Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

History of IDEs

vorgelegt von

Justin Welsch

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang:	Software-System-Entwicklung
Matrikelnummer:	6547005
Betreuer:	Christian Hovy

Hamburg, 2017-03-03

Abstract

Integrated Development Environments, oder kurz IDEs, sind fester Bestandteil der Software Entwicklung und werden es vermutlich auch noch lange bleiben.
In dieser Ausarbeitung wird allerdings kein Blick in die Zukunft, sondern in die Vergangenheit, auf die Geschichte der IDEs geworfen. Beginnend mit den ersten Prototypen in den 60ern, über die ersten richtigen IDEs Anfang der 80er bis zu den relativ modernen IDEs die auch heute noch benutzt werden. Hierbei handelt es sich aber nur um eine kleine Auswahl der mittlerweile riesigen Auswahl von Entwicklungsumbgebungen. Dabei sind leider nicht alle alten IDEs und Prototypen gut dokumentiert, weswegen zu beachten ist das für die Prototypen lediglich Wikipedia als Quelle genutzt werden konnte und die dort enthaltenen Informationen nicht weiter verifiziert werden können.

Contents

1 Einleitung

1.1 Was zeichnet eine IDE aus

Eine Integrated Development Environment oder auf Deutsch integrierte Entwicklungsumgebung ist im Grunde eine Sammlung von Tools die das Entwickeln einfacher macht. Es gibt sie für viele verschiedene Sprachen und in vielen verschiedenen Ausführungen, doch es gibt ein paar essentielle Features von denen zumindest die meisten enthalten sein müssen um als IDE durchzugehen.
Zunächst besteht jede IDE einmal aus einem Source Code Editor. In der schlichtesten Form ist es einfach ein gewöhnlicher Texteditor wie man ihn aus Programmen wie NotePad kennt, allerdings gibt es auch hier einiges an Features. So enthält heutzutage im Grunde jede IDE einen Texteditor der auch Syntax-Highlighting und Autoformatting anbietet.
Ein weiteres Wichtiges Feature, was in vielen IDEs enthalten ist, ist der Debugger. Er ermöglicht es, den Code zur Laufzeit zu analysieren und durch Breakpoints auch Analysen zum Laufzeitverhalten anzustellen. Dieses Feature war nicht von Anfang an Teil der IDEs und ist erst seit Ende der 80er Bestandteil der IDEs.
Ein drittes, essentielles Feature, ist die Code Übersetzung um ihn ausführbar zu machen. Dafür ist eine IDE entweder mit einem Compiler oder einem Interpreter ausgestattet, je nachdem für welche Sprache die IDE genutzt wird.
Weitere nice-to-have, aber nicht notwendige Features sind die automatische Code completion, was dem Autor viel aufwändige Schreibarbeit abnimmt durch intelligente Autovervollständigung und Einbindung von Version Control Systems wie git oder svn direkt in die IDE .
Und je nach Sprache auch noch Tools zum Bauen von GUIs und/oder Klassen und Objekt Browser, sowie weitere Objekt orientierte Hilfstools für Objektorientiere Sprachen.

2 Vor der ersten IDE

2.1 Vim und Emacs

Die Frage, ob Vim und Emacs als IDEs gewertet werden können, ist eine der klassischen Streitpunkte der Software Entwicklung mit zahlreichen Diskussionen, von daher werde ich hier nur eine kurze Einschätzung zu diesem Thema geben.

2.1.1 Vim

Betrachtet man zunächst nur Vim in seiner alleinstehenden Grundform sollte die Antwort klar ein Nein sein, denn Vim ist erstmal im Grunde nur ein Texteditor, der über keine zusätzlichen, für die Softwareentwicklung relevanten Features verfügt. Trotzdem ist Vim eine beliebte Wahl für Software Entwickler die mittels einiger Plug-ins die fehlenden Features hinzufügen. So gibt es Plug-ins für File Browsing, Code Browsing, Text Completion, Formatting, Syntax-Highlighting, integrierte Version Control Systems, eine verbesserte UI und es gibt sogar Plug-ins die es ermöglichen mit Vim zu debuggen, allerdings nicht ganz so komfortable wie mit einer klassischen, modernen IDE.
Es gibt auch einige Vim Distributions wie z. B. spf13 die schon von vornherein mit Plug-ins für die Entwicklung von Software ausgestattet sind.
Mit all diesen Plug-ins installiert erfüllt Vim genug Anforderungen um als IDE durchzugehen, allerdings wird damit die Frage aufgeworfen, wie viel Sinn dabei ist. Um diese Frage zu beantworten, muss man beachten aus welchem Grund man Vim als IDE benutzen möchte. Falls es zum Beispiel darum geht in einer möglichst simplen und überall verfügbaren Umgebung zu arbeiten, so ist das durch die Notwendigkeit der Plug-ins nicht mehr komplett gegeben und falls es nur um den Texteditor ansich geht, so gibt es mittlerweile für fast jede IDE die Möglichkeit Vim als Texteditor einzubinden.
Zusammenfassend kann man aber sagen, das Vim zur voll funktionalen IDE aufgerüstet werden kann und es am Ende eine Frage der Präferenz ist. [Vim, 1]

2.1.2 Emacs

Für Emacs lassen sich Grundsätzlich die gleichen Argumente wie für Vim finden. Es ist aller erstens ein Texteditor, der wie Vim eine Vielzahl von Plug-ins zur Verfügung hat mit denen er zu einer adäquaten IDE ausgebaut werden kann.
Der Unterschied liegt allerdings darin, dass Emacs von Haus aus bereits genug Funktionalität bietet um als IDE anerkannt zu werden. So kann auch ohne Plug-ins direkt aus Emacs kompiliert und debuggt werden. [Ema, 02]

2.2 Dartmouth BASIC

Dartmouth BASIC ist keine IDE im eigentlichen Sinne, allerdings kann man es als ersten Prototypen bezeichnen
Dartmouth BASIC beschreibt eigentlich die erste Iteration der BASIC Programmiersprache, wurde von John Kemeny und Thomas Kurtz an der Dartmouth Universität entwickelt und bereits 1964 veröffentlicht. Dartmouth BASIC lief auf dem Dartmouth Time Sharing System(DTSS). Ein Time Sharing System bezeichnet ein System, bei dem mehrere Nutzer auf den gleichen Ressourcen arbeiten um diese effizienter zu nutzen. Die Idee dahinter ist, dass ein einzelner Nutzer nicht kontinuierlich am System arbeitet, sondern viele Pausen hat. Diese Pausen-Zeit kann dann von anderen Nutzern genutzt werden. Mit diesem DTSS wurde über ein Command Line Interface interagiert welches den IDE Teil dieses Systems repräsentiert. Über diese CLI konnten neben dem Schreiben des Programms auch die Standard-Befehle des DTSS ausgeführt werden. Diese setzten sich zusammen aus Filesystem Operations wie dem Speichern und Überschreiben einer Datei, das Ausführen und Stoppen von Programmen sowie Standard System Management Operationen wie das An- und Abmelden.
Man ist also noch sehr eingeschränkt in dem was man tun kann, aber in den Ansätzen ist klar die Idee der IDE zu erkennen. [Dar, 03]

2.3 Maestro I

Auch Maestro I ist nicht das was man sich unter einer IDE vorstellt, fällt aber auch wie Dartmouth BASIC unter die Kategorie Prototyp. Es wurde erst ein paar Jahre später als Dartmouth BASIC entworfen, im Jahre 1975 und fand dafür aber anders als Dartmouth BASIC auch außerhalb von Universitäten in der kommerziellen Welt anklang und ermöglichte es die Lochkarten abzulösen.
Entwickelt wurde es von Harald Wieler in dem bayrischen Unternehmen Softlab und fand auch Unterstützung vom Bundesministerium für Bildung und Forschung und wurde weltweit ca 22000 mal installiert, ca. drei Viertel davon außerhalb Deutschlands.
[Mae, 04]

3 IDEs

3.1 Turbo Pascal

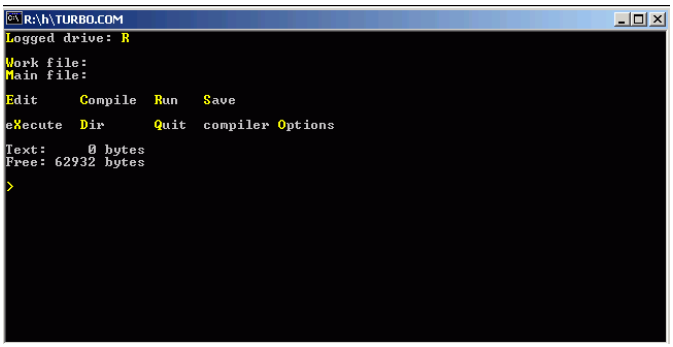

Figure 3.1: [Tur, TurboPascal]

Turbo Pascal erschien am 20. November 1983 mit der Version 1.0. Es war die erste weitverbreitete Entwicklungsumbgebung. Zu der Zeit als Turbo Pascal erschien, kosteten PCs noch mehrere tausend DM, waren nur mit 64 bis 128 K RAM ausgestattet und hatten auch keine Festplatten, sondern lediglich ein oder zwei Diskettenlaufwerke. Für den Heimgebrauch waren Computer zwar etwas günstiger, aber auch entsprechend weniger gut ausgestattet.
Die damals gängigste Programmiersprache war BASIC, so waren die meisten PCs auch mit einem BASIC Interpreter ausgestattet. Da BASIC eine interpretierte Sprache war sorgte dies noch einmal zusätzlich zur limitierten Hardware für Zeitkosten.
TurboPascal änderte dies, denn TurboPascal bestand aus Editor, Linker und Compiler in einem, ermöglichte schnelles compilieren und war trotzdem nur 31 KByte groß. Ein weiterer Vorteil war der geringe Preis von nur 50 USD was für damalige Verhältnisse sehr günstig war.
Turbo Pascals Benutzerinterface war schlicht, es ähnelte sehr einem einfachen Texteditor wie Vim. Es wurde komplett über Tastenkombination gesteuert, allerdings konnte man diese schon damals beliebig belegen. Ein weiteres Feature war das Filemanagment. So hatte man im Editor jederzeit zwei Datein geladen, einmal das Work File, welches die Datei war welche gerade bearbeitet wurde und einmal das Main File. Das Main File war die Datei die den Zentralen Code enthielt und von wo aus die anderen Dateien verwendet wurden. Wurde das ganze nun kompiliert und es kam ein Fehler auf der nicht in der Main File, sondern in einer der Work Files lag, so wurde automatisch die entsprechende Work File geladen, sowie die Zeile angegeben in der der Fehler lag.

Die größte Restriktion in Turbo Pascal 1.0 lag darin, das es nicht möglich war Datein zu erstellen die größer als 64 K waren.
Mit Version 2 und 3 von Turbo Pascal kam es zu keinen großen Änderungen. Die Interessanteste war noch, dass es nun möglich war Programme zu erstellen die eine Größe von bis zu 152 K haben konnten, die Dateigröße war allerdings noch immer auf 64 K beschränkt.
1987 kam es mit Version 4.0 hingegen zu umfangreichen Änderungen. Das User Interface wurde komplett überarbeitet und ging weg vom schlichten Texteditor und hin zum klassischen IDE Design. Das bedeutete graphisch deutlich aufgewertete Oberfläche mit Menüs und Mausunterstützung. Außerdem enthielt Turbo Pascal nun zwei weiter wichtige Features die eine IDE ausmachen, Syntax-Highlighting und ein Debugger. Des Weiteren wurde auch die 64 K (oder 152 K) grenze aufgehoben und das Erzeugen von .exe Files ermöglicht.
Mit Turbo Pascal 5.5 kam dann noch die Unterstützung für die nun in Pascal enthaltenen objektorientierten Elemente, doch im Grunde gab es hier und in der letzten Version 7.0 keine relevanten Änderungen für die IDE mehr. 1992 erschien die letzte Version von Turbo Pascal für DOS. Es erschienen zwar noch zwei Versionen für Windows, doch fanden diese wenig Anklang und mit Turbo Pascal ging es zu Ende.
Wenn man mit heutigen Standards auf Turbo Pascal zurück blickt, so sieht man zwar ein alte, aber auch immer noch sehr nutzbare IDE. Auch die Version 1.0 lässt sich auch heute noch mit wenig Eingewöhnung bedienen, denn es handelt sich im Grunde um Texteditoren wie Vim mit ein paar zusätzlichen Features.
[Pasa, 05]
[Pasb, 06]

3.2 Microsoft Visual Basic

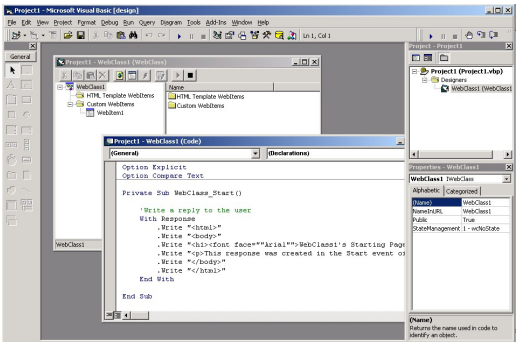

Figure 3.2: [Visd, VisualBasic]

Visual Basic, oder Visual Basic classic, ist eine vom Microsoft entwickelte und 1991 erschienene Programmiersprache und IDE und wurde darauf ausgelegt, Anwendungen für Windows zu erstellen. Visual Basic hatte anders als die Vorherigen IDEs keinen integrierten Compiler, sondern einen Interpreter der den Code in P-Code übersetzte, der nicht direkt von Maschinen ausführbar war und deshalb noch eine separate Laufzeitbibliothek von Nöten war. Erst mit Version 5 im Jahre 1997 wurde der Code compiliert.
Neben der Viusal Basic Variante für Windows gab es auch noch eine Version für MS-DOS, welche statt Windows Programmen, Programme für DOS erzeugte. Diese war allerdings nie erfolgreich und wurde nicht mit der Windows Version zusammen weiter geführt.
Was Visual Basic so besonders machte war der Ansatz des Visual Programmings. Das bedeutete das man in der IDE nicht einfach nur Code schreiben konnte, sondern es nun auch viele Drag and Drop Element zum designen von Graphical User Interfaces gab. Man konnte sich das Interface des Programms also einfach zusammenklicken und dann für die einzelnen Komponenten Code hinzufügen, was gleichzeitig einen leichten Einstieg in die Software Entwicklung bot aber auch simultan ein mächtiges Werkzeug zum Erstellen von Programmen war. 1995 kam dann mit Version 4.0 auch 32-Bit Support hinzu. Auch war dies das erste Mal das Microsoft ihr Produkt in verschieden Ausgaben auf den Markt bringt, der Standard, Professional und Enterprise Editionen. Mit Version 6.0 kam dann im Jahre 1998 die letzte Version auf den Markt, allerdings lief der Support noch weiter bis 2008.
[Vise, 07]
[Visc, 08]

3.3 IBM Visual Age

Visual Age ist eine Gruppe von Entwicklungsumgebungen entwickelt von IBM. Visual Age erschien für einige Sprachen, darunter C++, COBOL und Java, wurde aber primär für SmallTalk vertrieben und wurde auch in den meisten Implementationen in SmallTalk geschrieben. Die erste Visual Age Version für SmallTalk erschien bereits in 1993 und wurde noch bis 2011 weiter entwickelt.
Allerdings sind die meisten Visual Age Implementationen nicht sehr besonders und heutzutage auch kaum noch dokumentiert. Herausstechen allerdings tut Visual Age for Java, was das Entwickeln im objektorientierten Paradigma um einiges erleichterte. Denn, anders als bei den vorherigen Visual Age Iterationen, besaß Visual Age for Java ein Repository basiertes Storage System anstelle eines File basiertem Storage Systems. Vorher, im File basiertem System, war der komplette Code unorganisiert in einem oder mehreren Datein gespeichert. Mit dem Repository System gab es nicht nur eine geordnete Speicherstruktur, sondern es wurden auch zusätzlich Informationen über Packages, Klassen, Methoden und Ähnliches gespeichert. Das ermöglichte einen geordneten Aufbau von Beziehungen zwischen den Objekten und Konstrukten wodurch Referenzen klar darstellbar waren. Das ermöglichte nicht nur schnelleres Programmieren und Fehlerbeheben, sondern ermöglichte es auch, nur den Teil neu zu kompilieren, der tatsächlich verändert wurde, was einiges an Zeit sparte. Dies Vereinfachte die Objektorientierte Entwicklung enorm und viele der Features sind auch noch in Eclipse, dem Nachfolger von Visual Age for Java enthalten.
[Visb, 09]
[Visa, 10]

3.4 Eclipse

Eclipse erschien in der Version 1.0 im November 2001. Es ist der Nachfolger zu IBMs Visual Age for Java und ist mittlerweile mit Eclipse Neon in der Version 4.6, was dem 14. großen Release von Eclipse entspricht, erschienen. Seit 2004 ist jedes Jahr im Juli eine neue Version heraus gekommen. Eclipse ist eine der am weitesten verbreiteten IDEs und ist auch oft eine der ersten IDEs mit denen viele Software Entwickler in Berührung kommen, da sie häufig in Universitäten eingesetzt wird. Eclipse ist Open Source und vor allem für den Einsatz als Java IDE bekannt und wurde auch in Java geschrieben.
1998 begann IBM unter der Führung von Object Technology International mit der Entwicklung einer Software die später zu Eclipse werden würde. Die Software wurde mit dem Gedanken entwickelt, weg von den vielen inkompatiblen IDEs hin zu einer großen Plattform, die es vielen Leuten ermöglicht daran mitzuarbeiten, zu gehen. Da dies aber kein Anklang bei Investoren fand, schloss sich IBM mit sieben weiteren Firmen, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft und Webgain zu einem Konsortium zusammen, um mit eclipse.org den Schritt in Richtung open source zu wagen. Unter diesem Zusammenschluss wurden die ersten beiden Versionen von Eclipse veröffentlicht, die bereits relativ guten Anklang in der Entwicklergemeinde fanden, doch es hatte immer noch den Anschein, dass es sich bei Eclipse um ein IBM Produkt handelte. So entstand dann im Jahr 2004 die Eclipse Foundation, eine not-for-profit Organisation unter der Eclipse bis heute läuft.
[Eclb, 11]
[Ecle, 12]
[Eclc, 13]
Heutzutage ist Eclipse auch für seine Vielen Plug-ins und die Vielzahl an Anpassungsmöglichkeiten bekannt. So ist Eclipse keine reine Java IDE, sondern kann für eine große Anzahl von Sprachen verwendet werden, unter anderem auch C und C++ mit dem CDT Plug-in, welches das Ziel hat die Integration die Eclipse für Java mit dem JDK hat, mit entsprechenden Tools für C und C++ umzusetzen.
[Ecla, 14]
Der Erfolg der Plug-ins kommt auch daher, dass Eclipse einen eingebauten Marktplatz besitzt über den Plug-ins direkt installiert und verwaltet werden können. Mittlerweile bietet der Marktplatz über 24 Millionen Plug-ins.
[Ecld, 15]
Um diese Plug-ins zu verwalten und zu integrieren benutzt Eclipse Equinox, welches auf dem OSGi-Framework basiert. Das OSGi-Framework ist eine Menge von Spezifikationen, das darauf basiert die Implementation von Komponenten zu verbergen und stattdessen über dafür konzipierte Services zu kommunizieren, was es erleichtert unabhängig voneinander konzipierte Plug-ins zusammen arbeiten zu lassen. [Eclf, 16]

3.5 Cloud9

Cloud9 ist eine relativ junge IDE, erschienen in der ersten Version in 2012, seit 2013 mit der Version 3.0 open Source und seit Juli 2016 Teil von Amazon. Wie man dem Namen entnehmen kann, handelt es sich bei Cloud9 um eine IDE, mit der man in der Cloud arbeitet und nicht auf seinem eigenen Rechner. Cloud9 ist nicht die einzige IDE in der Cloud, sondern eine von vielen, da das Thema Cloud seit einigen Jahren durch die Informatik zieht.
Eine Cloud basierte IDE ist so aufgebaut, dass der gesamte Entwicklungsprozess auf einem Server stattfindet, das bedeutet auch, die IDE ansich läuft auf diesem Server und wird über den Browser aufgerufen. Bevor man sich nun mit Cloud9 ansich befasst, sollte man zunächst einmal die Vor- und Nachteile von Cloud basierten IDEs betrachten.
Vorteilhaft ist vor allem die Hardwareunabhängigkeit, welche es ermöglicht auch auf Geräten zu arbeiten, die nicht sehr leistungsstark sind, wie zum Beispiel Chromebooks. Man ist auch nicht nur von den Hardware Anforderungen losgelöst, sondern auch von den Ortsanforderungen. Mit der IDE in der Cloud ist es möglich jederzeit und an jedem Ort an bestehenden Projekten zu arbeiten. Das ist zwar zum Teil auch mit Version Control Systemen möglich, allerdings ist es hier meist noch notwendig das eine Anzahl von Softwares auf dem System installiert sind, während für Cloud IDEs nur der Browser benötigt wird.
Damit einher geht aber auch der erste Nachteil, man ist vom Internet abhängig. Um vernünftig mit solch einer IDE zu arbeiten, ist es nicht nur notwendig überhaupt einen Internetzugang zu haben, sondern man benötigt einen möglichst stabilen und schnellen, da die User Experience sonst unter schlechter Responiveness leidet was den Vorteil der Ortsunabhängigkeit wieder ein wenig einschränkt.
Ein weiteres Problem sind die Server auf, denen das ganze läuft, liegen diese bei dem Anbieter, hat das die Konsequenz, dass man nicht wirklich im Besitz der Daten ist, was gerade bei vertraulichen Daten ein Sicherheitsrisiko darstellt.
Auch ein Sicherheitsrisiko ist, dass eben alle Daten über das Netz übertragen werden, was dazu führt das neben einem stabilen Netz auch ein sicheres benötigt wird.
Da Cloud9 eine recht moderne IDE ist, besitzt sie alle Features die man heutzutage von einer IDE erwarten kann und hebt sich auch außerhalb der Cloud Features nicht besonders von anderen IDEs ab. Das, was Cloud9 ausmacht, konzentriert sich auf die Cloud und probiert den Nachteilen, die damit mit sich kommen, entgegenzuwirken. In der standardmäßig bereitgestellten Version bekommt man als Nutzer eine VM auf einem ihrer Server mit installiertem Ubuntu System zur Verfügung gestellt und hat dort weitgehende Rechte. Man hat zwar keine Root-Berechtigungen, kann aber über einen bereitgestellten Paket-Manager zusätzliche Programme installieren. Diese VMs sind allerdings auf 512 MB speicher begrenzt. Alternativ kann man Cloud9 auch auf einen eigenen Server migrieren und sich dorthin verbinden, allerdings nicht in der kostenlosen Variante. Da Cloud9 allerdings Open Source ist, ist es möglich das ganze auf eigenen Servern komplett selbst zu bauen und somit auch das Problem der Sicherheit deutlich einzuschränken.
Das Feature, das Cloud9 sonst noch auszeichnet, ist das in Echtzeit gleichzeitige Arbeiten

auf einem Workspace. Damit ist es möglich live zu sehen was der jeweils andere Programmiert. Das sorgt zwar in der Regel eher für Chaos, es gibt allerdings auch Fälle in denen dieses Feature durchaus interessant sein kann, falls man zum Beispiel so etwas wie remote Pair-Programming betreiben möchte.

[Clo, 17]

3.6 Fazit

IDEs haben schon immer eine wichtige Rolle in der Software Entwicklung gespielt und werden es vermutlich auch weiter tun. Guckt man sich die frühen IDEs wie TurboPascal an, so ähneln diese sehr den einfachen Texteditoren wie Vim. Man kann zwar ohne Probleme noch mit ihnen Arbeiten, aber es fehlt einiges an Komfort. Wenn man nur wenige Jahre in die Zukunft springt jedoch, zum Beispiel zu TurboPascal 5 so sieht man hier schon deutlich mehr Features moderner IDEs wie Debugger oder Code completion. In einem Zeitraum von nur weniger Jahren haben sie die Entwicklungsumgebungen immens weit entwickelt und sind noch immer nicht am Ende. Moderne IDEs sind zwar deutlich fortschrittlicher mit objektorientierten Features, Plug-ins und Web-Integration, aber sie sind noch lange nicht perfekt und man kann mit Sicherheit noch mit einigen substanziellen Änderungen rechnen.

.

Bibliography

[Clo] Cloud9report. https://blog.jbrosi.ch/cloud9-ide-c9-io-ein-erfahrungsbericht/. Accessed: 2017-03-02.

[Dar] Dartmouth. https://en.wikipedia.org/wiki/Dartmouth_BASIC. Accessed: 2017-03-02.

[Ecla] Eclipsecdt. https://www.ibm.com/developerworks/opensource/library/os-eclipse-stlcdt/. Accessed: 2017-03-02.

[Eclb] Eclipsehistory. http://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F. Accessed: 2017-03-02.

[Eclc] Eclipsehistory2. https://www.ibm.com/developerworks/rational/library/nov05/cernosek/. Accessed: 2017-03-02.

[Ecld] Eclipseplugins. https://marketplace.eclipse.org/. Accessed: 2017-03-02.

[Ecle] Eclipsepublished. http://archive.eclipse.org/eclipse/downloads/. Accessed: 2017-03-02.

[Eclf] Eclispeosgi. https://www.osgi.org/developer/architecture/. Accessed: 2017-03-02.

[Ema] Emacs. https://codecraft.co/2014/05/13/why-you-should-use-an-ide-instead-of-vim-or-emacs/. Accessed: 2017-03-02.

[Mae] Maestro1. https://de.wikipedia.org/wiki/Maestro_I. Accessed: 2017-03-02.

[Pasa] Turbopascal. https://www.bernd-leitenberger.de/turbo-pascal-history.shtml. Accessed: 2017-03-02.

[Pasb] Turbopascalmanual. http://bitsavers.informatik.uni-stuttgart.de/pdf/borland/turbo_pascal/TURBO_Pascal_Reference_Manual_CPM_Version_3_Dec88.pdf. Accessed: 2017-03-02.

[Tur] Turboide. https://de.wikipedia.org/w/index.php?curid=531838. Accessed: 2017-03-02.

[Vim] Vim. http://vim.wikia.com/wiki/Use_Vim_like_an_IDE. Accessed: 2017-03-02.

[Visa] Visualage. https://en.wikipedia.org/wiki/IBM_VisualAge. Accessed: 2017-03-02.

[Visb] Visualagejava. http://javadude.com/articles/whyvaj.html. Accessed: 2017-03-02.

[Visc] Visualbasichistory2. https://h2g2.com/edited_entry/A599646#footnote3. Accessed: 2017-03-02.

[Visd] Visualbasicide. http://blog.rtwilson.com/wp-content/uploads/2011/05/VB6.jpg. Accessed: 2017-03-02.

[Vise] Visuaversions. http://www.antonis.de/qb2vb/index.htm. Accessed: 2017-03-02.