Die Software-Welt 2013

Ein Positionspapier

Ralf Grohmann
rg@rrgrohmann.de

Leonberg, im Juni 2013

Inhaltsverzeichnis

Historie

- Juni 2013 – Version 1.0

1. Motivation

Software wird als Thema in der Industrie quer durch alle Branchen immer wichtiger. Immer komplexere Systeme mit immer stärker variierendem Zugriffsverhalten und Fähigkeiten werden immer schneller vom Markt zur Lösung immer größerer Herausforderungen (Massendatenverarbeitung, Webseiten, ...) verlangt. Auch abseits der reinen IT Lösungen wird der Anteil von Software in eingebetteten Systemen mit zunehmender Leistungsfähigkeit bei gleichzeitiger Miniaturisierung und Leistungsaufnahme der Prozessoren und I/O Chips immer größer[1].
Somit muss die Produktivität der Entwickler immer weiter steigen, da immer komplexere Programme immer schneller und zuverlässiger produziert werden sollen. Einen besonders großen Sprung in den letzten 10 Jahren ist zum Beispiel in der Steuergeräte-Industrie zu bemerken. Vergleichen wir nur einmal die Komplexität eines ABS Systems, das verschiedene Sensoren in Echtzeit auswertet um Bremsbefehle zu generieren mit dem moderneren Spurhalte-Assistent, der in Echtzeit Kamera-Bilder, unter Umständen sogar in 3D, auswerten muss, um den Fahrer beim „abdriften“ zu warnen.
Eine solche Entwicklung ist nur noch unter Verwendung ausgefeilter Entwicklungsumgebungen und großer Bibliotheken, die komplexe Standard-Operationen „off the shelf“ bieten, zu leisten. Hier hat eine rasante Entwicklung eingesetzt. Gleiches gilt z.B. in der Medizintechnik, wo sich die Programmierung von kleineren Steuerungsaufgaben in Geräten schnell zu 3D – Visualisierungen auf Monitoren im Operationssaal entwickelt hat. Die Software hat also im Umfang von einigen Tausend zu Millionen Lines-of-Code zugenommen, die in beiden angesprochenen Industrien natürlich zuverlässig funktionieren müssen.
Analoge Entwicklungen sind jedoch auch in weniger kritischen Bereichen zu sehen, denn wer kommt heutzutage noch ohne Internet-Verbindung aus und verwendet regelmässig Software auf weltweit verteilten Servern ? Moderne Content-Management-Systeme für die Web-Präsenzen sind nicht mehr vergleichbar mit den einfachen Webseiten, die vor 10-20 Jahren noch bei vielen Firmen vorherrschten. Diese sind heutzutage kritische Erfolgskomponente jeder Firma (auch bei denen, die augenscheinlich nicht web-zentrisch sind), und müssen daher flexibel, skalierbar, zuverlässig, sicher, und erweiterbar sein.

Viele, insbesondere kleinere, Firmen können jedoch nicht genau einschätzen, wo sie im Vergleich mit der Konkurrenz in dieser Beziehung stehen, um einen eventuellen Investitionsstau oder schlimmer Schaden durch Hacker in der Softwareentwicklung rechtzeitig zu erkennen.

Daher stellt dieses Dokument die aktuellen Themen in der Softwareentwicklung und den zugehörigen Tools und Methoden kurz dar. Es kann natürlich keine Beratung zum Vorgehen in einem konkreten Szenario ersetzen, sondern nur ein erstes Erkennen der Situation erleichtern, indem über die Kategorisierungen die State-of-the-Art dargestellt werden soll.

1 Die gerade angekündigte „Google Glass“ Brille bringt ein komplettes Smartphone mit Kamera, GPS, Touch-bedienung, Internetzugang, Sprachein- und -ausgabe und Batter in einem etwas verdickten Gestell unter!

2. Software-Ökosysteme[2]

Über die Jahre haben sich in der IT Industrie verschiedene Lager ausgebildet:

1. **Die „Open Source“ Szene:** Die Basis aller Architekturen und Tools sind hier frei verfügbare Software, die aus verschiedenen Quellen stammt. Es gibt ein großes Ökosystem von Herstellern, deren Produkte auf diesem Stack basieren. Teilweise werden auch Produkte namhafter Hersteller in die Open Source gegeben – so hat z.B. IBM Eclipse als mächtiges Entwicklertool veröffentlicht. Die dabei verwendete Programmiersprache ist meistens Java, aber auch C/C++ findet oft, gerade in hardwarenahen Fällen, Verwendung. Hier gibt es auch die meisten Nischen-Anbieter und -Sprachen (Eiffel & Co.), die enthusiastisch gepflegt und weiterentwickelt werden.
   Auch das erfolgreiche Linux-Betriebssystem gehört natürlich hierher.
   Ein typischer Stack, der in vielen Webanwendungen Verwendung findet, ist der sogenannte „LAMP“ - Linux Apache MySql PHP. Es gibt auch leichte Abwandlungen davon (mit PostgreSQL, Pearl, Python), oder, als XAMPP, ohne die Bindung auf Linux (daher X).
   Es gibt eine äußerst reichhaltige Landschaft von Tools und Bibliotheken, die die Entwicklung moderner Anwendungen erlauben. Vom Software-Stack her ist vom Entwicklungstool (o.a. Eclipse, oder auch Netbeans), Betriebssystem (verschiedenste Linux-Versionen, verbreitet z.B. SUSE Linux, Red Hat, Ubuntu), Apache oder nginx und andere als Basis-Webserver, make, Maven oder auch Ant als Buildtool bzw. Hudson/Jenkins zur continuous integration des immer beliebteren Agile Entwicklungsprozesses alles vorstellbare verfügbar. Bugzilla kann zum Bugtracking, Mantis ist frei als Ticketsystem, GIT oder Subversion als Versionskontrollsystem, PostgreSQL oder mySQL als Datenbank, Hibernate als Persistenz-Schicht, verfügbar. RAP / RCP oder z.B. Spring kann als Programmiermodell und -library für Webanwendungen dienen – und die Liste könnte fast endlos fortgesetzt werden.
   Die meisten dieser Tools sind mittlerweile „Enterprise-ready“ und für den Einsatz in großen und komplexen Szenarien geeignet – und auch Linux, Apache und andere große Software selbst wird mit den genannten Tools erstellt[3].

2. **Das Microsoft-Universum**: Als größter Softwarehersteller der Welt hat Microsoft mit seinen Partnern ein komplettes Ökosystem für Software geschaffen.
   Basis bilden natürlich die verschiedenen Versionen des Microsoft Windows Betriebssystems, sowie die Office-Suite. Als Programmiersprache kommt in der Microsoft-eigenen Entwicklungsumgebung (VisualStudio) meist C# (gesprochen „C-sharp“) oder VisualBasic zum Einsatz; und die Webfähigkeiten werden über .NET in Kombination mit Sharepoint (der MS

2 Die Entwicklungs-Ökosysteme haben mittlerweile eine außerordentliche Vielfalt und Größe hervorgebracht, so dass ich hier unmöglich einen kompletten Überblick geben kann. Genauso bitte ich zu entschuldigen, sollte ich das eine oder andere Tool nicht 100% korrekt bezeichnen, da die mächtigen Tools oft Funktionalitäten mehrerer Kategorien Software bieten, die sich schlecht in einem Wort bzw. Satz darstellen lassen.

3 Selbst der Flugzeughersteller Airbus setzt bei Ihren Entwicklungsumgebungen auf einen OpenSource Stack, und gibt diesen wiederum als OpenSource frei – siehe http://www.topcased.org/

Webserver) realisiert.
Vorteile im Vergleich zu dem OpenSource Ökosystem ist die höhere Integration der Tools und Methoden, sowie professioneller Support aus einer Hand. Nachteilig natürlich die Lizenzkosten und -bedingungen (CAL, client access license), die gerade im Web-Bereich skalierbare Anwendungen teuer macht.

3. **Apple**. Im Gegensatz zu Microsoft wurde hier ein geschlossenes Ökosystem erstellt, in dem jede Komponente (Hardware und Software) eng kontrolliert wird. Wichtige Grundlage ist interessanterweise jedoch auch hier die Open-Source-Szene, denn Unix ist auch die Basis des MAC OS Betriebssystems.
   Erst mit dem Siegeszug des Internet und des Webbrowsers ist hier eine gewisse Öffnung festzustellen – und so werden die Apple-Rechner heutzutage gerne als Entwicklungsgeräte für Open-Source basierte Webanwendungen verwendet.

4. In speziellen Einsatzbereichen gibt es weitere kleinere Ökosysteme, z.B. ist die IBM zSeries Großrechnerwelt mit dem Betriebssystem MVS nach wie vor bei Großfirmen breit im Einsatz. IBM setzt dort mittlerweile jedoch auch auf Linux als Basis, und es ist viel Open Source Zusatzsoftware verfügbar.

Mobile Anwendungen: Auch existiert hier eine Zwei- bzw. Dreiteilung des Marktes in Open-Source (hauptsächlich Android von Google), sowie die kontrollierten Märkte Apple (iPhone) und Microsoft (Windows Phone). Bada von Samsung und auch ältere Betriebssysteme von Nokia, Motorola & Co spielen kaum noch eine Rolle.
Auf der Browser-Seite konnte sich das ursprünglich von Apple entwickelte „Webkit“ als Basis vieler mobiler Browser etablieren (es stammt ursprünglich jedoch wieder von Khtml, einem Programm des KDE Desktop der Linux Systeme ab …) . Hier zeichnet sich aktuell jedoch, von Google getrieben, bereits wieder eine Aufspaltung ab.
Zur Anwendungsentwicklung stehen sowohl die jeweils nativen Entwicklungs-umgebungen der Betriebssystem-Hersteller zur Verfügung, es gibt jedoch auch einige Plattformübergreifende Bibliotheken und Umgebungen mit jeweils verschiedenen Vor- und Nachteilen, z.B. PhoneGAP.
HTML5 als mächtige nächste HTML-Version besitzt auch das Potential, Anwendungen Betriebssystemunabhängig zu entwickeln die ähnlich mächtig wie native Apps sein können. Dieses Feld ist jedoch nach wie vor stark in Bewegung, HTML5 noch nicht komplett standardisiert, und es sei daher auf Wikipedia und ähnliche Internetquellen für Details verwiesen.

3. Webtechnologie

Wie oben erwähnt können sich die proprietären Ökosysteme dem Web-Trend natürlich nicht entziehen, und so werden immer wieder OpenSource-Technologien verwendet bzw. emuliert, um einen einfachen Datenaustausch über Betriebssystemgrenzen hinweg zu erlauben.

Darunter fallen neben den Basis-Internet-Technologien wie TCP/IP und Socket-basierte Kommunikation auch die höheren Schichten, wo dann HTML (hypertext

transfer markup language), AJAX (asynchronous javascript and xml), oder REST (representational state transfer) im Browserumfeld eine Rolle spielen.

Weitere unterstützende freie Software wie z.B. die Verwendung von Blogs oder Wikis als kollaborative Dokumentenablage findet auch immer mehr Verbreitung.

Der heute meistverbreitete Webserver im Internet ist der Apache HTTP Server. Er findet auch in kommerziellen Servern wie z.B. den IBM WebSphere Application Server Verwendung, und stellt die Basis für Webauftritte bereit.

Man unterscheidet *Server-Seitige Erweiterungen* – meist um Daten dynamisch abzufragen und darzustellen, sowie *Client-Seitige Technologien* um die Webseite vom Look-and-Feel her an native Anwendungen anzunähern (z.B. Dropdown-Menues, Einblendungen, Grafiken, Mausaktionen des Nutzers auswerten).
Eine Vielzahl von PlugIns bzw. Modulen erweitern den Webserver mit Funktionalität. Das beginnt bei den Scriptsprachen wie PHP, Python, Perl, Ruby, JavaScript, oder Microsoft's VB.NET. Damit kann man die gelieferten Seiten dynamisch erzeugen.
Für große Enterprise-Anwendungen auf der Serverseite wurde J2EE (Java Enterprise Edition) entwickelt, und verschiedene Webserver implementieren dies – z.B. der offene JBOSS J2EE Server, oder IBM's WebSphere. Microsoft hat für ihren Webserver analog auf C# basierende Technologie entwickelt (ASP.NET).
Diese Serversoftware erlaubt eine Sitzungs-orientierte Behandlung der Nutzerinteraktion, Persistenz von Daten über mehrere Sessions hinweg, und integrieren z.B. Datenbanksysteme oder andere Enterprise-Systeme in Web-Auftritte.
Für Webseiten, die User-Accounts anlegen und damit eine Personalisierung erlauben verwendet man Portale, die es auch wiederum in Open-Source Ausprägung (z.B. LifeRay) als auch kommerziell gibt (MS Sharepoint bzw. IBM WebSphere Portal Server). Sie erlauben einem User, einmal eingeloggt, eine funktionell angepasste Oberfläche zu bieten, unterstützen ein Fensterkonzept innerhalb des Browsers die mit der Maus verschiebbar sind, und verringern damit weiter den „Abstand“ zwischen Webanwendung und einem lokalen Programm.
Auf der *Client-seite* gibt es neben den HTML Funktionen, die mit HTML4 und der Einführung der CSS (cascading style sheets) bereits deutlich erweitert wurden die Möglichkeit, mittels JavaScript (und darauf aufbauenden AJAX-Bibliotheken) die Mächtigkeit der Darstellung zu erhöhen, ohne jedes mal eine neue Seite vom Server anzufordern. Ausserdem gibt es eine Vielzahl von Erweiterungen, z.B. den Adobe Flash Player, der neue Funktionen im Browser ermöglicht. HTML5 stellt, wie vormals erwähnt, hierbei den nächsten Evolutionsschritt dar.

Und wo sind jetzt die Webserver installiert, auf die zugegriffen werden soll ? Natürlich in Rechenzentren, die entweder in der Firma (kleinste Ausprägung: Unter dem Tisch des Entwicklers), bei einem RZ-Hosting Anbieter oder „in der Cloud“ laufen. Anhand dieser Frage lassen sich die heute üblichen Serverlandschaften gut aufzeigen:

4. Serverarchitekturen

4.1 Rich Client Anwendungen

Hier wird ein Programm lokal auf einem Desktop- oder Laptop-Rechner installiert. Nach wie vor üblich für Anwendungen, die nur ein Benutzer selbst verwendet, z.B. CAD-Zeichenprogramme, Produktivitätsanwendungen (Textverarbeitungen, Tabellenkalkulation, Präsentationsprogramm, Taschenrechner, etc).
Diese Kategorie nimmt immer mehr ab, da zum einen die Internet-Technologien immer leistungsfähigere Programme erlauben, zum anderen Updates und Umzüge auf neue Rechner mühsam sind.

4.2 Client – Server Anwendungen

Dies war der nächste Entwicklungsschritt - mit Verbreitung von lokalen Netzwerken, die die Computer koppelten (Ethernet oder Token Ring). Nach wie vor wird hier ein lokales Programm als Client installiert, es kann jedoch maßgebliche Funktionen nur in Verbindung mit einem Serverprogramm bereitstellen. Die Synchronisation der Arbeit verschiedener Nutzer geschieht auf dem Server.
Einfachstes Beispiel einer solchen Anwendung ist der Datei-Explorer und die Bereitstellung eines Netzlaufwerks auf einem Server. Noch bis weit in die 90er Jahre war dies die übliche Software-Architektur, und Windows z.B. gibt es nach wie vor als Client und als Server-Version (die sich jedoch praktisch nur noch durch Lizenzbedingungen unterscheiden).

Der nächste Entwicklungsschritt kam dann mit der zunehmenden Verbreitung des Internet (TCP/IP – basierte Kommunikation) in Verbindung mit HTML (hypertext markup language) als Beschreibungssprache, die basierend auf HTTP (hypertext transfer protocol) ausgetauscht wurde. Es wurde jetzt möglich, Informationen auf Seiten, eine Datenanzeige und auch einfache Eingabemasken abstrakt zu beschreiben und per Browser dem Nutzer anzeigen zu lassen. Der „Client“ der Client-Server-Anwendung wurde damit vereinheitlicht, und ganz ohne spezielle Client-Software auf dem lokalen Rechner konnten dem Nutzer Funktionen eines Servers zur Verfügung gestellt werden:

4.3 Browser-basierte Anwendungen

Es begann 1989 am CERN in Genf mit einfachem formatiertem Text und Eingabemasken, ganz im Sinne der alten Großrechner-Terminal-Technologie. Sie wurde jedoch in der Wissenschaft schnell aufgenommen da sie jedem mittels eines Browsers einheitlichen Zugriff auf die jeweiligen Server mit den Informationen erlaubte.
Der erste grafische Browser war übrigens „Mosaic“, aus dem sich später Netscape entwickelte und dessen Nachfahre heute Firefox ist.
Schnell wurde die Mächtigkeit von HTML erweitert, z.B. kamen in HTML 3 Auswahlboxen und weitergehende grafische Gestaltungsmöglichkeiten dazu. Serverseitig wurde es in den Webservern zunehmend ermöglicht, Programmfragmente einzubinden (meist per Javascript). Zuerst nur, um z.B. Daten auf einer Seite dynamisch aus Datenbanken laden zu können.
In den 2000'er Jahren gelang dann mit AJAX (asynchronous javascript and XML) und

REST (representational state transfer) ein weiterer Durchbruch. Neben weitaus dynamischeren Nutzerfunktionalitäten (Verschieben von Seitenelementen mit der Maus, Markierungen setzen, erweiterte Fenstersteuerungen) wurde dadurch auch der bis dato übliche „Seite-Anzeigen – Daten Eingeben – Seite Neu Anzeigen“-Ablauf im Browser durchbrochen. Änderungen kleiner Bereiche einer Seite im Browser führten nicht mehr zu einer vollständigen Neuzeichnung der Seite, sondern nur die betroffenen Bereiche werden neu „gerendert“ – nach Eingabe einer neuen Suche wird z.B. nur der Ergebnisausgabe-Bereich neu angezeigt, nicht die Umrandung und der Such-Teil einer Seite. Dies war ein großer Sprung in der Bedienbarkeit für den Nutzer, und eine Annäherung der Funktionalität an ein lokales Programm. Es tauchten auch immer weitere, leistungsfähige Bibliotheken für auf, die den Abstand zu Rich-Client-Apps (also lokal installierte Programme) immer weiter verringerte.

XML – an dieser Stelle ist eine Erwähnung von XML (eXtensible Markup Language) als dem Internet / Browser zugrundeliegende Datenstruktur angebracht. Schon früh, basierend auf SGML (standard general markup language) wurde HTML in den 80/90ern als ein weiteres Markup entwickelt, was sich wie oben geschildert schnell verbreitete. Es war aber nicht dafür geeignet, beliebige Daten zu beschreiben. Daher wurde XML als weiteres Markup basierend auf SGML entwickelt, und somit wurden früher verwendete binäre Datenstrukturen durch eine textbasierte Beschreibung abgelöst Dies brachte den Vorteil der Lesbarkeit und einfacheren Übertragbarkeit. XML wird mittlerweile in praktisch jedem Interface als Transferformat verwendet. Durch die Beschreibung der Datentypen und Definition derselben in einer separaten Datei, dem XML-DTD (document type descriptor) bzw. XML Schema (das DTD wieder in XML Format spezifiziert) wurde außerdem die Trennung von Daten und Formatierung erreicht, was eine Flexibilisierung der Darstellung erlaubte.
Im Falle HTML wird die Formatierung seit HTML V3 in einer separaten CSS Datei gespeichert. Die Browser lesen die HTML Beschreibung der Seite sowie die CSS Formatierungsinformation und legen sie dann in einem hierarchischen DOM (Document Object Model) ab, welches zur Angezege verwendet wird. Mittels des DOM können auch Programme auf beliebige Seitenelemente zugreifen (z.B. für AJAX Operationen.

Schon um die Jahrtausendwende wurde von SUN mit dem „Thin Client“ ein PC vorgestellt, der gar keine lokale Datenablage mehr hatte – wer erinnert sich noch an den Slogan „the network is the computer“? Heutzutage wagt Google mit den Chromebooks einen neuen Anlauf dieses komplett Internet/Cloud/Server-zentrischen Ansatzes.

In den 2000'ern wurden die Internet-Firmen immer größer: Amazon, Ebay, und natürlich Google wuchsen und wuchsen. Deren IT Systeme basierten rein auf Browser-Clients, und diese Firmen lernten, weltweit verteilt riesige Serverfarmen zu betreiben.

4.4 Cloud Computing

Amazon war eine der ersten Firmen, die auf die Idee kamen, Ihre skalierbaren und weltweit verteilten Server auch direkt Kunden zur Verfügung zu stellen. Man konnte

also für wenige Cent pro Minute Server-Rechenkapazität flexibel einkaufen. Sie nannten es Amazon EC2 - „Elastic Cloud“, und dies war die Geburtsstunde des Cloud computing.

Zunächst ging es rein um virtuelle Server (*Infrastructure-as-a-service*), man bekam also einen Rechner plus Betriebssystem zur Verfügung gestellt, oder im noch einfacheren Fall wurde nur Speicherplatz „provisioniert“. Später konnte man auch komplette Softwarepakete vorinstalliert buchen, z.B. ein Server der ein Web Portal komplett konfiguriert enthielt (*Platform-as-a-service*), in wenigen Minuten nach der Bestellung hochgefahren und betriebsbereit.

Einer der ersten und damit erfolgreichsten StartUps, die noch einen Schritt weiter gingen, war SalesForce.com, die ein komplettes CRM (Customer Relationship) System anboten. Der Kunde konnte alle seine Daten auf SalesForce Servern halten und das System entsprechend seiner Wünsche konfigurieren und erweitern. Dafür war nur eine monatliche Rate pro Benutzer fällig, und sämtliche lokalen IT-Server Ausgaben konnten damit eingespart werden: *Software-as-a-service* war geboren. Man könnte jedoch auch bereits die web-basierten Email-Dienste (gmail.com, gmx.de, hotmail.com etc) als erste Software-as-a-service bezeichnen.

Mit den aufkommenden Smartphones und vor allem Tablets in den späten 00'ern wurden dann auch die Einsatzbereiche für diese Serveranwendungen breiter gefasst, und es wurde nötig, dass die Webseiten bzw. Server im Internet verschiedene Darstellungen ihrer Inhalte beherrschten, je nachdem ob ein PC, ein Tablet Computer oder ein Smartphone auf sie zugreift. Dabei sind insbesondere die verschiedenen Nutzungsszenarien zu berücksichtigen: von einem Smartphone aus will man meist gezielt zu einer bestimmten Information gelangen (Öffnungszeiten, Hotline Nummern, kurz ein Zimmer buchen etc.) während man mit Tablets und PC's auch einfach mal herumstöbern will und erweiterte Such- und Konfigurationsmöglichkeiten wünscht.
Im Businessumfeld entspricht das der kurz-Datenabfrage bzw. -eingabe mit dem Smartphone, während man im Tablet / PC eher eine Gesamtsicht der Business-anwendung benötigt.

Der nächste Schritt des Internet-basierten Paradigmas ist die Verkettung verschiedener Dienste – wir sehen es z.B. oft bei einer Einbindung von Google Maps in Webseiten als Anfahrtsbeschreibungen, oder auch beim Zahlungsprozess in Webshops. Dazu werden *Webservices*[4] verwendet, die auch wiederum auf XML als Datenstruktur basieren. Hier ergeben sich insbesondere im Geschäftsumfeld jedoch auch vielfältige Datenschutz-, Sicherheits- und Abrechnungsthemen, die zum Teil erst noch geklärt werden müssen. Daher findet in diesem Feld nach wie vor eine stürmische Entwicklung statt.

4 Womit klar ist, dass die Softwarearchitektur dieser Systeme der Service-orientierten-Architektur (SOA) folgt.

5. Softwarekategorien

Es folgt ein Versuch, Software in Kategorien einzuteilen. Dies ist äußerst schwierig, da man hier immer weiter unterteilen könnte, und sich auch ständig neue Kategorien bilden. Ich probiere es trotzdem:

1. Betriebssysteme
   Unix und Derivate wie Linux, Windows, MacOS, MVS, viele Spezial-betriebssystem, insbesondere im Echtzeit-Bereich, wobei auch hier ein starker Trend zu erkennen ist auf Linux aufzubauen. Beispiel: CentOS.
2. Entwicklungsumgebungen
   Eclipse, Netbeans, VisualAge, VisualStudio, Rational Application Developer, u.v.a.m.
   Hierzu gehören auch die meist Server-basierten Tools, die für die Softwareentwicklung benötigt werden: Build-Tools, Repositories, Testautomatisierung, Dokumentationsgeneratoren, Syntax Checker, etc. die ich z.T. In Kapitel 2 bereits erwähnt habe.
3. Entwicklungsbibliotheken
   zur leichteren Erstellung leistungsfähiger Programme. Es gibt tausende Bibliotheken beginnend bei den Basis-Bibliotheken der Programmiersprachen für mathematische Operationen, String-Handhabung, Pattern Matching, Grafikfunktionen, Datenbankzugriffen, oder Hardwaregeräte-Anbindung. Ein paar Beispiele weiterführender Bibliotheken für Java sind z.B. Swing, Spring, Hybernate, RAP RCP. Es gibt das Windows API bzw. .Net mit großem Funkionsumfang im Windowsumfeld, und DirectX als fortgeschrittenes Grafiklibrary. Eine andere, plattformunabhängige verbreitete Grafikbibliothek ist QT, und für spezialfälle kommt auch OpenGL zum Einsatz. Letztendlich ist auch das ur-alte XWindows-Grafik-System als integraler Bestandteil der Oberfläche von Linux weit verbreitet.
4. Scriptingsprachen
   werden zur Automatisierung von Befehlen und Erstellung überschaubarer Programme verwendet: Das sind zum einen Shells selbst (C, bash, zsh, etc in Linux, oder der Windows Command interpreter und Erweiterungen davon), aber auch mächtigere Sprachen wie Perl, Python, oder Ruby-on-Rails. JavaScript als von jedem Browser unterstützte Sprache darf hier nicht fehlen.
5. Programmiersprachen
   Hierunter fallen Sprachen wie Assembler, C, C++, C#, Java, Fortran, Cobol, Basic und viele hundert weitere. Diese stellen i.V. Zu den Scriptsprachen weitere Konstrukte zur Verfügung die es erlauben auch große Projekte damit zu erstellen.
6. Datenbanken
   Da praktisch jede nichttriviale Anwendung Daten speichern muss, hat sich das Feld der Datenbanken mit der zunehmenden Computerisierung stürmisch entwickelt.

Es gibt heutzutage folgende Typen[5]:

- Hierarchische Datenbanken, (z.B. IMS) – eines der ältesten Typen von Datenbanksystemen. Findet heute keine neue Verwendung, wird jedoch nach wie vor vor allem auf IBM Großrechnern für kritische (meist ältere) Programme eingesetzt.
- Relationale DB (DB/2, Oracle, mySQL, Postgresql, uvam.) – der wohl verbreitetste Datenbanktypus. Mit SQL wurde eine mengenbasierte Abfragemethodik eingeführt, die die Arbeit mit Daten erheblich erleichtert da die Navigation im Suchbaum nicht mehr durch das verwendende Programm selbst durchgeführt werden muss.
- Objektorientierte Datenbanken – der Schritt weg von „einfachen“ Rohdaten (Strings, Zahlen) hin zu komplexen Objekten die einfach gesucht werden können.
- In-Memory Datenbanken (SAP Hana, Oracle TimesTen, IBM SolidDB, Hyper SQL) – da die Hauptspeicher im GB Bereich in den letzten Jahren immer günstiger wurden, erlaubt diese Technologie deutlich erhöhte Zugriffsgeschwindigkeiten auf die Daten.
- Dokumentenzentrische DB's / NoSQL: CouchDB, Lotus Notes uvam. - Weiterentwicklung der OO-DB's auf Dokumente und entsprechende Such-Möglichkeiten.
- Special-Use-Datenbanken: Es gibt einige Spezialtypen für besondere Anwendungsfälle, z.B. Echtzeit-Datenbanken, die eine gewisse Zugriffsgeschwindigkeit garantieren, oder Datenbanken für integrierte Systeme, die besonders klein sind bzw. wenig CPU-Power benötigen. Diese werden jedoch immer seltener, da die Standard-Produkte einen immer breiteren Anwendungsbereich gut abdecken.

7. Business Analytics Software / „Big Data“:
   Entstanden als Ergänzung bzw. Erweiterung zu leistungsfähigen Datenbanksystemen, mittlerweile eine eigenständige Softwarekategorie. Es geht hier darum, große Datenmengen schnell zu durchsuchen und Muster zu erkennen (meist um daraus verkaufsfördernde Schlüsse zu ziehen – Beispiel: Aus den Verkaufszahlen im Supermarkt Schlüsse zur Verbesserung der Warenplatzierung zu erhalten).
   Man sagt, die US Regierung prüft den gesamten Internet-Verkehr auf terroristische Aktivitäten, falls wahr ist dies sicherlich die größte „Big Data“ Anwendung[6].
   Da die Datenmengen immer schneller steigen (z.B. auch durch eine Vielzahl von Sensoren / Videocameras), wird diese Software-Kategorie immer größer.
8. Dokumenten- und Content-Management-Software:
   Dokumentenablagesoftware dient der Archivierung und Indexierung der Dokumenten-Flut in Firmen (viele Anbieter).
   Content Management wird verwendet, um Inhalte von Webseiten zu erstellen, verwalten, verändern und zu publizieren. (z.B. Typo3, Joomla,

5 Die Datenbanken der großen Anbieter decken oft Fähigkeiten aus mehreren Kategorien ab.

6 Interessant, dass ich diese Zeilen ca. 2 Wochen bevor der „PRISM“ Skandal in den USA über die Internet-Spionage aufkam, schrieb! Das „falls wahr“ kann also gestrichen werden.

sowie andere komerzielle Anbieter)

9. Produktivitätssoftware, d.h. Email- und Kollaborationssoftware, Office-software, Tools. Hier gibt es eine sehr große Anzahl von Produkten, nur ganz auszugsweise:
   Microsoft bietet u.a. Sharepoint 2013, Office365, Outlook, Yammer (zur Collaboration), Lync (Unified Communications) an. Das entsprechende IBM Portfolie beginnt mit Lotus Notes, Sametiem und Connections.
   Es gibt viele PDF Editoren bzw. Viewer von Adobe oder anderen Herstellern.
   Open Source Lösungen sind: LibreOffice, OpenOffice, Thunderbird (Email), Pidgeon (Collaboration) und viele andere .
   Im Internet Umfeld fallen hier auch die allgegenwärtigen Wikis und Blogs herein.

10. Systemmanagementsoftware
    Wichtig vor allem in Firmen und Rechenzentren. Privatleute haben damit weniger Kontakt (außer als Beigabe für Heim-Netze). Ein Beispiel ist HP OpenView, und es gibt mit SNMP (simple network management protocol) ein immer weiter standardisiertes Protokoll zwischen den Geräten und der Management Software.

11. Geschäfts- oder „Business“-software:
    Hierunter fällt für mich Finanzsoftware, Buchhaltung, Mitarbeiter- und Personalverwaltung, CRM (Customer Relationship Management), ERP (Enterprise resource planning) Lagerverwaltungssoftware, Bankensysteme, und vieles mehr.
    Große Player sind SAP, Oracle, Microsoft, es gibt aber auch viele kleinere Anbieter, in Deutschland z.B. Lexware.

12. Internet-Basissoftware
    Kommunikationssoftware (heutzutage normalerweise im Betriebssystem als TCP/IP Stack enthalten).
    BROWSER: Mittlerweile die nach dem Betriebssystemen vielleicht verbreitetste Softwarekategorie: Internet Explorer, Firefox, Chrome, Opera und andere.
    VPN Software zur Verschlüsselung und Absicherung der Kommunikation (z.B. Viprinet, …)

13. „Cloud-Software“
    Webserver und -Portale, Webshops, Online-Speicher, Bilderspeicher, ...
    Firmen: Salesforce.com, Google+, Facebook, Amazon, eBay, Twitter, Flickr, tausende mehr.
    Da es mittlerweile fast keine Software mehr gibt, die nicht das Internet verwendet / einbindet / benutzt, geht der Trend dazu, eher die Funktionalitäten zu beschreiben und nicht das Internet als Technologiekategorie zu verwenden. Social Software z.B. ist ohne das Internet als Kommunikationsmedium nicht denkbar.

14. Spezialsoftware
    für CAD (computer aided design), Grafik (bis hin zu 3D Spielfilmerstellung), Mathematik und Simulationen, und last but not least Spiele.

15. Eingebettete Software
    In fast jedem elektronischen Gerät ist heutzutage einiges an Software zu finden, und je komplexer das Gerät desto mehr Software wird benötigt. Hierunter fällt also z.B. Software für den Radio-Wecker, den Backofen, aber auch für spezielle Bedürfnisse, z.B. zur Steuerung von Flugzeugen, Satelliten, Schiffen, Autos und anderem mehr.
    Die Grenzen zwischen eingebetteten und Desktop bzw. Laptop Systemen sind dabei fließend, da z.B. die Smartphones recht kleine Geräte sind, aber mittlerweile Software beinhalten, die vor 10 Jahren noch nicht einmal auf leistungsfähigen PC's lauffähig gewesen ist. Software speziell für embedded Geräte verschwindet somit zunehmend, da diese auch die normalen Versionen der Software verwenden können – auch ein Smartphone hat heutzutage mehrere GB Speicher, Multi-Core Prozessoren mit GHz-Taktraten sowie FullHD–Touch-Displays.

6. Entwicklungsmethoden

In diesem Abschnitt soll es nun um eine Kurzübersicht über die gebräuchlichen Entwicklungsmethoden gehen. Software-Entwicklung war und ist ein komplexer Prozess, und es gibt nach wie vor Glaubenskriege, ob es sich um einen kreativen, ja künstlerischen Akt handelt oder ob es möglich ist, durch Regeln und strikte Vorgehensweisen eine Art industriellen Produktionsprozess für Software zu definieren.

Während viele Bücher immer wieder von letzterem, der industriellen Softwareerstellung, reden, ist die Verbreitung des agilen Entwicklungsansatzes sowie die mit Ihm einhergehende Klarstellung, dass der Entwicklungsprozess nicht fest planbar ist, ein Zeichen, dass es doch ein kreativer Akt ist. So ist die Softwareentwicklung eben doch nach wie vor nicht mit komplexen Ingenieurs-projekten wie Brücken- oder Hochhausbau vergleichbar.
Die existierenden Ansätze werden weiter unten erklärt, unabhängig vom gewählten Ansatz sind jedoch immer folgende Schritte in verschiedener Ausprägung zu erledigen:

- Zieldefinition / Projektdefinition / Nutzungsszenarien aufstellen (Use Cases)
- Aufwandsschätzung (diverse Methodiken).
- Architektur & Design erstellen, Entscheidungen zu verwendeten Tools, Zielumgebung, Lebensdauer, Risikobetrachtungen.
- Entwicklungszyklus durchführen, bestehend aus Codieren, Testen, Dokumentation, Reaktion auf geänderte Anforderungen.
- Installation.
- Produktivbetrieb.

Je nach Größe des Projekts können diese Schritte natürlich mehr oder weniger intensiv ausfallen. Für das hier beschriebene „Testen“ z.B. gibt es mehr als 10 Ausprägungen in verschiedenen Phasen eines Projekts – so können schon die erstellten Nutzungsszenarien auf Korrektheit geprüft werden, der Entwickler macht Unittests für erstellten Code, es gibt Funktionstests für Komponenten, einen Systemtest, der das Gesamtsystem am Ende prüft, einen Integrationstest, ob die erstellten Komponenten zusammenpassen, und auch Übersetzungstests und

Performanztests.

Bei einem kleinen Projekt, im Minimalfall mit einem Entwickler, können sogar einzelne Schritte wegfallen. Dann erübrigt sich auch eine größere Projektverwaltung, da Kommunikationsprobleme nicht auftreten können. Auch hier kann jedoch z.B. ein Auftraggeber etwas ganz anderes gewünscht haben als der Entwickler umsetzt. Außerdem kann es sein, dass ein anderer Entwickler die Wartung der Software übernimmt, womit auch bei einer solchen Entwicklung darauf geachtet werden muss, z.B. sauber zu dokumentieren.

Es gibt verschiedene Entwicklungsansätze für größere Projekte:

Das Wasserfall Modell

Ein streng organisiertes in eine Richtung ablaufendes Projekt. Genaue Planung, dann Design und Architektur, dann Entwicklung und Dokumentation, Test, und schließlich die Auslieferung.
Diese Methodik war Ausdruck der Zeit, in der man fest an die „Industrialisierung“ der Softwareentwicklung glaubte, und dass alles eine Frage guter Planung sei. Gerade bei kritischer Software (Flugzeuge, Medizinbereich etc.) klingt dies in der Theorie auch sehr gut und wird daher bis heute verwendet und oft von den Entwicklungsnormen gefordert. Leider hat es in Realität auch in den kritischen Bereichen, in denen man viel Aufwand für die Planung reserviert, die Erfolgesqupte nicht erhöht. Ein Hauptgrund sind die unabsehbaren Änderungen, die während der Entwicklungszeit auf das Projekt einwirken, z.T. mit fatalen Folgen. Insbesondere bei Projektlaufzeiten über einem Jahr änderten sich Anforderungen, die Technologie macht Fortschritte, und oft passiert es auch, dass der Kunde, kaum dass er eine erste Version des Programms sieht bzw. testet, merkt, dass es andersherum doch besser wäre.

Agile Entwicklung

Kennzeichnet eine Reihe von Methodiken, die auf o.a. Herausforderung der ständig einströmenden Änderungen während eines Projektes eine Antwort suchen. Verbreitet ist die SCRUM Methodik.
Ein Projekt besteht aus überschaubaren Zyklen (Sprints) von 2-6 Wochen, für jede dieser Zyklen werden Funktionalitäten geplant, implementiert, getestet und nach jedem Zyklus steht eine lauffähige neue Softwareversion zur Verfügung. Diese wird vorgeführt, und der Feedback des Stakeholders kann in die weitere Planung einfliessen. Es können auch Funktionen in einen späteren Sprint verschoben werden, wenn sich herausstellt, dass sie den Zeitrahmen sprengen. Die Maxime ist, dass Zeit wichtiger ist als Funktion, d.h. ein Sprint muss immer in der geplanten Zeit abgeschlossen werden.
Auch eine weitere große Projekt-Problem-Quelle wurde angegangen: Die Kommunikation. Die Entwicklerteams sind auf maximal 7 Personen beschränkt , und es gibt ein tägliches Stand-up Meeting zum Informationsaustausch. Am Ende eines Sprints gibt es eine große „Demo“ wo den Stakeholdern das Erreichte gezeigt und der nächste Sprint besprochen wird. Das wird oft in Form einer kleinen Motivationsfeier veranstaltet. Größer werden die Projekte nicht, indem man die Sprint-Teams vergrößert, sondern indem man mehrere, unabhängige Teams die

jeweils einen eigenen Sprint erledigen, einführt.
Auch innerhalb der Entwicklung werden Fehler-vermeidende Tätigkeiten forciert, z.B. die Team Programierung (d.h. gemeinsam kodieren) und Code Reviews. Außerdem werden auch genaue Rollen festgelegt, die vermeiden sollen, dass am Bedarf vorbei entwickelt wird bzw. unklar ist wer welche Aufgaben und Verantwortungen trägt. Schlüssel hierbei ist der „Product Owner“, der die Stakeholder, d.h. Auftraggeber, vertritt, sowie der Sprint Master, der jederzeit den Überblick behält und auch über die Einhladung der Agilen Regeln wacht.

Weitere Methodiken, die in eine ähnliche Richtung gehen bzw. Teilbereiche davon abdecken, sind Extreme Programming, Lean Development, Feature driven Development, oder Behavior driven development, Test driven development.

Diese modernen Methoden der Softwareentwicklung stellen den Entwickler in den Vordergrund, basierend auf der Erkenntnis, dass hier der größte Einflussfaktor auf den Projekterfolg und die Software-Qualität sitzt. Dies wird oft durch die Reihenfolge **„People Processes Technology“** dargestellt.
Damit wird auch klar, dass die letztendlich verwendete Technologie den geringeren Anteil an Erfolg/Misserfolg eines Projektes hat, sondern die Teamarbeit und gegenseitige Verantwortung für den Erfolg maßgeblich sind. Gerade daher ist bei Agile eine maximale Teamgröße definiert, und es muss auch nach jedem Zyklus (Sprint) ein Quasi-Endprodukt geben.

7. Software-Qualitätsnormen

Auch die beste Entwicklungsmethodik kann schlecht oder gut implementiert sein, und sie lebt natürlich auch der Expertise des Teams. Außerdem ist das Entwicklungsprojekt nur ein kleiner Schritt im Lebenszyklus von Software. Insbesondere die Wartung, d.h. Fehlerbehebung, Anpassung an neue IT Systeme und Technologien, Funktionserweiterungen und ähnliches während des Betriebs ist ein großer und meist auch kostenintensiver Teil des Lebenszyklus – und wird gerne unterschätzt. Wartung und Dokumentation ist neben intensiven Testen daher wichtiger Teil des Entwicklungsprozesses (Design for maintenance).

Die meisten übergreifenden Regeln und Normierungen berücksichtigen den gesamten Lebenszyklus:

Das **“CMM“** – Capability Maturity Model, in neueren Versionen auch CMMI (I für integration) genannt – erlaubt, den Reifegrad einer Organisation von Entwicklung bis zum Betrieb und Wartung einzuschätzen. Es gibt 5 Stufen, auf denen sich eine Organisation befinden kann – Initial, Repeatable, Defined, Managed, Optimizing. Aufgrund gewisser Verhaltenscharakteristika, die abgefragt werden, wird die Erreichung eines dieser Stufen begründet.

Die **ISO 9001** als allgemeine Qualitätsnorm deckt ein noch breiteres Spektrum ab, ein höherer CMMI Level ist aber sicherlich hilfreich für die ISO Zertifizierung.

Eine andere, konkurrierende Normierung ist **SPICE / ISO 15504.** Auch hier wird der Focus auf Software gelegt, aber im größeren Zusammenhang des Lebenszyklus

gesehen.

Es existieren verschiedene Spezialversionen der o.a. Normen und Methodiken sowie Implementierungsdefinitionen. Bekannt ist z.B. die **„automotive SPICE“** als Spezialisierung der Norm für die Softwareentwicklung in der Automobilindustrie.

In der Medizintechnik wird auch immer mehr Softwareentwicklung notwendig. Aufgrund der besonderen Anforderungen an Zuverlässigkeit und Fehlervermeidung gibt es, in Deutschland im Medizinproduktegesetz (**MPG**) festgehaltene, Detail-Normierungen. Die im MPG referenzierte Norm ist (u.a.) die **DIN EN/ISO 13485** und beschäftigt sich mit der Qualität bei der Medizinprodukte-Herstellung. Sie gliedert sich in eine Vielzahl (um die 100!) Detailnormen auf, je nachdem in welchen Bereich ein Medizinprodukt erstellt wird (von Pinzetten, Spritzen über medizinische Sensorgeräte bis hin zu Implantaten). Arzneimittel zur Einnahme selbst sind nochmal separat abgedeckt.

Auch das Projektmanagement ist natürlich wichtiger Bestandteil des Entwicklungsprozesses und wird auch in o.a. Normen gefordert. Hier gibt es 3 große Verfahren bzw. Normierungsgremien:

- PMI – Projekt Management Institute mit dem PMP (PM Professional),
- Prince2 – OCG, vom britischen Government ursprünglich definiert oder
- IPMA - International PM Association.

In Deutschland wird Projektmanagement in DIN 69901 definiert.

8. Fazit

Ich hoffe, dieser Durchmarsch durch die verschiedenen Themenblöcke der Software-Entwicklung war aufschlussreich. Insgesamt ist zu sagen, dass sich der Fortschritt der Softwareentwicklung nach wie vor schnell vorangeht, in der Theorie zeichnet sich jedoch langsam eine Komplettierung ab.
Softwareentwicklung ist nach wie vor sehr aufwändig, jedoch kann man heute bei gleichem Aufwand deutlich bessere Software nach allen Qualitätsmerkmalen (Stabilität, Performance, Benutzerfreundlichkeit) erstellen als noch vor 10 Jahren und davor. Dies ist sowohl der agilen Entwicklungsmethodik als auch den ständig mächtigeren Tools / Entwicklungsumgebungen und Bibliotheken zu verdanken (Beispiel: Google Maps als grafisches Kartenanzeige kann man heute in ein paar Stunden in sein Programm einbinden, früher wäre etwas vergleichbares eine mehrwöchige Aktion gewesen … wenn man denn überhaupt auf damaligen Rechnern Karten solcher Güte hätte anzeigen können).

Ich hoffe auch, dass diese Beschreibung gezeigt hat, dass es viele Problematiken und Problemstellen im Entwicklungsprozess gibt - daher liegt die Prozentzahl der erfolgreichen Projekte nach wie vor nur bei um die 70% (je nach Messlatte).
Frei nach dem Paradigma „People Process Technology“ ist der Einsatz erfahrener Personen immer noch der beste Erfolgsgarant.

Über jedes Kapitel dieses Papiers kann man auch ein ganzes Buch füllen, daher bitte ich eventuell fehlenden Tiefgang oder auch die Nichterwähnung vieler weiterer Alternativen zu entschuldigen. Auch so ist dieses Papier länger als ursprünglich geplant geworden.

Leonberg, im Juni 2013

Ralf Grohmann

P.S.: Das Bild auf dem Deckblatt zeigt die Sonne auf den Kanaren mit einem leichten Dunstschleier und entsprechenden Reflexionen. Diese Imperfektionen machen das Bild erst interessant, womit der Bezug zur Softwareentwicklung hergestellt wäre ...

9. Referenzen

- Hauptquelle zum Nachschlagen ist de.wikipedia.org – hier findet man alles, also auch Artikel zu den erwähnten Tools und Methoden, z.B.
  - http://de.wikipedia.org/wiki/Webserver
    http://en.wikipedia.org/wiki/Document-oriented_database
  - http://de.wikipedia.org/wiki/Agile_Softwareentwicklung
  - http://en.wikipedia.org/wiki/Scrum_(development)
  - http://de.wikipedia.org/wiki/DIN_69901
  - http://de.wikipedia.org/wiki/Capability_Maturity_Model
  - ...
- www.topcased.org – die Airbus Entwicklungsumgebung
- www.sourceforge.net – die Heimat vieler Open Source tools
- www.github.com – weitere Open Source Projekte