V_2 = V_1 + 1$	1																
3	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 0, V_5 = 0, V_6 = 0, V_7 = 0, V_8 = 0, V_9 = 0, V_{10} = 0$	$V_3 = V_2 + 1$	1																
4	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 0, V_6 = 0, V_7 = 0, V_8 = 0, V_9 = 0, V_{10} = 0$	$V_4 = V_3 + 1$	1																
5	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 0, V_7 = 0, V_8 = 0, V_9 = 0, V_{10} = 0$	$V_5 = V_4 + 1$	1																
6	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 0, V_8 = 0, V_9 = 0, V_{10} = 0$	$V_6 = V_5 + 1$	1																
7	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 0, V_9 = 0, V_{10} = 0$	$V_7 = V_6 + 1$	1																
8	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 0, V_{10} = 0$	$V_8 = V_7 + 1$	1																
9	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 0$	$V_9 = V_8 + 1$	1																
10	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$V_{10} = V_9 + 1$	1																
11	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_1 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
12	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_2 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
13	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_3 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
14	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_4 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
15	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_5 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
16	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_6 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
17	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_7 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
18	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_8 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
19	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_9 = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																
20	$V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 1, V_6 = 1, V_7 = 1, V_8 = 1, V_9 = 1, V_{10} = 1$	$R_{10} = V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10}$	1																

Here follows a repetition of Operations (lines 1 to twenty-one).

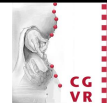


Ada Lovelace (1815 – 1852)





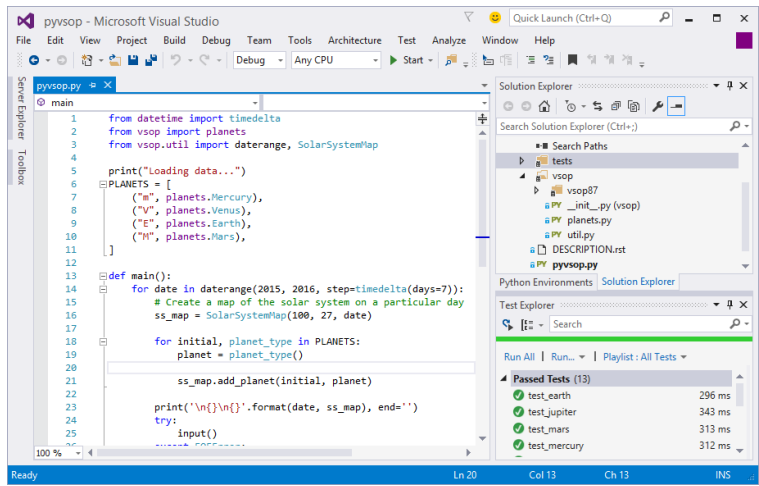
Programmierung im Wandel der Zeit



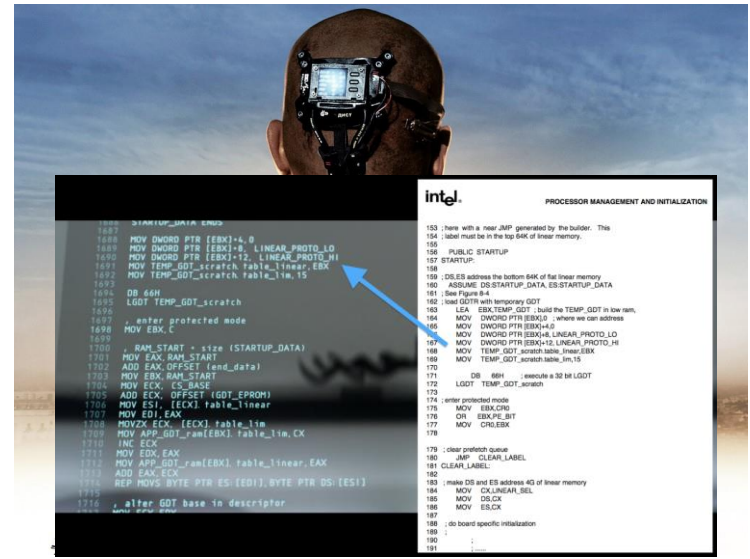
1969



1980



Heute



2154?

Weg zum Ausführbaren Programm

■ Quellcode

■ In C++:

- Header als *.h
- Implementation als *.cpp – Dateien speichern

```
void CHyfic30bView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CHyfic30bDoc* pDoc = GetDocument();
    if (a_tracker.HitTest(point) == CRectTracker::hitMiddle) {
        CObjectSource* pSource = SaveDib();
        if (pSource) {
            // DoDragDrop returns only after drop is complete
            CClientDC dc(this);
            OnPrepareDC(&dc);
            CPoint topLeft = a_rectTracker.TopLeft();
            dc.DPtoLP(&topLeft);
            // 'point' here is not the same as the point parameter in
            // OnDragEnter, so we use this one to compute the offset
            a_dragOffset = point - topLeft; // device coordinates
            pDoc->a_bDragHere = TRUE;
            DROPEFFECT dropEffect = pSource->DoDragDrop(
                DROPEFFECT_MOVE | DROPEFFECT_COPY, CRect(0, 0, 0, 0));
            TRACE("after DoDragDrop -- dropEffect = %ld\n", dropEffect);
            if (dropEffect == DROPEFFECT_MOVE && pDoc->a_bDragHere) {
                pDoc->OnEditClearAll();
            }
            pDoc->a_bDragHere = FALSE;
            delete pSource;
        }
    }
    else {
        if (a_tracker.Track(this, point, FALSE, NULL)) {
            CClientDC dc(this);
            OnPrepareDC(&dc);
            // should have some way to prevent it going out of bounds
            a_rectTracker = a_tracker; a_rect;
            dc.DPtoLP(a_rectTracker); // Update logical coords
        }
    }
    Invalidate();
}
```

■ Compiler

- Übersetzt Quellcode in maschinenlesbare Sprache



■ Linker

- Fasst mehrere kompilierte Module zu Gesamtprogramm zusammen
- Symbolische Adressen von Funktionen oder Variablen werden in Speicheradressen umgewandelt



- Features
 - Syntax-Highlighting
 - Suchen und Ersetzen
 - Code-Faltung
 - Codevervollständigung
 - Calltips
 - Automatische Einrückung
 - Symbolbrowser
 - Makros

```

=====
idSmokeParticles::Init
=====
void idSmokeParticles::Init( void ) {
    if ( !initialized ) {
        Shutdown();
    }

    // set up the free list
    for ( int i = 0; i < MAX_SMOKE_PARTICLES-1; i++ ) {
        smokes[i].next = &smokes[i+1];
    }
    smokes[MAX_SMOKE_PARTICLES-1].next = NULL;
    freeSmokes = &smokes[0];
    numActiveSmokes = 0;

    activeStages.Clear();

    memset( &renderEntity, 0, sizeof( renderEntity ) );

    renderEntity.bounds.Clear();
    renderEntity.axis = Mat3_Identity();
    renderEntity.shaderParams[ SHADERPARAM_RED ] = 1;
    renderEntity.shaderParams[ SHADERPARAM_GREEN ] = 1;
    renderEntity.shaderParams[ SHADERPARAM_BLUE ] = 1;
    renderEntity.shaderParams[3] = 1;

    renderEntity.Model = renderModeManager->AllocModel();
    renderEntity.Model->InitEmpty( smokeParticle_SnapshotName );

    // we certainly don't want particle shadows
    renderEntity.noShadow = 1;

    // huge bounds, so it will be present in every world area
    renderEntity.bounds.AddPoint( idVec3( 100000, -100000, -100000 ) );
    renderEntity.bounds.AddPoint( idVec3( 100000, 100000, 100000 ) );

    renderEntity.callback = idSmokeParticles::ModelCallback;
    // add to renderer list
    renderEntity.Handle = gameRenderWorld->AddEntityDef( &renderEntity );

    currentParticleTime = -1;

    initialized = true;
}
    
```

```

=====
idSmokeParticles::Init
=====
void idSmokeParticles::Init( void ) {
    if ( !initialized ) {
        Shutdown();
    }

    // set up the free list
    for ( int i = 0; i < MAX_SMOKE_PARTICLES-1; i++ ) {
        smokes[i].next = &smokes[i+1];
    }
    smokes[MAX_SMOKE_PARTICLES-1].next = NULL;
    freeSmokes = &smokes[0];
    numActiveSmokes = 0;

    activeStages.Clear();

    memset( &renderEntity, 0, sizeof( renderEntity ) );

    renderEntity.bounds.Clear();
    renderEntity.axis = Mat3_Identity();
    renderEntity.shaderParams[ SHADERPARAM_RED ] = 1;
    renderEntity.shaderParams[ SHADERPARAM_GREEN ] = 1;
    renderEntity.shaderParams[ SHADERPARAM_BLUE ] = 1;
    renderEntity.shaderParams[3] = 1;

    renderEntity.Model = renderModeManager->AllocModel();
    renderEntity.Model->InitEmpty( smokeParticle_SnapshotName );

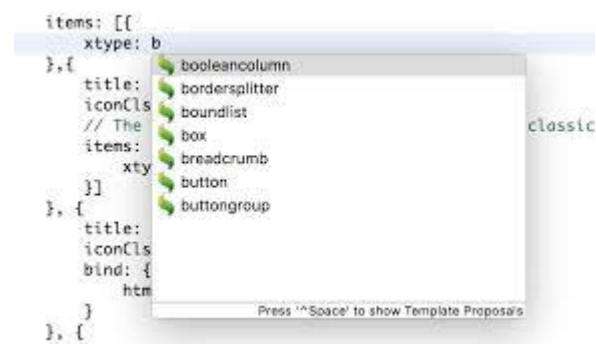
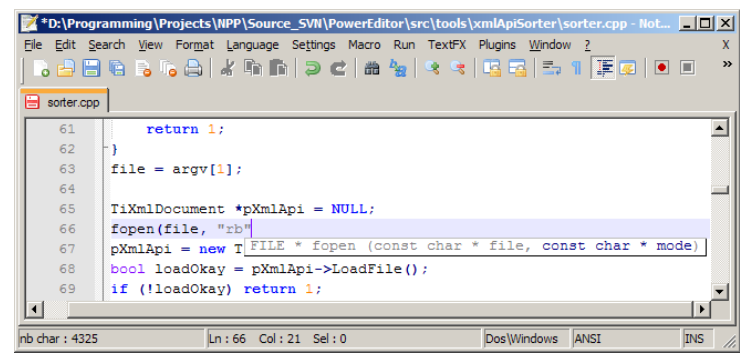
    // we certainly don't want particle shadows
    renderEntity.noShadow = 1;

    // huge bounds, so it will be present in every world area
    renderEntity.bounds.AddPoint( idVec3( 100000, -100000, -100000 ) );
    renderEntity.bounds.AddPoint( idVec3( 100000, 100000, 100000 ) );

    renderEntity.callback = idSmokeParticles::ModelCallback;
    // add to renderer list
    renderEntity.Handle = gameRenderWorld->AddEntityDef( &renderEntity );

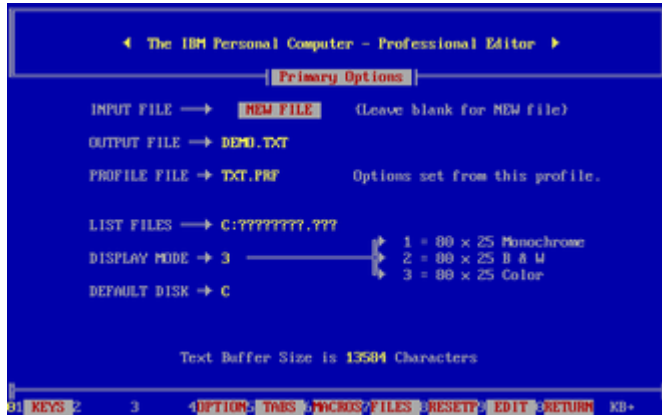
    currentParticleTime = -1;

    initialized = true;
}
    
```

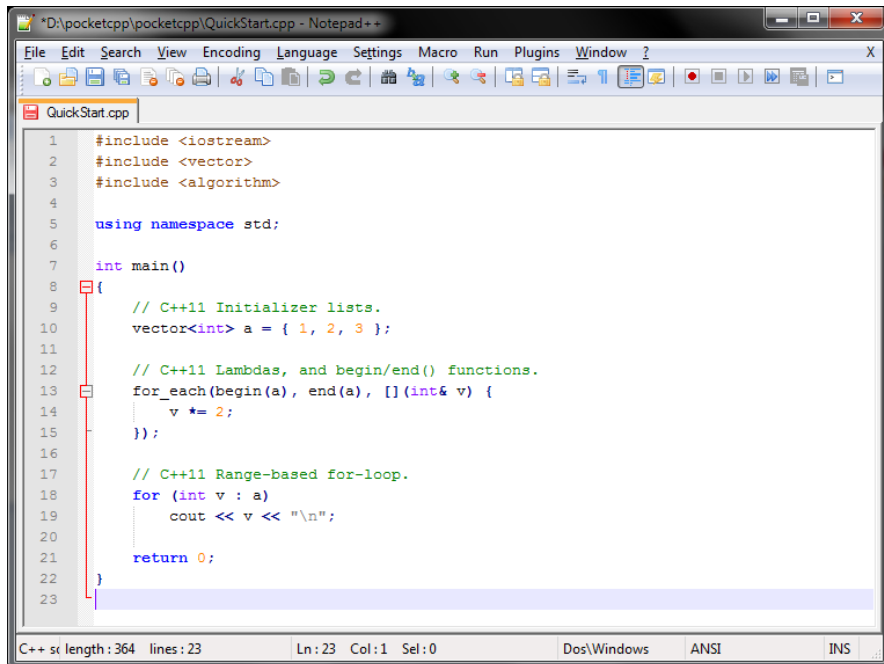




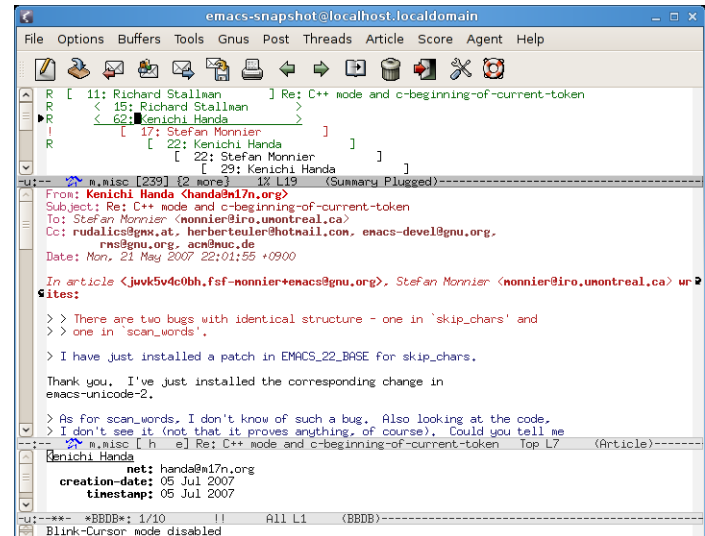
Einige Quellcode-Editoren



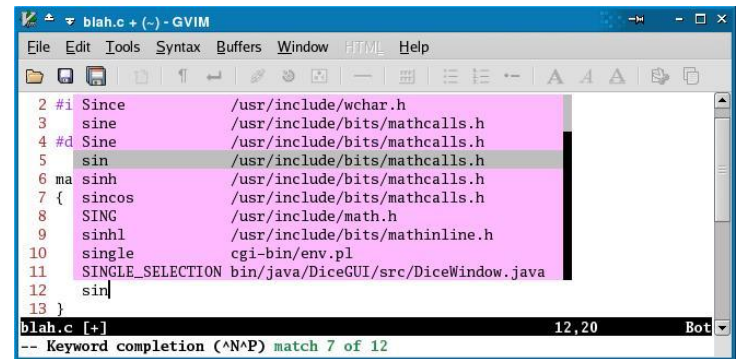
IBM Professional Editor



Notepad++



Emacs



Vi

Warum braucht man schönen Code?

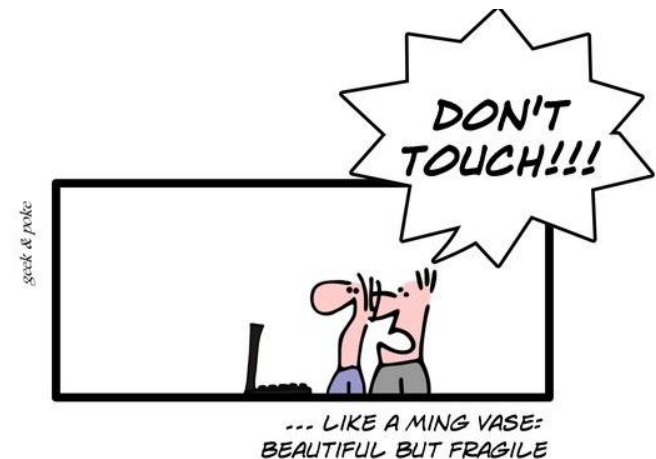
- „Schönen“ Code schreibt man nicht (nur) für sich selbst
 - Programmieren heute meist Teamarbeit
 - Kollegen müssen den Code auch lesen
 - Wiederverwendbarkeit
 - Eventuell will man Codeteile in Zukunft in anderen Projekten wieder verwenden
 - Wartung
 - 80% der Lebenszeit entfallen auf Wartung
 - Programmierer warten selten selbst



Wann ist Code schön?

- Ziel: Der perfekte Code ist
 - Lesbar
 - Verständlich
 - Wartbar
- Elemente guten **Programmierstils**
 - Festlegung von Namenskonventionen
 - Benennung von Klassen, Variablen, Funktionen
 - Code-Strukturierung
 - Einrückungen
 - Leerzeichen
 - Zeilenanzahl
 - Dokumentation
 - Kommentare

GOOD CODE IS...



Standards?

- Einige Qualitätsnormen im Softwarebereich fordern die explizite Anwendung von Regelwerken
 - Beispiel: MISRA-C (1998)
- Tatsächlich hängen die Regeln von vielen Faktoren ab
 - Programmiersprache
 - Projektgröße
 - Historische Entwicklung
 - ...
- Beispiele
 - Google Java Style (19 Seiten)
 - GNU Coding Standards (85 Seiten)
 - Linux Kernel Coding Style (3 Seiten)
 - CGVR-Programmierrichtlinien



Bekannte Namenskonventionen



- Ungarische Notation
 - Entwickelt von Charles Simonyi
 - Lange Zeit von Microsoft verwendet
 - Beispiele:
 - Variablennamen setzen sich aus (Präfix) (Datentyp) (Bezeichner) zusammen
 - ichArray, uuluwchArray
 - Namen wegen Herkunft des Erfinders und des exotischen Aussehens
 - Linus Torwalds Meinung: „Encoding the type of a function into the name (so-called Hungarian notation) is brain damaged—the compiler knows the types anyway and can check those, and it only confuses the programmer”
- Ähnliche Konventionen
 - Reddick-Namenskonvention (Microsoft .NET)
 - Leszynski-Namenskonvention (Microsoft Access)

Art	Beispiel	Erläuterung
Lokale Variablen	mylocalvar my_local_var	Alles klein
Instanzvariablen	m_var	m für „member“
Klassennamen	MyCoolClass	Anfang Groß (ohne „C“ für Klasse)
Methoden	calcSomeCoolValue()	inCaps, klein anfangen
Konstanten	const int MaxNum = 10;	Anfang groß
Defines	#define MY_DEFINE	all-caps mit Underscores
Typedef	SomethingT	Wie Klassen mit Suffix „T“
Enum-Typen	myEnumTypeE	Wie Variablen, Suffix E
Enum-Member	COL_TRUE, COL_FALSE	Wie Defines
Template-Parameter	<MyTypeT>	Wie Klassennamen, Suffix T

CGVR-Konventionen für Einrückungen

- Allman/“East Coast“-Style
- Einrückung: 4 Leerzeichen
 - Keine Tabulatoren verwenden
- Klammern beginnen in eigener Zeile
 - Leichtere Zuordnung

```

for( ... )
{
    if( ... )
    {
        ...
    }
    else
    {
        ...
    }
}
    
```

- Nicht K&R-Style verwenden
 - Vorteil: Code kompakter
 - Aber bei heutiger Monitorauflösung kein Argument mehr
 - Nachteil: Öffnende Klammern werden leicht übersehen

```

for ( ... ) {
    if ( .. ) {
        ...
    } else {
        ...
    }
} else {
    ...
}
    
```


- Nicht mit Leerzeichen sparen

```
for( i = obj->begin(); i < obj->l() && obj->M(i) != -1; i++ )
    obj->M(i) = -1;
}
```

```
for( i = obj->begin(); i < obj->l() && obj->f(i) != -1; i++ )
{
    obj->f(i) = -1;
}
```

- Maximale Zeilenlänge: 80 Zeichen
 - Kürzere Zeilen leichter lesbar
 - Vergleiche mehrspaltigen Satz in Zeitungen
 - Formatierung bleibt beim Ausdrucken vorhanden
 - Bei zu großer Schachtelungstiefe eher in Funktionen auslagern
 - Verhindert zu lange Namen
 - Namenswahl extrem wichtig, lang Klassennamen deuten oft auf Designfehler hin

■ Klassenlayout

```
class MyClass
{
    public:
        1. Konstanten
        2. Typen
        3. Variablen
        4. Methoden

    protected:
        ...

    private:
        ...

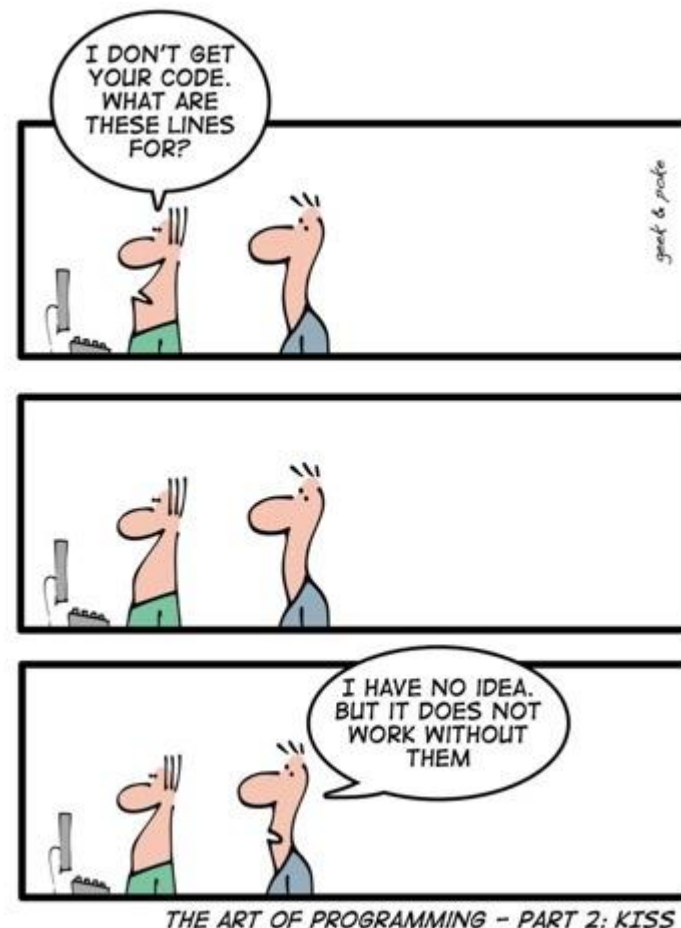
}
```

■ Tabulieren von Variablen

```
int          x, y;           // dominant coord planes of polygon
int          xturns, yturns; // # turns of xslope, yslope
objPolyhedronP  p1, p2;
int          i;
```

```
int          x, y;           // dominant coord planes of polygon
int          xturns, yturns; // # turns of xslope, yslope
objPolyhedronP  p1, p2;
int          i;
```

- Mehrere Funktionen
 - Hilft eigene Gedanken zu ordnen (und damit sauberer zu programmieren)
 - Hilft anderen zu verstehen, was passiert
 - Code ist nicht selbsterklärend!
- Wie man es macht und wie nicht
 - „Später“ kommt nie!
 - Optimalerweise dann kommentieren, wenn Funktion „halb“-fertig ist
 - Gedanken noch frisch
 - Anschließend nochmal überprüfen



Arten von Kommentare

- Grundsätzlich 4 Arten von Kommentaren
 - Am Anfange von Klassen
 - Adressat: Verwender der Klasse
 - Big Picture
 - Besonderheiten (z.B. Compiler-Flags)
 - Vor Funktionen
 - Beschreibung
 - Parameter (besonders Ausgabeparameter kennzeichnen)
 - Pre-/Postconditions
 - Seiteneffekte
 - Vor Variablen
 - Immer vor globalen Variablen
 - Zwischen Code-Abschnitten
 - Beschreiben **was** gemacht wird, nicht **wie**

Beispiel: Kommentierung einer Funktion

- Diesen Header vor jede Funktion kopieren und ausfüllen!

```

/** Do something (Einzeiler)
 *
 * @param param1    blubber (in)
 * @param param2    bla (out)
 *
 * @return
 *   Beschreibung der Rückgabe-Werte, z.B.
 *   -1 falls fehlgeschlagen, 0 wenn alles ok.
 *
 * Ausführliche, mehrzeilige Beschreibung, z.B.
 * Diese Funktion berechnet ...
 * Basiert auf dem Algorithmus von ...
 *
 * @throw Exception
 *   Genaue Beschreibung der Exceptions und deren Bedingungen, z.B.
 *   XOutOfMemory, falls kein Speicher mehr verfügbar.
 *   XCoffee, falls kein Kaffee mehr da.
 *
 * @warning
 *   Beschreibung von Eingabe-Parameter-Werten oder Zuständen,
 *   die zu Fehlern oder Abstürzen führen, die aber nicht abgefangen werden.
 *   (sollte nur sehr selten vorkommen!)

```

```
* @pre
*   Genaue Beschreibung der Vorbedingungen, z.B.
*   Erwartet, dass die Funktion init() schon aufgerufen wurde.
*   Param1 muss von der Funktion blub() berechnet worden sein.
*
* @sideeffects
*   Nebenwirkungen, globale Variablen, die verändert werden, ..., z.B.
*   @arg The global variable @c M_Interest will be modified.
*
* @todo
*   Schneller machen.
*
* @bug
*   Produziert einen core dump, wenn @a param1 = 0.0 ist.
*
* @internal
*
* @see
*   Querverweise auf andere Funktionen, z.B., weil sie etwas ähnliches berechnen,
*   oder weil es irgend eine Abhängigkeit zwischen beiden gibt. Beispiel:
*   eineAndereFunktion()
*
**/
```

Software-Dokumentationswerkzeuge

- Automatische Generierung von Dokumenten aus verschiedenen Quellen
 - Quelltext
 - UML-Diagrammen
- Export als browsbare HTML-, PDF- oder Epub-Dateien
- Beispiele:
 - Doxygen (C++)
 - Javadoc (Java)



The screenshot shows the Doxygen GUI frontend window titled "Doxygen GUI frontend +". The menu bar includes "File", "Settings", and "Help".

Step 1: Specify the working directory from which doxygen will run

Working directory:

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Wizard | Expert | Run

Topics

- Project
- Mode
- Output
- Diagrams

Provide some information about the project you are documenting

Project name:

Project synopsis:

Project version or id:

Project logo: No Project logo selected.

Specify the directory to scan for source code

Source code directory:

Scan recursively

Specify the directory where doxygen should put the generated documentation

Destination directory:

- Beautifier (auch Quelltext-Formatierer)
 - Formatieren vorhandenen Quelltext gemäß voreingestellter Regeln
 - Beispiel: Prettyprint
- Style Checker
 - Überprüft die Einhaltung eines vorher definierten Programmierstils
 - Führt im Gegensatz zum Beautifier aber meist keine Umformatierung durch
 - Unterstützen oft auch eine (sehr eingeschränkte) semantische Analyse

```

int
foo (int k)
{
    if (k < 1 || k > 2)
    {
        printf ("out of range\n");
        printf ("this function requires of 1 or 2\ ");
    }
    else
    {
        printf ("Switching\n");
        switch (k)
        {
            case 1:
                printf ("1\n");
                break;
            case 2:
                printf ("2\n");
                break;
        }
    }
}
    
```

```

int foo(int k){if(k<1||k>2){printf("out of range\n");
printf("this function requires a value of 1 or 2\n");}else{
printf("Switching\n");switch(k){case 1:printf("1\n");break;case 2:printf("2\n");break;}}}
    
```

Das Gegenteil – Code Obfuscators

- Macht das Gegenteil vom Beautifier
- Ziel: Soll Reverse-Engineering verhindern
 - Security-by-Obscurity
- Beispiel

```

void _ (int __,int ____,int _____,int _____){((
__ / __ )<= _____ )? _ (__,__+_____,_____,_____ ):!
(____ % __) ? _ (__,__+_____,_____% __, _____ ):(
(____ % __)==(____ / __) && !_____)?(printf("%d "
, (____ / __)), _ (__,__+_____,_____,_____ )):((____
%__)>_____ &&(_____% __) < (____ / __))?(
____,____+_____,_____+!((____ / __)% (____% _____)),
_____):(____<_*__)?_(__,____+_____,
_____,_____):0;} int main(void){_(50,0,0,1 );}
    
```

- Speichern aller *.cpp und *.h-Dateien
- Aufruf des Compilers
 - Dieser erzeugt *.obj-Dateien als Zwischenergebnis
- Aufruf des Linkers
 - Zusammenfassen der *.obj-Dateien (und eventueller externer Abhängigkeiten) zu ausführbarem Programm

```
E:\Zhenya\Program\menuet\article_h11\examples\uc\hello>cl /c /O2 /nologo /GS- /GR- hello.cpp kosFile.cpp kosSyst.cpp mcsmemm.cpp
hello.cpp
kosFile.cpp
kosSyst.cpp
mcsmemm.cpp
Generating Code...

E:\Zhenya\Program\menuet\article_h11\examples\uc\hello>link /nologo /manifest:no
/entry:crtStartup /subsystem:native /base:0 /fixed /align:16 /nodefaultlib hell
o.obj kosFile.obj kosSyst.obj mcsmemm.obj
LINK : warning LNK4108: /ALIGN specified without /DRIVER; image may not run

E:\Zhenya\Program\menuet\article_h11\examples\uc\hello>pe2kos hello.exe hello

E:\Zhenya\Program\menuet\article_h11\examples\uc\hello>
```

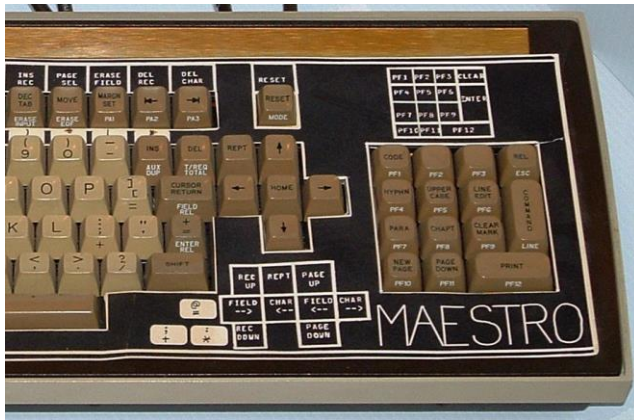


Integrierte Entwicklungsumgebungen (IDEs)

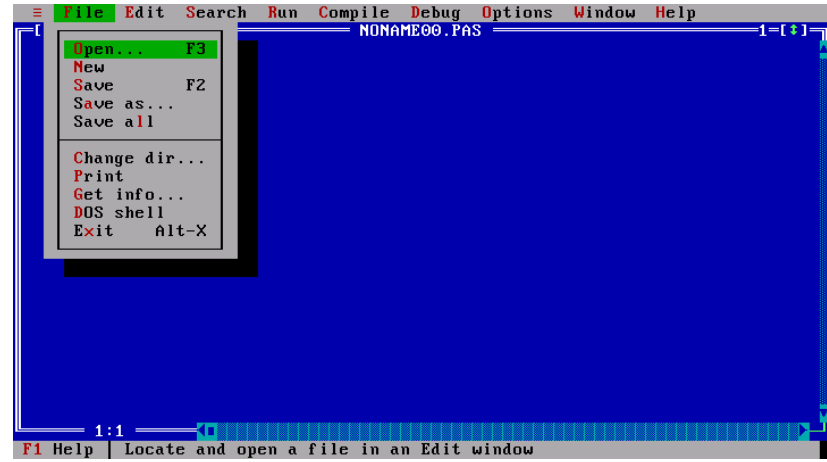
- Quellcode-Editor
- Integrierter (automatisierter) Aufruf von Compiler und Linker
 - Je nach unterstützten Sprachen auch Interpreter
- Oft noch zusätzliche Features:
 - Integrierter Debugger
 - Refactoring
 - Versionsverwaltung
 - Profiling
 - Projektmanagementtools
 - UML-Editoren
 - Werkzeuge für das GUI-Design
 - ...



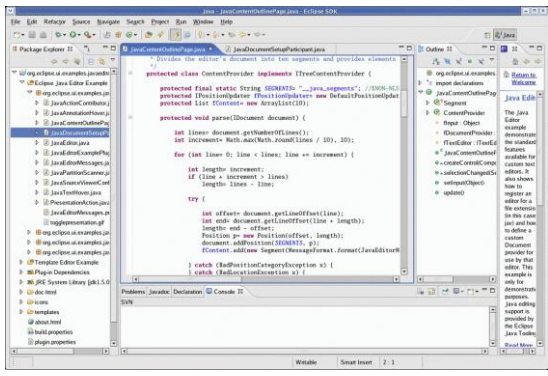
IDEs - Damals und Heute



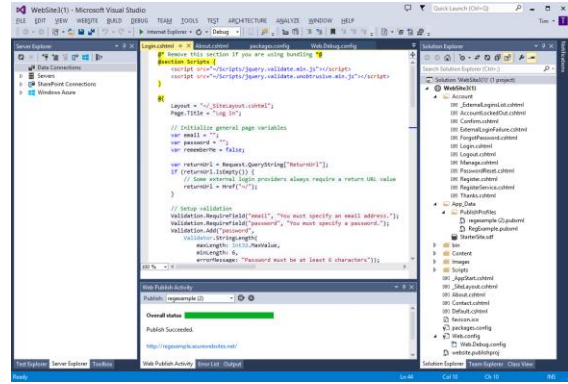
Maestro I (1975)



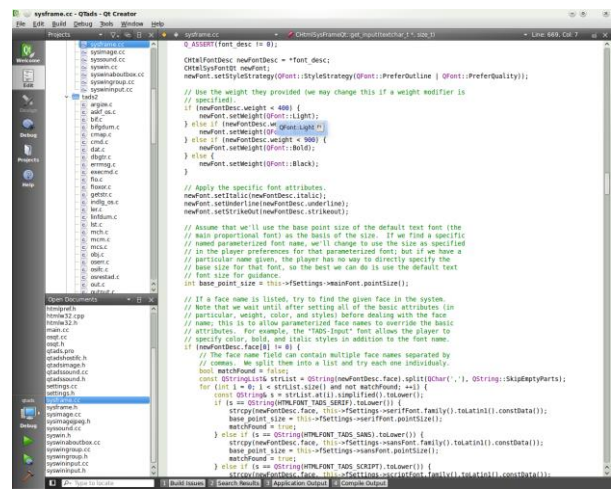
Borland Turbo Pascal (1983)



Eclipse



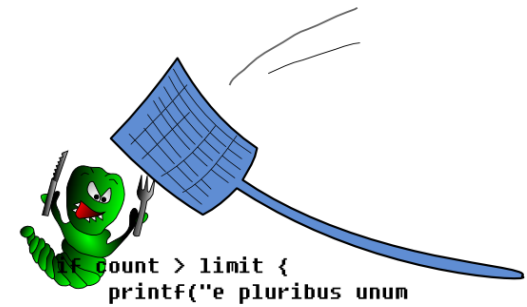
Visual Studio



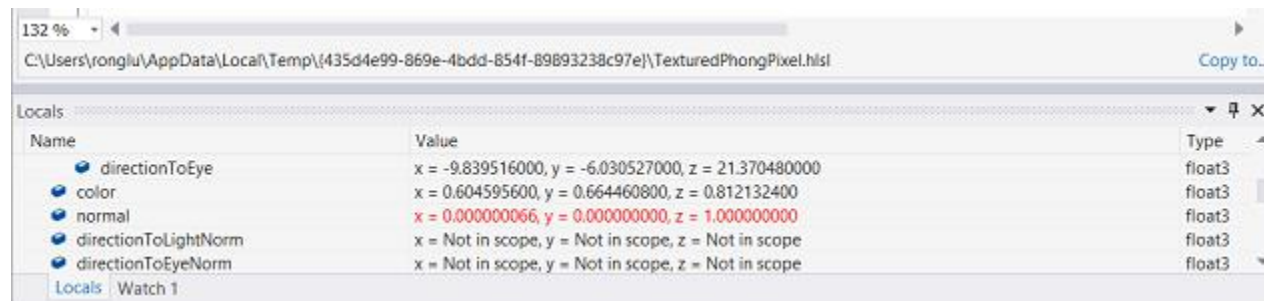
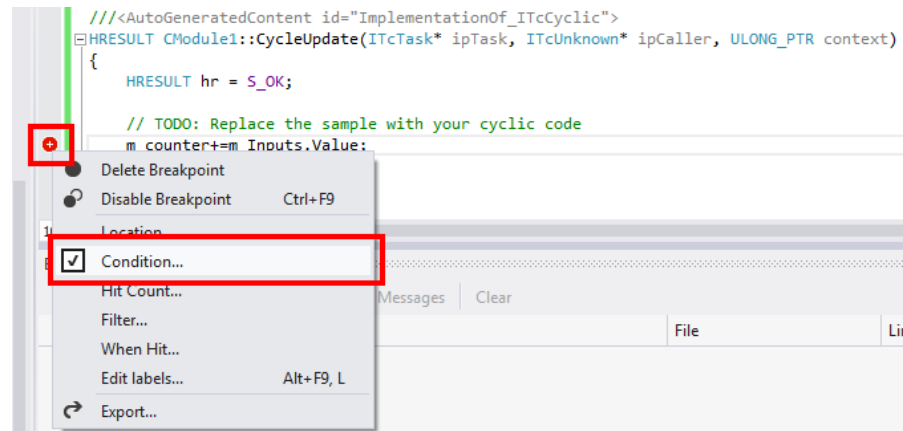
Qt Creator

Debugger

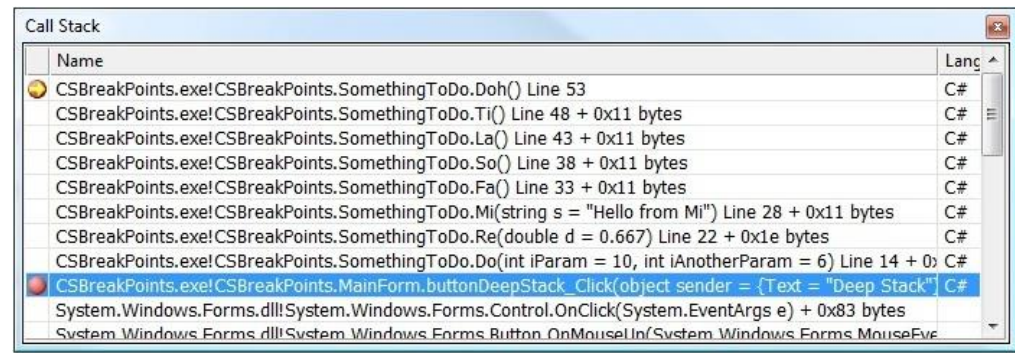
- Werkzeug zum Auffinden von Fehlern
 - Vornehmlich in Programmen
 - Gibt es aber auch für Hardware
 - Kann auch zum Reverse-Engineering eingesetzt werden
- Hauptfunktion:
 - Anhalten eines Programms zu beliebigem Zeitpunkt
 - Inspektion des Inhalts des Speichers
- Prinzipiell lässt sich jedes Programm auf Assembler-Ebene debuggen
- Debugger für Hochsprachen bieten aber viele Komfortfunktionen
 - Auflösung von Funktionen
 - Haltepunkte direkt im Quellcode-Editor
 - Ändern von Speicherinhalten



- Haltepunkte
 - Definition von Haltebedingungen
- Darstellung des Speicherinhalts
 - Globale und lokale Variablen



- Aufrufliste (Call-Stack)
 - Zeigt, welche Funktionen vor dem Haltepunkt aufgerufen wurden



The screenshot shows the Visual Studio IDE with the FirefoxDebugging (Debugging) project open. The main window displays the source code for nsWindow.cpp, with the following code visible:

```
4341     case WM_MOUSEMOVE:
4342     {
4343         // Suppress dispatch of pending events
4344         // when mouse moves are generated by widget
4345         // creation instead of user input.
4346         LPARAM lParamScreen = lParamToScreen(lParam);
4347         POINT mp;
4348         mp.x      = GET_X_LPARAM(lParamScreen);
4349         mp.y      = GET_Y_LPARAM(lParamScreen);
4350         PRBool userMovedMouse = PR_FALSE;
4351         if ((gLastMouseMovePoint.x != mp.x) || (gLastMouseMovePoint.y != mp.y))
4352             userMovedMouse = PR_TRUE;
```

The Autos window shows the following variables:

Name	Value	Type
lParamScreen	27263500	long
mp	{x=524 y=416}	tagPOINT
x	524	long
y	416	long
mp.y	416	long
this	0x011d2e70 {mLastSize={...}}	nsWindow
userMovedMouse	2147348480	int

The Call Stack window shows the following frames:

Name	Language
gkwidget.dll!nsWindow::ProcessMessage(unsigned int msg= C++	C++
gkwidget.dll!nsWindow::WindowProc(HWND__ * hWnd=0x0	C++
user32.dll!77d48734()	
[Frames below may be incorrect and/or missing, no symbols	
user32.dll!77d48816()	
user32.dll!77d489cd()	
gkwidget.dll!nsAppShell::GetNativeEvent(int & aRealEvent= C++	C++
appshell.dll!nsXULWindow::ShowModal() Line 405	C++
appshell.dll!nsContentTreeOwner::ShowAsModal() Line 458	C++
appshell.dll!nsContentTreeOwner::ShowAsModal() Line 458	C++

- Die Unreal-Engine bietet neben dem Debugging in der IDE auch Funktionalität für das Debugging von Blueprint

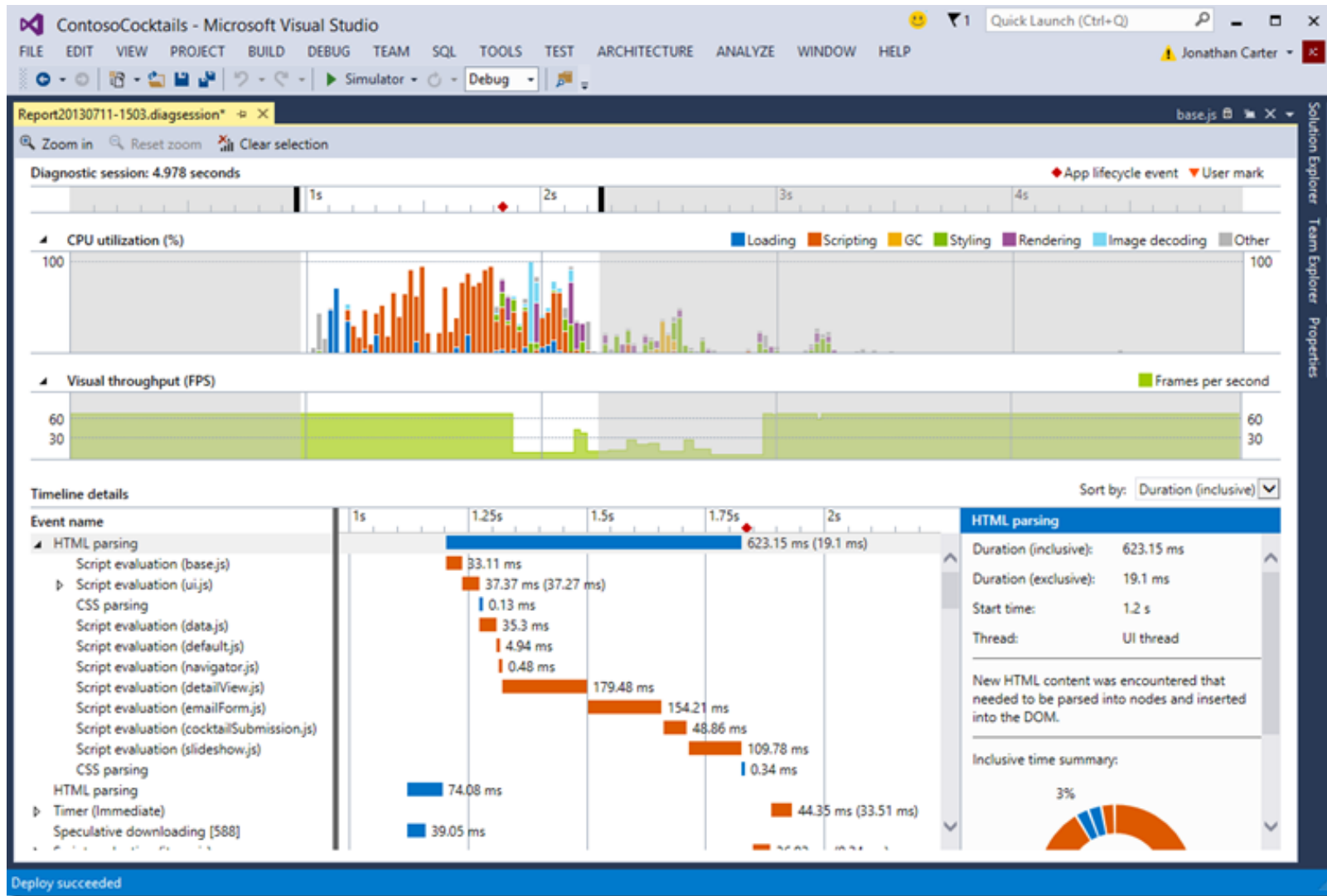
The screenshot displays the Unreal Engine IDE interface. The main window shows a Blueprint graph for the 'GenerateQuads' function. The graph includes nodes for 'Add pin', 'Watching Float Return Value', and a 'Branch' node. The 'Watching Float Return Value' nodes show values of 600.000000 and 3400.000000. The 'Branch' node has a 'Condition' set to 'True'. The 'Blueprint Debugger' window is open in the foreground, showing a list of actors and their debug information. The debugger is currently showing debug info for selected actors.

Name	Value
DebuggingExample_3	
Display Timer	0.000000
CallFunc_AddRelativeLocation	
CallFunc_SetText	
PersistentLevel_LevelScript1	No debugging info
Execution Trace	
Add Relative Location	DebuggingExample_9 @ 4313.87 s
Event ReceiveTick	DebuggingExample_9 @ 4313.87 s
Event ReceiveTick	DebuggingExample_9 @ 4313.87 s
ConstructionScript	DebuggingExample_9 @ 4313.86 s
ConstructionScript	DebuggingExample_9 @ 4313.86 s

- Werkzeug zur Analyse des Laufzeitverhaltens
 - Helfen Performance-Bottlenecks zu identifizieren
 - Protokollieren während der Laufzeit verschiedene Daten
 - Speicherverbrauch
 - Funktionsaufrufe
 - Berechnungszeiten
 - Nebenläufigkeit
 - Stellen diese Daten meist grafisch dar

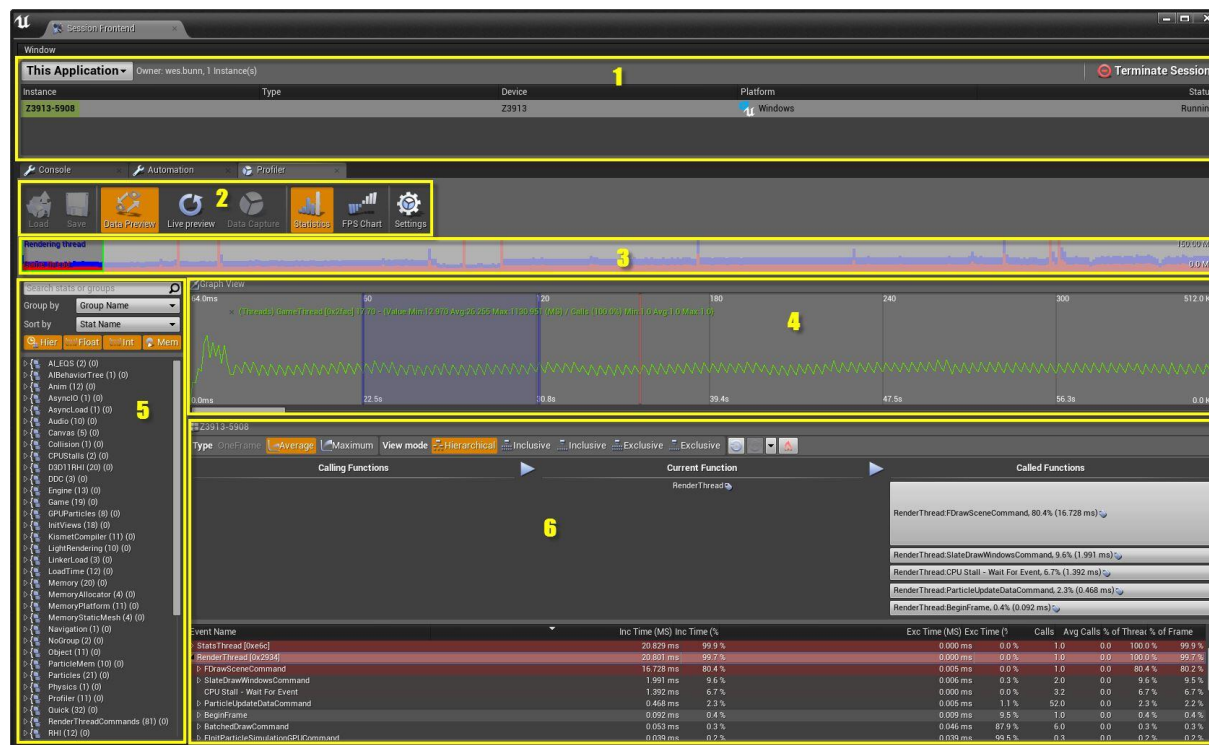
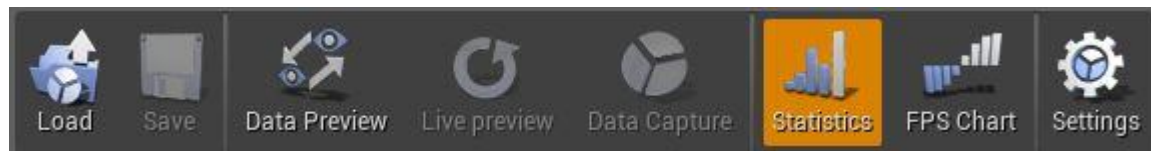


Live-Demo Visual Studio Profiler

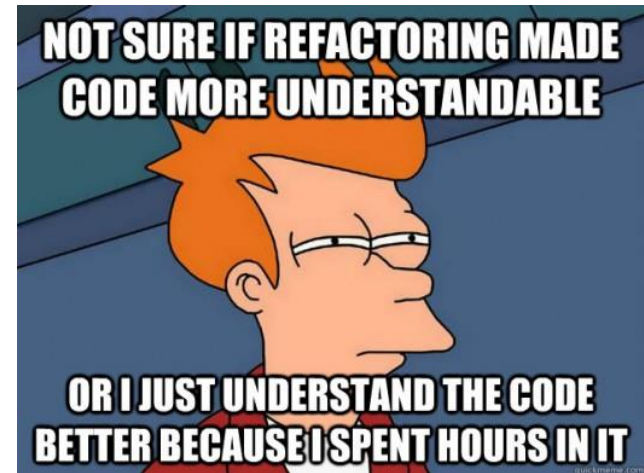


Profiling in der Unreal-Engine

- Die Unreal-Engine bietet Datensammlung von CPU- und GPU-Daten an um Performance-Bottlenecks zu finden



- Werkzeug zur **strukturellen Verbesserung** von Quelltexten unter **Beibehaltung des Verhaltens**
- Ziele
 - Lesbarkeit
 - Übersichtlichkeit
 - Verständlichkeit
 - Vermeiden von Redundanzen
- Funktionalitäten
 - Umbenennung von Symbolnamen
 - Aufteilung/Zusammenfassen von Funktionseinheiten (Klassen, Methoden)
 - Verschieben von Symbolen in andere Funktionseinheiten



- Schlecht: Lokale nichtssagende Variable x wird mehrfach verwendet

```
double x = 2 * (breite + hoehe);
cout << "Umfang: " << x << endl;
x = breite * hoehe;
cout << "Fläche: " << x << endl;
```

- Besser: Getrennte Variablen mit aussagekräftigem Namen

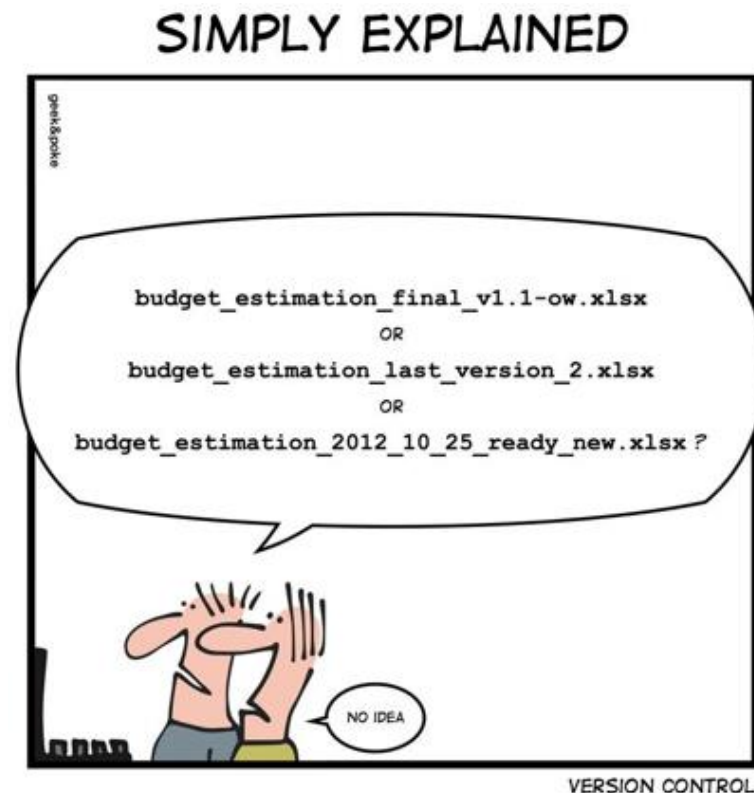
```
double umfang = 2 * (breite + hoehe);
cout << "Umfang: " << umfang << endl;
double flaeche = breite * hoehe;
cout << "Fläche: " << flaeche << endl;
```

- Andere Variante: Berechnung in Funktionen der Klasse kapseln

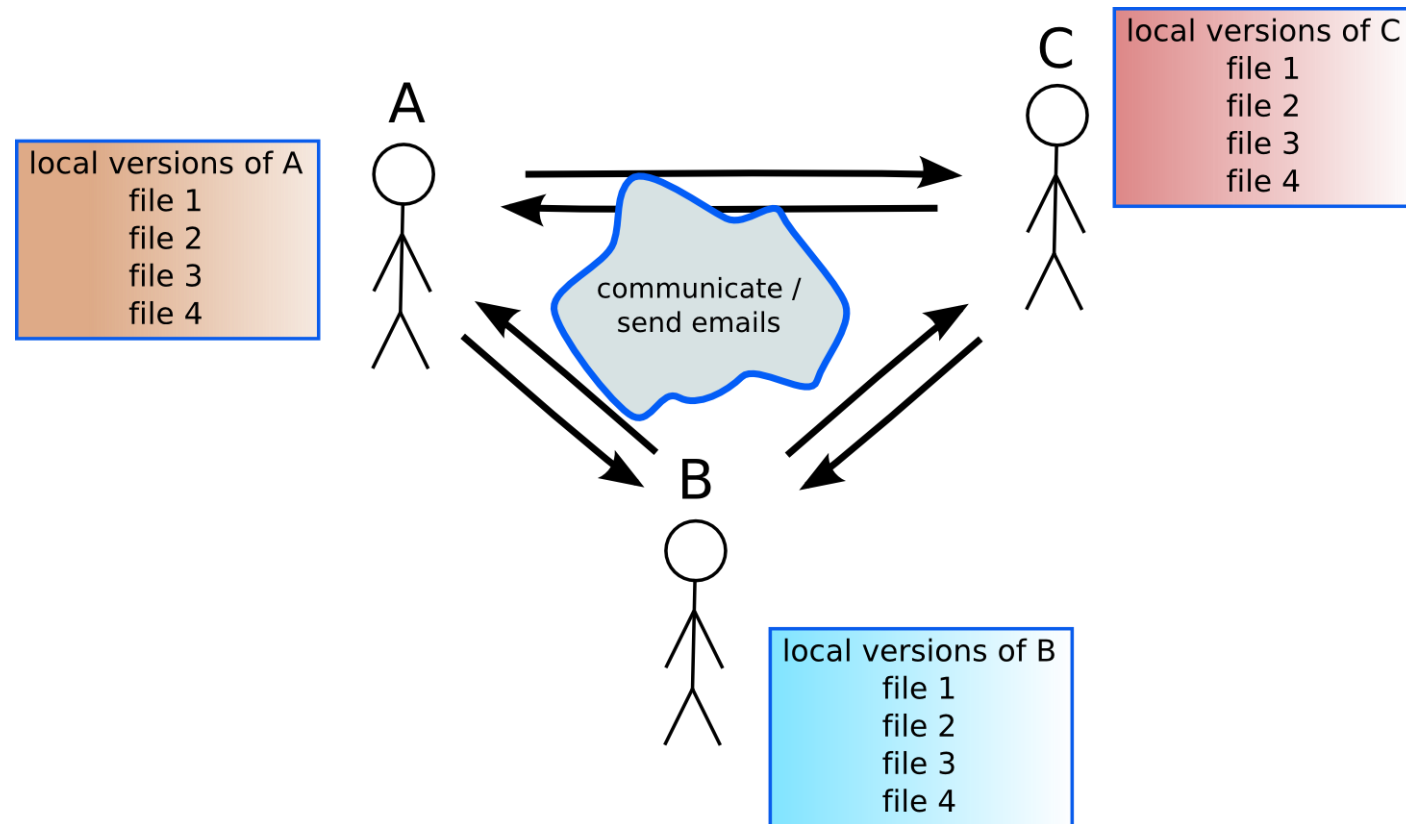
```
Rechteck *rect = new Rechteck(breite, hoehe);
cout << "Umfang: " << rect->getUmfang() << endl;
cout << „Flaeche: " << rect->getFlaeche() << endl;
```

- Wiederverwendbarkeit

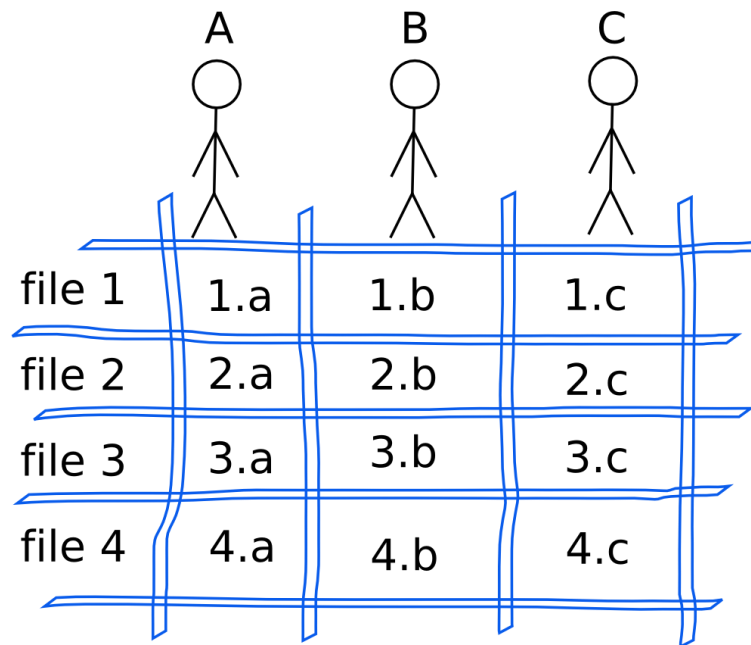
- Quellcode (aber auch anderen Content, wie Dokumente, Diagramme, ...) zu verwalten ist relativ einfach, wenn nur eine Person am Projekt beteiligt ist
- Bei Teamarbeit treten Probleme auf
 - Die gleiche Datei kann von mehreren Personen gleichzeitig bearbeitet werden
 - Diese können in anderem Raum/Land sitzen
 - Wie garantiert man trotzdem konsistenten Stand des Quellcodes?
 - Jeder muss Änderungen bekommen/sehen
 - Es darf keine Konflikte geben



- Versuch 1: Kommunikation via Email
 - Senden der bearbeiteten Dateien an alle anderen Beteiligten
 - Beispiel: 3 Personen in einem Projekt mit 4 Dateien

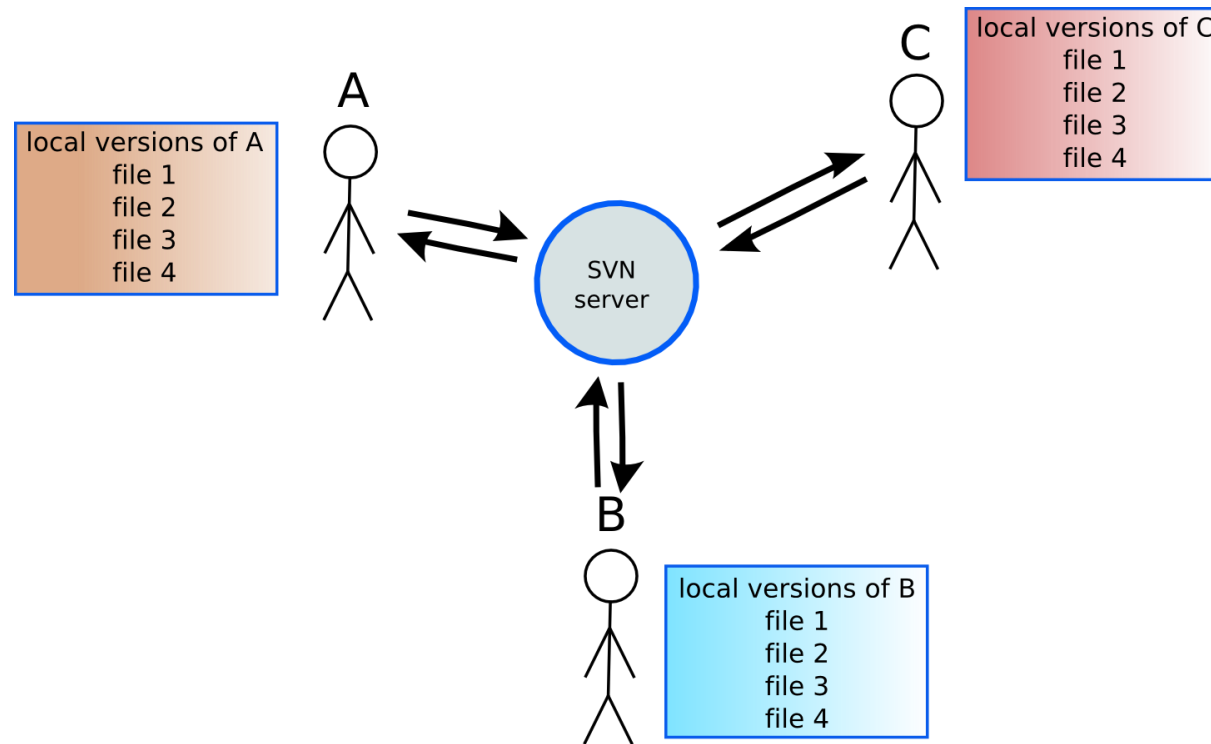


- Problem:
 - 3 unterschiedliche Versionen für je 4 Dateien
 - $4^3=64$ mögliche Kombinationen aus Dateiversion und Person



Verteilte Softwareentwicklung - Serveransatz

- Versuch 2: Speichere Dateien auf gemeinsamem lokalem Server
 - Speichere auch ältere Versionen (Historie)
 - Auflösung von Konflikten

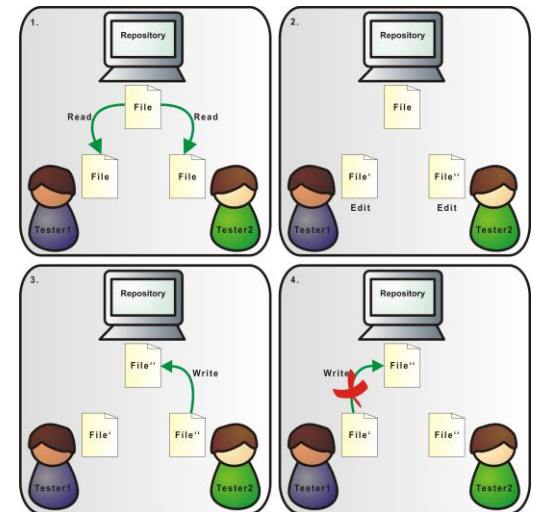


- Lock-Modify-Unlock

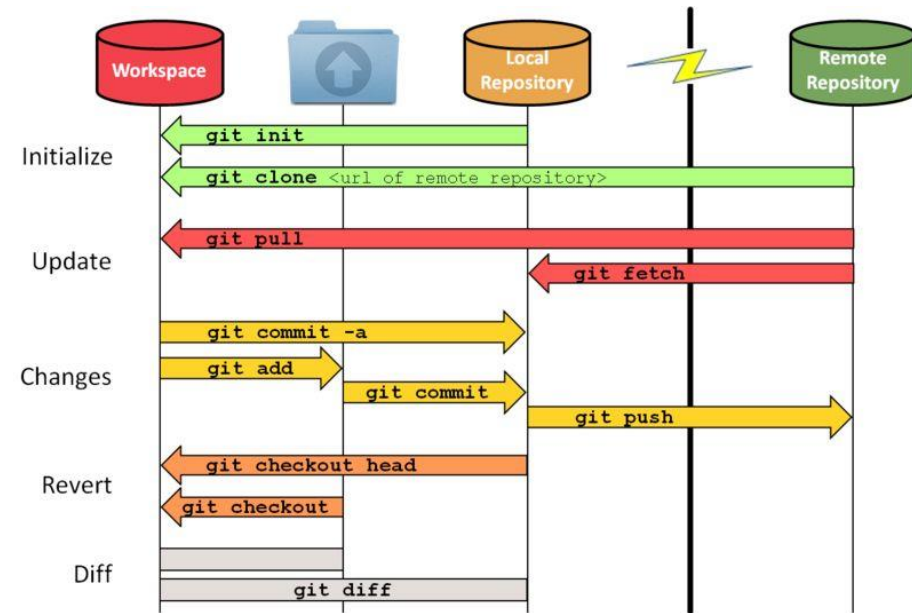
- Wer eine Datei bearbeiten will, sperrt sie für alle anderen Benutzer
- Vorteil: Keine Konflikte
- Nachteil: Eine Datei kann immer nur von einer Person gleichzeitig bearbeitet werden
- Nur für kleine Teams

- Copy-Modify-Merge

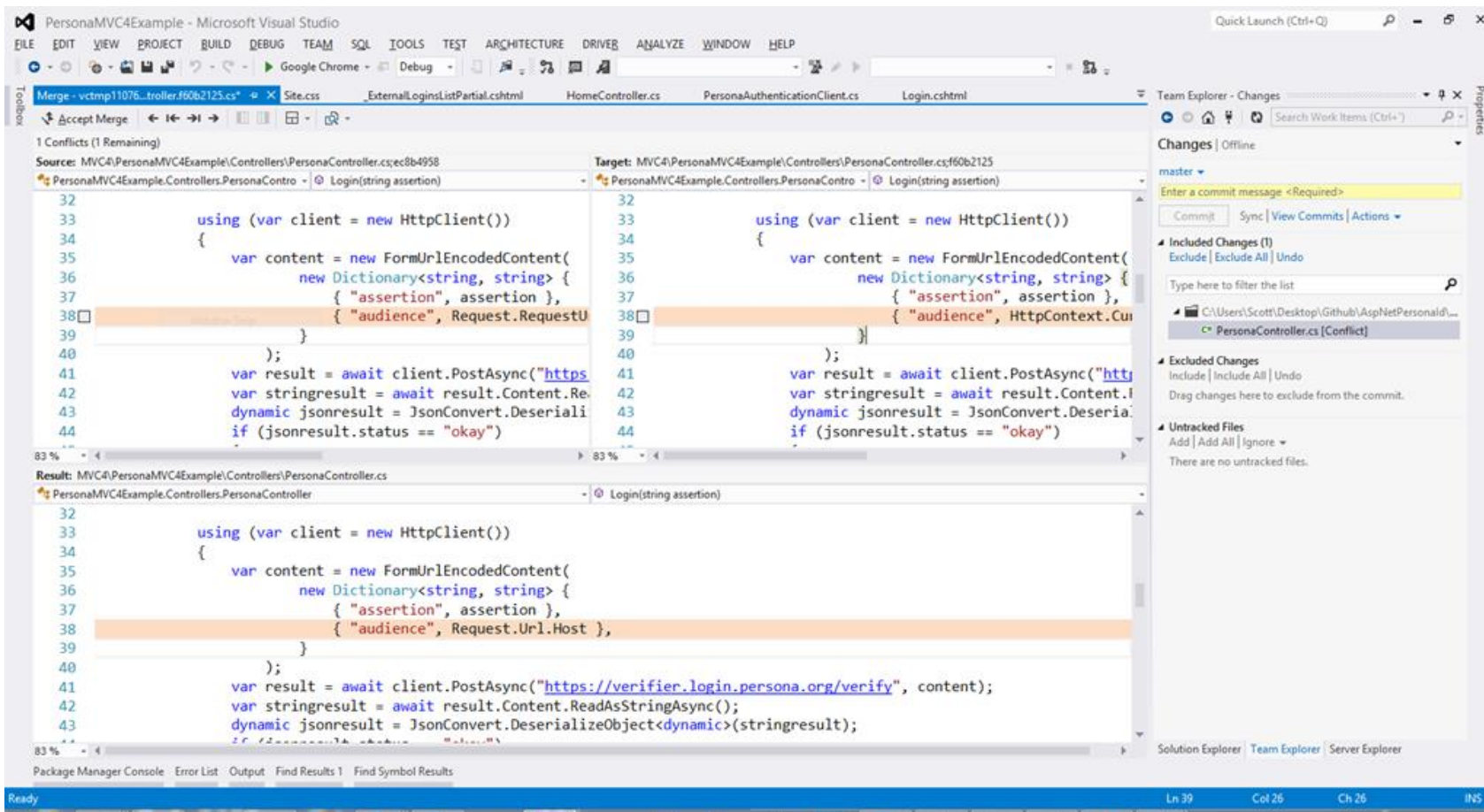
- Jeder holt sich eine lokale Version der Dateien
- Alle können simultan an allen Dateien arbeiten
- Im Fall von Konflikten: Auflösung erforderlich
- Geeignet auch für große Projekte und Teams
- Beispiele: CVS, SVN (SubVersion), Mercurial, **Git**



- Basisbefehle
 - clone
 - Erstellt lokale Kopie eines Git-Repositories
 - commit
 - Übernimmt Änderungen in lokale Kopie
 - push
 - Überträgt Änderungen an Server
 - pull
 - Holt Änderungen vom Server



- Wenn Konflikte auftreten, werden sie in einem Diff-Editor angezeigt
- In einem Merge-Schritt werden sie vom Entwickler aufgelöst



Was gehört ins Repository und was nicht?

- In der Datei `.gitignore` wird definiert, welche Dateien nicht ins Repository gepusht werden
- Es sollten alle Dateien enthalten sein, die man zum Erstellen des Projekts braucht
 - Sourcecode
 - Externe Libraries
 - Entweder als Sourcecode
 - Oder als Binary (*.lib, *.dll)
 - Content
 - (eventuell) Dokumentation
- Dateien die automatisch während des Builds erstellt werden gehören nicht ins Git-Repository
 - z.B. das erstellte Executable und Libraries (*.dll)
 - Debug-Informationen (*.pdb), Zwischendateien (*.obj), Logs (*.log),...

■ Das gehört rein:

- Config (Default-Konfigurationsdateien)
- Content (3D-Objekte, Texturen, Maps,...)
- Source (Quellcode)

■ Das nicht:

- DerivedDataCache (temporäre Dateien die während des Spiels erstellt werden)
- Intermediate (temporäre Dateien die beim Kompilieren entstehen)
- Saved (Lokale Konfigurationsdateien, Auto-Saves, Screenshots,...)

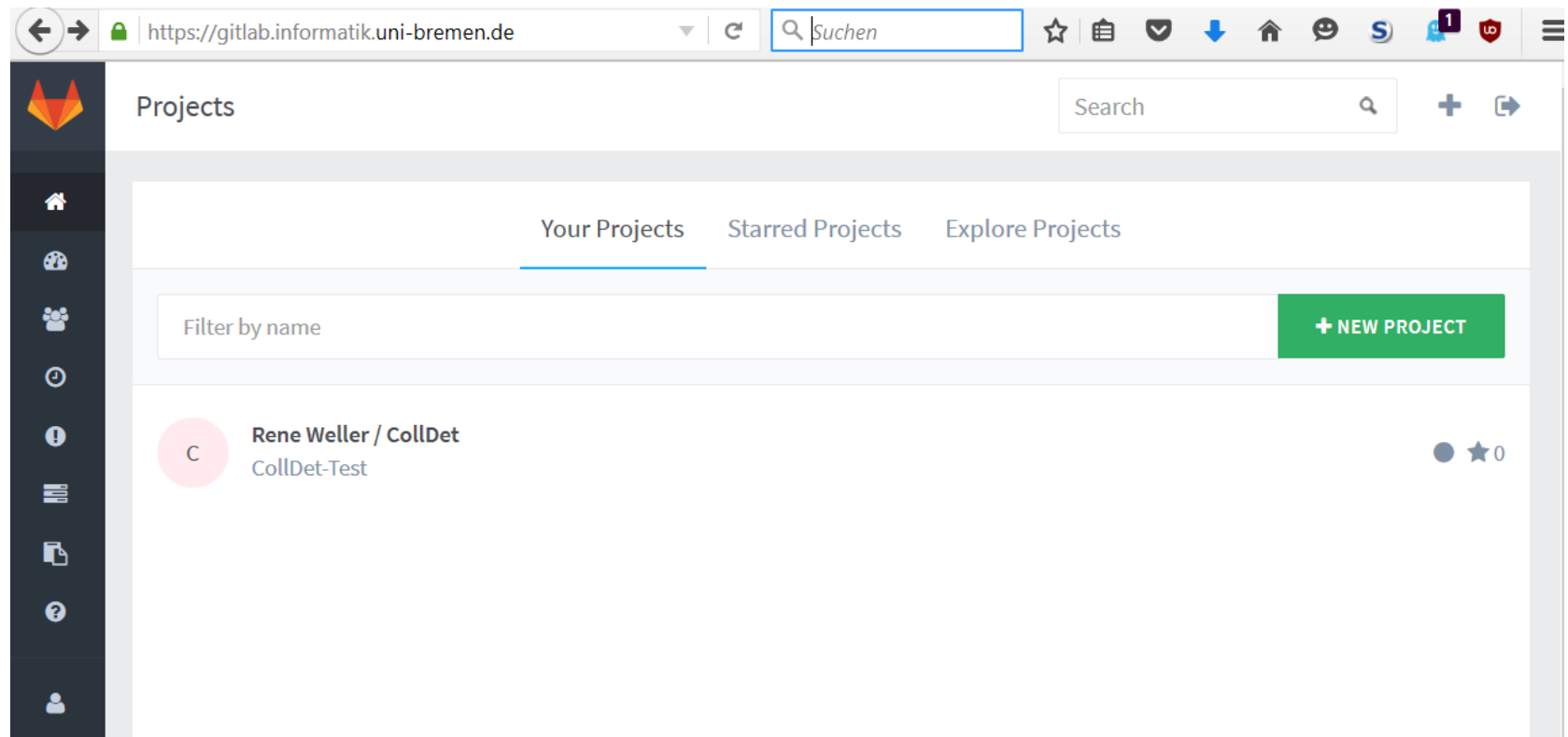
■ Kann, muss aber nicht:

- Build (Plattformabhängige Skripte zum Erstellen des Projekts und Abhängigkeiten)

Name	Date modified	Type	Size
Binaries	03/10/2014 09:52	File folder	
Build	03/10/2014 09:50	File folder	
Config	03/10/2014 09:50	File folder	
Content	03/10/2014 09:50	File folder	
DerivedDataCache	03/10/2014 10:05	File folder	
Intermediate	03/10/2014 11:10	File folder	
Saved	03/10/2014 11:10	File folder	
Source	03/10/2014 09:50	File folder	
MyProject.uproject	03/10/2014 09:50	Unreal Engine Proj...	1 KB
TP_FirstPerson_Preview.png	03/10/2014 09:50	PNG image	97 KB

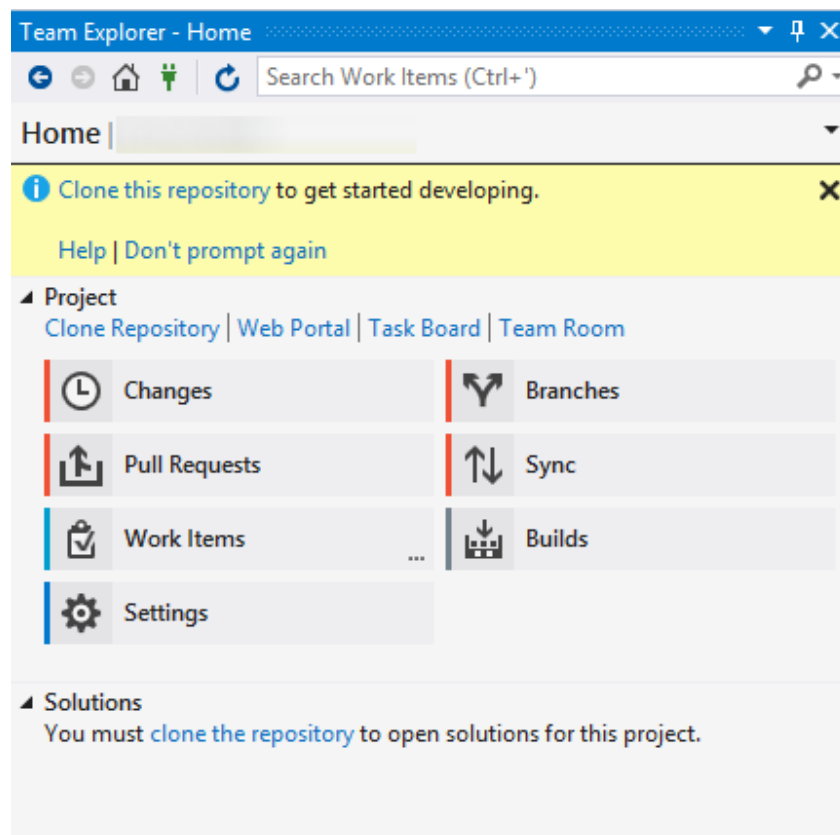
Git-Repository für die Gruppe aufsetzen

- Der FB3 der Uni Bremen stellt einen Gitlab-Server für die Benutzer der Uni bereit
 - <http://www.informatik.uni-bremen.de/t/Dienste/Gitlab>



Live-Demo Git in Visual Studio

- Git wird in VS 2015 nicht direkt unterstützt und benötigt ein kostenloses Plugin (direkt von Microsoft)



- Bieten meist erweiterte Funktionalität im Vergleich zu reinen Versionsverwaltungsservern
 - Webhosting
 - Forum, Mailinglisten
 - Build-Systeme
 - Bug-Tracking
 - Wiki
 - Statistiken
 - Blogs
 - ...
- Beispiele
 - Github, Google Code, Assembla, Bitbucket, SourceForge



- Overleaf.com
 - Gemeinsam Latex-Dokumente erstellen
 - Live-Vorschau
 - Arbeiten direkt im Browser

The screenshot shows the Overleaf LaTeX editor interface. On the left, there is a file manager showing a project named 'files' with several documents like 'Compare_S.pdf', 'Compare_pulse.pdf', 'Courant_2.pdf', 'FigureLP_ancien_mod.pdf', 'IEEEabrv.bib', 'IEEEtran.bst', 'IEEEtran.cls', 'biblio_traps_dynamics.bib', and 'main.tex'. Below the file manager are buttons for 'Download as zip' and 'Save to Dropbox'.

The main editor area is split into three panes: Source, Rich Text, and Preview. The Rich Text pane shows the title 'Modeling of Trap Induced Dispersion of Large Signal Dynamic Characteristics of GaN HEMTs' and the authors 'O. Jardel, S. Laurent, T. Reveyrand, R. Quere, P. Nakkala, A. Martin, S. Piotrowicz, M. Campovecchio, S.L. Delage'. Below the authors is their affiliation: '1111-V Lab, route de Nozay, 91461 Marcoussis Cedex, France' and '2XLM, 7 rue Jules Valles, 19100 Brive-la-gaillarde, France'. The Preview pane shows the rendered document, including the title, authors, and the start of the abstract. The abstract text is: 'We propose here a non-linear GaN HEMT model for CAD including a trapping effects description consistent with both small-signal and large-signal operating modes. It takes into account the dynamics of the traps and then allows to accurately model the modulated large signal characteristics that are encountered in telecommunication and radar signals. This model is elaborated through low-frequency S-parameter measurements complementary to more classical pulsed-IV characterizations. A 8x75µm AlInN/GaN HEMT model was designed and particularly validated in large-signal pulsed RF operation. It is also shown that thermal and trapping effects have opposite effects on the output conductance, thus opening the way for separate characterizations of the two effects.'

The Preview pane also shows the start of the introduction section, which discusses Gallium Nitride (GaN) High Electron Mobility Transistors (HEMT) on SiC and their applications in telecommunication and radar. It mentions that these devices are subject to parasitic effects such as thermal effects and especially trapping effects. The text continues to describe the impact of these trapping effects on the dynamic large signal characteristics and the need for a consistent nonlinear model.

Extern oder in die IDE integriert?

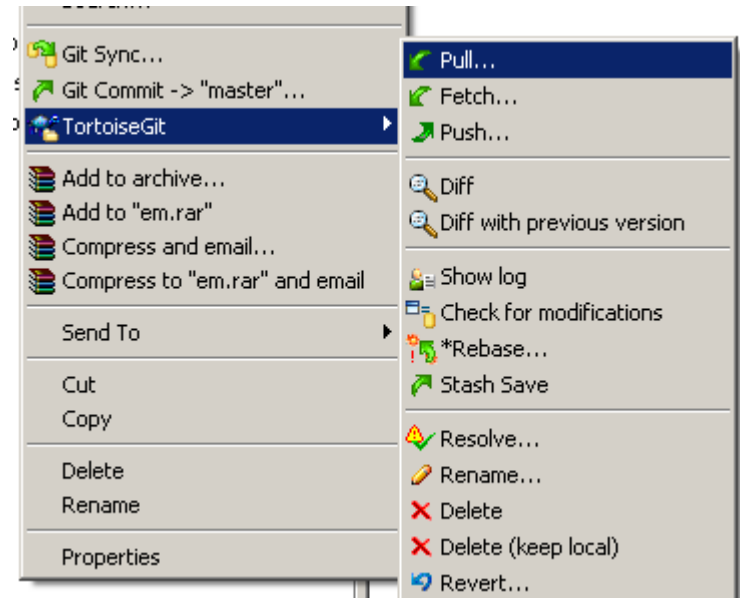
- Für alle hier in die IDE vorgestellten Werkzeuge gibt es auch externe Einzelprogramme
 - Vorteile:
 - oft deutlich erhöhte Funktionalität
 - Unterstützung für Spezialhardware (z.B. die GPU)
 - Nachteile
 - Einarbeitungszeit
 - Kosten
 - Unterbrechung des Workflow



VS



- Beispiele:
 - Editor: Hatten wir schon
 - Emacs, Vim
 - Profiler
 - Intel VTune
 - NVIDIA Nsight (für die GPU)
 - Versionskontrolle
 - TortoiseGit
- Welche man nimmt ist Geschmackssache (oder vom Auftraggeber vorgeschrieben)
- Für unser einfaches Projektchen sollten die integrierten Werkzeuge genügen

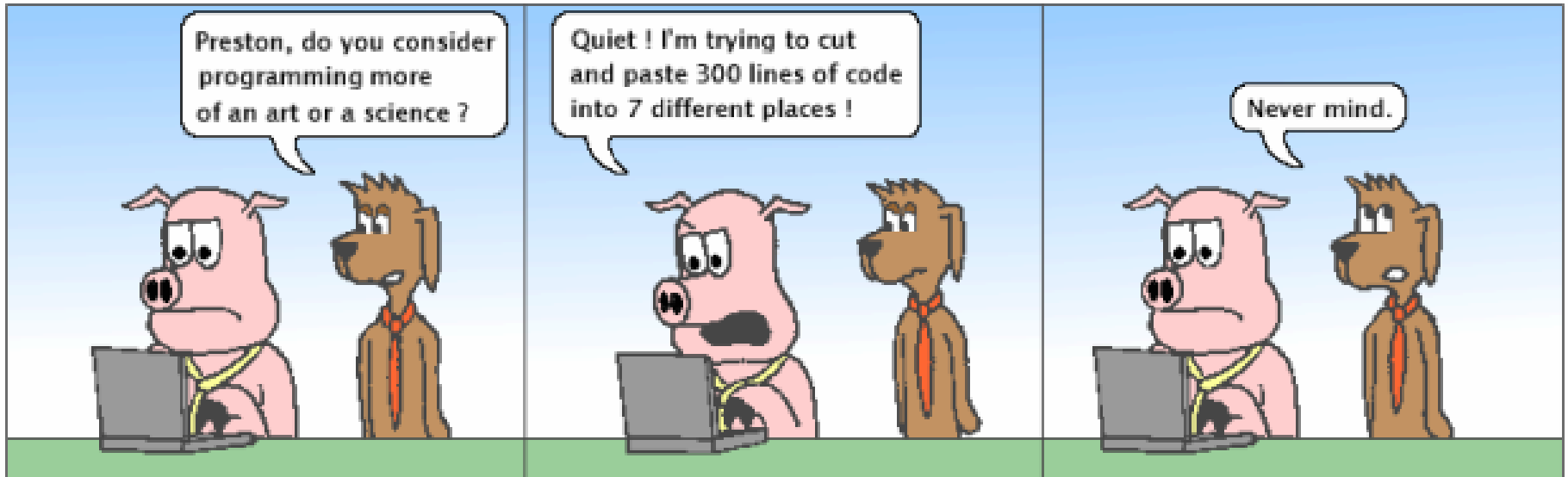


Für unser Projektchen notwendige Werkzeuge

- IDE (Visual Studio, Xcode)
 - Editor
 - Compiler/Linker
 - Debugger
 - Versionskontrolle (mit Git-Plugin)
 - Profiler
- Dokumentationswerkzeug
 - Doxygen

Hackles

By Drake Emko & Jen Brodzik



<http://hackles.org>

Copyright © 2001 Drake Emko & Jen Brodzik