Interreg

EUROPÄISCHE UNION

Österreich-Tschechische Republik

Europäischer Fonds für regionale Entwicklung

INFORMATIK

Softwareentwicklung und Modellierung

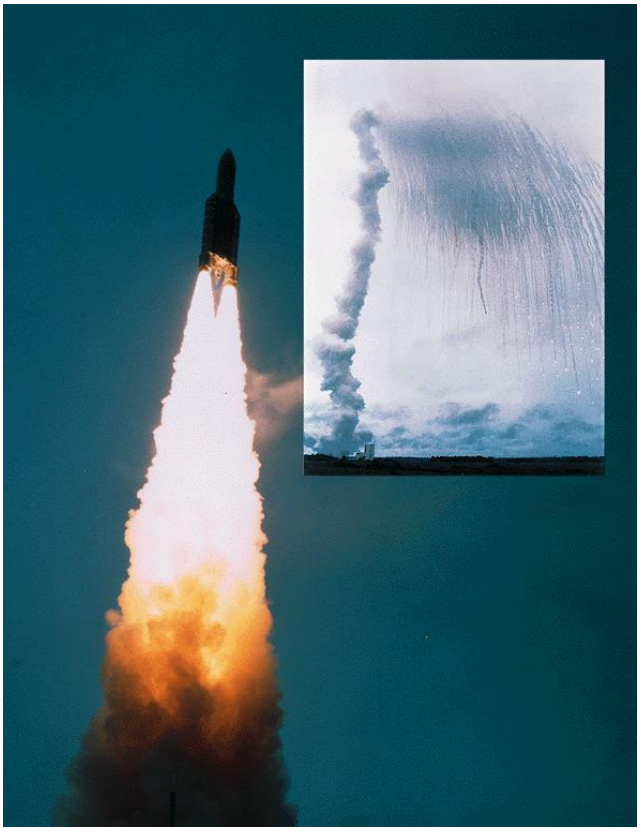
UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

VŠTE

EUROPÄISCHE UNION

Inhalt

1. EINFÜHRUNG IN DIE SOFTWAREENT-WICKLUNG



Softwarefehler können entweder einen Anwendungsabsturz auf einem Telefon oder ein wirklich großes Problem verursachen. Einige der negativen Folgen fehlerhafter Software werden in Richta's Vortrag (2011) aufgezeigt:

Absturz der Ariane-5-Rakete: Aufgrund eines Fehlers erteilte der Hauptrechner den Befehl, die Düsen der Feststoffraketenverstärkern und Motordüsen abzulenken. Dadurch wurde der Raketenkurs dramatisch verändert. Zusätzlich zu den aerodynamischen Kräften brach der obere Teil der Rakete ab. Anschließend wurde das Selbstzerstörungssystem der Rakete aktiviert und die Rakete in eine Wolke aus brennenden Scherben verwandelt. Gesamtsumme: 100 Mio. US $ (einschließlich Cluster-Satelliten, die Teil der Rakete waren, 500 Millionen Dollar).

Der Nordost-Ausfall von 2003 in den USA: Er wurde durch einen nicht registrierten Ausfall des automatischen Fehlerdetektors im Kraftwerk am Niagara River verursacht. Der Blackout verbreitete sich allmählich über das Netzwerk. Der Stromausfall hat insgesamt 21 Kraftwerke stillgelegt. Etwa 50 Millionen Menschen waren betroffen; drei Menschen starben; der Schaden betrug 6 Milliarden US $

Die Kollision des Landemoduls auf dem Mars (1999): Es gab ein Kommunikationsproblem zwischen den Komponenten - der Schnittstellenbenutzer erwartete einen Wert in Kilometern, der vom Anbieter jedoch in Meilen angegeben wurde. Statt 140-150 Kilometer über der Oberfläche geplant, landete er 57 Kilometer über der Oberfläche. In einer solchen Höhe ist die Marsatmosphäre zu schwer für eine Sonde. Der Climate Orbiter brannte wahrscheinlich in der Höhe von 80 Kilometern. (Quelle: Technet.cz)

I.I. Gründe für die Entwicklung von Software Engineering

Zu Beginn der Computerentwicklung in den 1940er und 1950er Jahren war die Computerrechenleistung sehr gering. Im Laufe der Zeit wurde diese Rechenleistung durch geometrische Reihen nach dem etablierten Moore'schen Gesetz entwickelt. Die Softwareentwicklung steckte noch in den Kinderschuhen. Es wurden keine Methoden oder Techniken entwickelt, welche die Entwicklung erleichtern würden. Als sich die Rechenleistung in den 1960er Jahren wieder deutlich weiterentwickelte, kam es zu einer so genannten Softwarekrise.

Die Besonderheiten der Softwarekrise waren unhaltbare Verzögerungen und Preiserhöhungen bei Projekten, schlechte Programmqualität, Wartungs- und Innovationsschwierigkeiten, geringe Effizienz der Programmierer, Entwicklungsineffektivität, schlechte Ergebnisse etc.

Die **Hauptursachen für die Krise** sind wie folgt *(Wikipedia)*:

- Schlechte Kommunikation zwischen den an der Softwareentwicklung beteiligten Personen und zwischen Entwicklern und Kunden.
- Falscher Ansatz. Die jeweilige Software wurde von einem Entwickler entwickelt und genehmigt und ein unzufriedener Kunde wurde als eine Person bezeichnet, die die Angelegenheit nicht versteht.
- Schlechte Planung des Projekts. Die Entwickler hofften, die Frist irgendwie einzuhalten.
- Unrealistische Schätzungen der Entwicklungsdauer, der Kosten und des Umfangs.
- Geringe Arbeitseffizienz. Die Programmierer achteten auf unwichtige Dinge und ignorierten das Wesentliche.
- Unkenntnis der Grundregeln. Z.B. das Gesetz von Brook aus dem Jahr 1975: "Das Hinzufügen einer Forschungskapazität zu einem verzögerten Projekt wird die Verzögerung nur noch verstärken."
- Unterschätzung von Bedrohungen und Risiken. Die Bedrohungen wurden nicht berücksichtigt und verursachten schließlich große Probleme.
- Schlechte Technologieanwendung. Falsche Vorstellung, dass alle Probleme nach der Einführung einer neuen Technologie gelöst werden.

Als Reaktion auf die oben genannten Themen fand 1969 eine Konferenz statt, die Grundlagen des Software-Engineerings vorstellte.

Damals war es wie folgt definiert:

"Software-Engineering ist eine Disziplin, die sich mit der Festlegung und Anwendung von Schlüsselingenieurprinzipien in der Softwareentwicklung beschäftigt, um eine wirtschaftliche Softwareentwicklung zu erreichen, die zuverlässig ist und effizient auf verfügbaren Computergeräten arbeitet."

Software Engineering kann als eine Ingenieurdisziplin bezeichnet werden, die sich mit aktuellen Problemen der Entwicklung komplexer Softwaresysteme beschäftigt. Diese Disziplin beinhaltet mehrere komplexe Denkprozesse. Diese Ingenieurdisziplin verfolgt pragmatische Ansätze. Diese Ansätze beinhalten die Kenntnis der speziellen Anforderungen an das Softwareprodukt, seine Analyse, sein Design, seine Implementierung, seinen Test und seine Installation beim Endanwender. Gleichzeitig beinhalten sie einen betriebswirtschaftlichen Ansatz für das Projektmanagement, der den effektiven Einsatz einzelner Techniken ermöglicht und deren Hauptziel es ist, ein qualitativ hochwertiges Softwareprodukt effektiv zu entwickeln.

1.2. Psychologische Exkursionen

Die Kognitionspsychologie erklärte verschiedene Wege der Problemlösung zwischen Experten und Novizen. Experten sind in der Lage, Teillösungen ähnlicher Situationen in Erinnerung zu rufen und das Vorwissen zur Identifizierung und Definition des Problems anzuwenden (Eysenck und Keane, 2008). Experten befassen sich mit dem Problem, indem sie den Kern der Sache und alle äußeren Umstände untersuchen. Auf der Grundlage dieser Informationen versuchen sie, eine Lösung zu finden. Anfänger hingegen versuchen, eine Lösung zu erraten und anschließend zu beurteilen, ob es möglich ist, den Lösungsvorschlag aus relevanten Fakten zu erarbeiten. Dabei nutzen sie die Rückwärtsshift-Strategie (Nolen Hoeksema at al., 2012). Laut Plháková liegt der größte Unterschied zwischen Novizen und Experten im Umfang der Wissensorganisation (Plháková, 2003). Hochentwickelte selbstregulierende Fähigkeiten (in einem separaten Kapitel untersucht) umfassen Merkmale erfolgreicher Individuen (Experten). Diese Fähigkeiten ermöglichen eine effektive Nutzung von Wissen und Fähigkeiten (Helus & Pavelková, 1992). Tatsächlich ist das Wissen von Laien, Probleme zu verstehen, meist sehr oberflächlich. Auf der anderen Seite nähern sich Experten zunächst dem Kern des Problems und versuchen, es zu identifizieren und zu organisieren. Sie nähern sich der Lösung erst, wenn sie einen realisierbaren Plan haben. Darüber hinaus konzentrieren sich die Experten viel länger auf die Vorbereitung als Laien, finden aber die optimale Lösung des Problems viel schneller (Plháková, 2003). Říčan (2016) sagt, zu Experten: "Sie verfügen nicht nur über mehr Wissen (quantitativer Aspekt), sondern unterscheiden sich auch qualitativ, d.h. im Hinblick auf ein besser

entwickeltes strategisches Verhalten bei der Bewältigung von Problemen (sie unterscheiden sich dadurch in ihren Lernprozessen). Experten haben besser organisiertes fachspezifisches Wissen, so dass sie neue Informationen aus ihrem Fachgebiet schneller erhalten können, da das vorhandene Vorwissen als Integrationsstruktur für neue eingehende Informationen über das assoziative Gedächtnis mit nur minimalem kognitiven Aufwand fungiert."

Im Allgemeinen können wir sagen, dass Software-Engineering Problemlösungsstrategien integriert, zu denen laut Kognitionspsychologie nur Experten in der Lage sind.

2. LEBENSZYKLUS VON INFORMATIONSSYSTEMEN

Wikipedia vergleicht den Lebenszyklus treffend mit einer Brücke. Eine Brücke ist eigentlich ein Mittel, um menschliche Aktivitäten zu erleichtern. Dementsprechend sollten Softwaresysteme den gleichen Zweck erfüllen. Wie Brücken muss auch Software entworfen, genutzt, gewartet und schließlich außer Betrieb genommen werden. In der Bautechnik gibt es einen Lebenszyklus bestimmter Gebäude, der dem von Software sehr ähnlich ist.

Es gibt viele Klassifizierungen von Software-Lebenszyklen. Wir haben uns für den Systems Development Life Cycle (SDLC) des Justizministeriums der USA aus dem Jahr 2003 entschieden.

- **Initiierung** - erstens ist es notwendig, dass das Objekt mit genügend Geld (Kunde) den Bedarf an einer durch ein Softwaresystem implementierten Verbesserung verspürt. Nennen wir es "Absicht".

- **Systemkonzeptentwicklung** - diese Absicht wird auf Machbarkeit und Eignung überprüft. Der Systemumfang, auf dessen Basis die vom Kunden zu genehmigenden Basiskosten berechnet werden, ist lose definiert.

- **Planungsphase** - das Konzept wird so entwickelt, dass es beschreibt, wie das Unternehmen nach der Installation des genehmigten Systems funktionieren soll. Darüber hinaus werden spezifische Projektpläne erstellt und genehmigt.

- **Phase der Anforderungsanalyse** - alle Anforderungen sind detailliert beschrieben, was eine Weiterentwicklung des Systems ermöglicht. Diese Anforderungen betreffen in der Regel Daten, Rechenleistung, Sicherheitssystem, Wartungsanforderungen etc.

- **Designphase** - In dieser Phase werden die physischen Merkmale des Systems entworfen. Die Hauptteile des Systems, seine Ein- und Ausgänge werden definiert. Alles, was eine Eingabe oder die Zustimmung des Benutzers erfordert, muss vom Benutzer aufgezeichnet und überprüft werden.

- **Entwicklungsphase** - alle in den vorangegangenen Phasen festgelegten Spezifikationen werden in die entwickelte Software übernommen. Sie werden anschließend getestet.
- Integrations- und Testphase - einzelne Systemkomponenten werden integriert und systematisch getestet.

- **Implementierungsphase** - das System wird beim Kunden installiert und in Betrieb genommen. Der Kunde muss das System testen und freigeben. Die Phase

endet mit der Inbetriebnahme des Produkts.

- **Betriebs- und Wartungsphase** - Der Betrieb des Systems sollte angemessen überwacht werden. Bei Bedarf werden notwendige Systemmodifikationen vorgenommen. Dieser Prozess geht weiter, bis das System effektiv an die Anforderungen des Kunden angepasst werden kann. Ist der Prozess nicht weiterführbar oder zu teuer, beginnt eine Phase der neuen Softwareplanung.

- **Dispositionsphase** - Außerbetriebnahme des Systems. Besonderes Augenmerk wird auf die genaue Speicherung der vom System verarbeiteten Daten gelegt, damit die Informationen im Falle eines zukünftigen Zugriffs effektiv in ein anderes System übertragen oder gemäß den offiziellen Vorschriften und Richtlinien der Aktenverwaltung archiviert werden können.

Der gesamte Software-Lebenszyklus wird im folgenden Text beschrieben. Einzelne Methoden der Softwareentwicklung werden gezeigt, um sich eng auf die RUP-Methode (Rational Unified Process) zu konzentrieren. Diese Methode umfasst Phasen von der Initiierung bis zur Implementierung. Die RUP-Methode arbeitet viel mit UML, daher werden sechs verschiedene Arten von UML-Diagrammen angezeigt. Geschäftsprozessmodell und Notation, die zu Beginn der Entwicklung neuer Software nützlich sein können, werden diskutiert. Softwaretests und deren Wartung mittels ITIL-Methode werden in einem separaten Kapitel beschrieben.

3. METHODEN DER SOFTWAREENTWICKLUNG

Im Allgemeinen beziehen sie sich auf Verfahren, Regeln und Geräte, die zur Softwareentwicklung führen. Manchmal werden sie auch als Softwareprozess bezeichnet. Einzelne Geräte, die für die Softwareentwicklung konzipiert sind, werden als Framework bezeichnet. Ziel dieser Verfahren ist eine effektive Entwicklung und Wartung der Software.

Durch die Einhaltung dieser Verfahren reduzieren wir das Risiko unerwarteter Probleme und machen den Arbeitsplan einer bestimmten Software klarer. Wenn einzelne Entwickler die gleichen Verfahren befolgen, fördern sie damit die Zusammenarbeit durch die zuvor formulierten Regeln der Programmentwicklung.

Im Laufe der Zeit wurden viele Methoden der Softwareentwicklung entwickelt. Dennoch gibt es bisher keinen detaillierten und klar definierten Rahmen einer Softwareentwicklungsmethode, der als referentiell angesehen werden könnte. Zu diesen Methoden gehören:

3.1. Wasserfallmodell

Das Wasserfallmodell ist ein sequentieller und linearer Prozess, der als sequentieller Fortschritt betrachtet wird, der durch die einzelnen Phasen der Konzeption fließt: Design, Test, Integration, Wartung und alternativ Übergang. Dieses Modell stammt aus den 1970er Jahren. Es kann als ein wesentliches Modell betrachtet werden, auf dem andere Modelle basieren.

Der Wechsel von einer Phase zur anderen sollte nur dann durchgeführt werden, wenn die vorhergehende Phase teilweise oder vollständig verifiziert ist. Leider ist es oft nicht möglich, dieses Modell im wirklichen Leben zu nutzen. So ändert der Kunde beispielsweise die Programmspezifikationen oft erst in der Phase der Programmentwicklung oder bis zu deren Anwendung. Die Hauptprobleme sind wie folgt:

- Eine Beurteilung der Endqualität des Produkts im Rahmen der Softwareentwicklung ist aufgrund von zuvor festgelegten Kriterien für Softwarespezifikationen nicht möglich.
- Das Endprodukt hängt von den anfänglichen Anforderungen an die endgültige Software ab.
- Der Zeitaufwand für die Softwareentwicklung ist zu hoch.

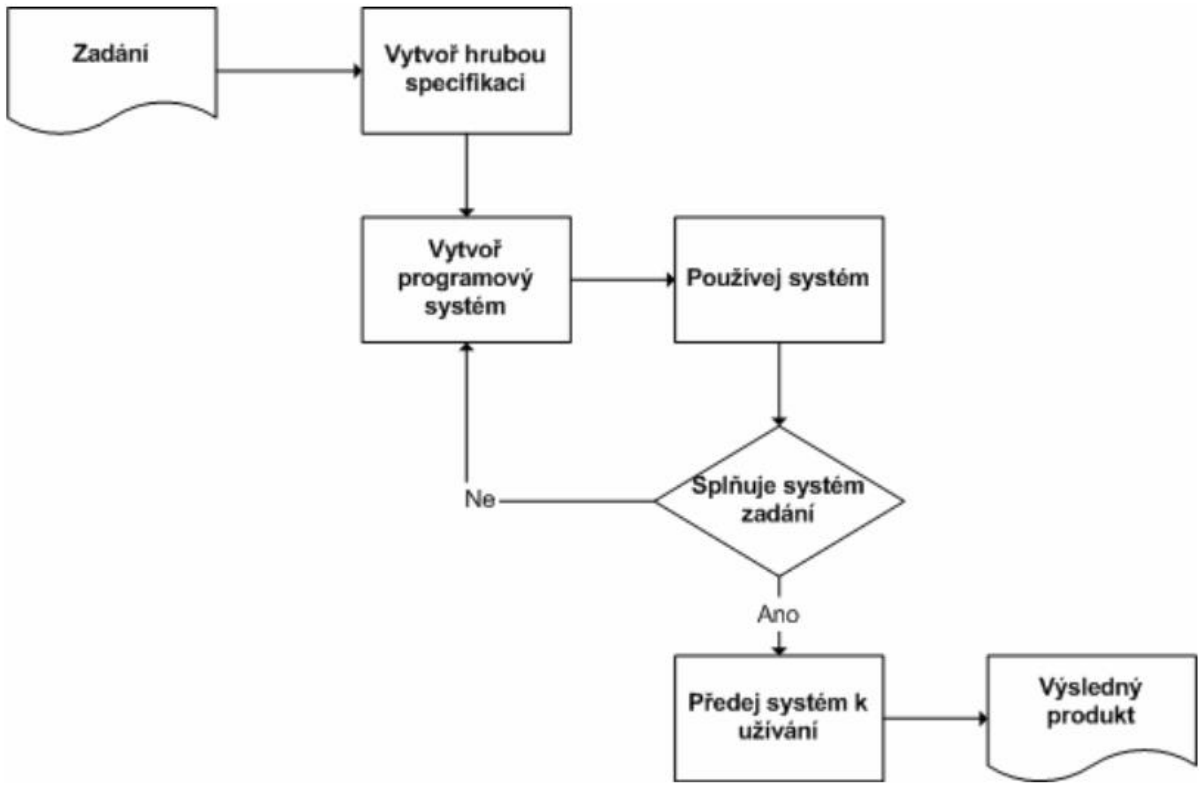
In Bezug auf dieses Modell würde der Versuch, Fehler zu beheben, zur Entwicklung anderer Modelle führen. Diese Prozesse werden von prof. Vondrák (2002) beschrieben:

3.2. Iterative und inkrementelle Entwicklung

Einzelne Schritte werden in wiederholten Zyklen durchgeführt "Iterationen". Jede Iteration nimmt einen kleineren Teil (Inkrement) der Funktionen in die Entwicklungssoftware auf, was zum Soll-Zustand führt. So wird das erforderliche Programm entwickelt und seine Funktionsfähigkeit verbessert. Damit hat der Kunde die Möglichkeit, die Entwicklungssoftware bereits in der frühen Entwicklungsphase auszuprobieren. Auf diese Weise kann er seine Anforderungen an die entwickelte Software besser spezifizieren. So werden Fehler, Mängel oder falsch gestellte Anforderungen an das Programm deutlich früher erkannt. Da der Kunde die Software von Anfang an sieht, kann er sich stärker an der endgültigen Form des Produkts beteiligen. Im Gegensatz zum Wasserfallmodell sind die Unterschiede zwischen den einzelnen Phasen sehr gering. Es ist vielmehr eine Parallele, in der die Iteration einen kleinen Wasserfall darstellt. Der RUP-Prozess, der in einem separaten Kapitel behandelt wird, fällt ebenfalls in diese Kategorie.

Explorative Programmierung

Prof. Vondrák (2002) betrachtet dieses Modell als abschreckendes Beispiel, das noch immer manchmal verwendet wird. Er nennt dieses Modell Exploratory Programming.



3.3. Agile Softwareentwicklung

Die meisten Softwareprozesse stammen aus Großunternehmen. Diese Prozesse erforderten die Einhaltung des geplanten Verfahrens und die obligatorische Erstellung vieler Dokumente. Das ist in großen Unternehmen angemessen, in denen viele Menschen an dem Projekt arbeiten oder in welchen ein hohes Maß an Zuverlässigkeit gefordert ist (z.B. Netzsteuerung). Bei kleinen Projekten war die Zeit, die damit verbracht wurde, alle Verfahren und sonstigen Formalitäten gehorsam zu befolgen, viel länger als die Zeit für Programmierung und Tests.

Als Reaktion darauf entstanden zu Beginn des neuen Jahrtausends agile Methoden oder Techniken. Die Grundprinzipien dieser Techniken wurden im Manifest für Agile Softwareentwicklung formuliert.

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Arbeitsprogramme sind wichtiger als eine umfassende Dokumentation.
- Die Zusammenarbeit mit dem Kunden ist wichtiger als die Vertragsverhandlung.
- Die Reaktion auf Veränderungen ist wichtiger als die Befolgung eines Plans.

Das Grundprinzip dieser Methoden ist es, den Entwicklern die Umgebung und Unterstützung zur Verfügung zu stellen, die sie für die Softwareentwicklung benötigen. Die Entwickler konzentrieren sich auf die Softwareentwicklung, ihr Design. Sie beschäftigen sich nicht zu sehr mit der Dokumentation. Diese Methoden fallen in iterative Methoden. Ihre hohe Priorität liegt in der aktiven Zusammenarbeit mit den Kunden. Der Arbeitsplan wird in der Regel innerhalb der nächsten Iteration erstellt. Aufgrund ihrer "anarchistischen" Natur eignen sich diese Techniken für kleinere Qualitätsentwicklerfirmen. Das zentrale Prinzip der Informationsvermittlung innerhalb eines Entwicklungsteams ist die persönliche Kommunikation.

Derzeit gibt es mehrere agile Methoden wie Extreme Programming und Scrum (Sklenář, 2007).

4. RUP

Der RUP-Prozess (Rational Unified Process) entstand in der zweiten Hälfte der 90er Jahre in Zusammenarbeit mit mehreren Unternehmen unter der Leitung der Rational Company. Dieses Unternehmen wurde 2003 in eine IBM-Abteilung umgewandelt. RUP ist nicht der einzige etablierte Prozess, es ist vielmehr ein iteratives Prozess-Framework, das an die Bedürfnisse von Entwicklerorganisationen und einem Software-Entwicklungsteam angepasst werden sollte. Diese Anpassung ist so implementiert, dass die Entwicklungsteams Prozesselemente auswählen, die ihren Bedürfnissen entsprechen. RUP stammt von UP (Unified Process). RUP gehören derzeit zu den weit verbreiteten Software-Prozessmodellen. Viele Tools unterstützen es.

Ein gemeinsames Wasserfallmodell definiert separate Rollen. Dabei kommt es vor, dass ein Analytiker die Vorstellung des Kunden vom Funktionieren des Systems akzeptiert. Diese Idee wird anschließend in ein Dokument umgesetzt, das dem Programmierer übergeben wird. Wenn es keine effektive Zusammenarbeit zwischen dem Analysten und dem Entwickler gibt, muss sich der Entwickler auf die im Dokument dargelegten Anforderungen konzentrieren. In diesem Fall kann die Interpretation der Anforderungen durch den Entwickler erheblich von der ursprünglichen Idee des Kunden abweichen.

Dieser Prozess basiert auf mehreren Prinzipien. Auch basiert er auf der Idee der softwareiterativen Entwicklung. Dadurch sollte jedes Programm in Rekursionen (Iterationen) entwickelt werden. Die Iteration sollte durch einen ausführbaren Code erzeugt werden.

Žáček (2017) legt die Grundprinzipien von RUP fest:

- Der Versuch, Projektrisiken so schnell wie möglich und regelmäßig wie möglich zu minimieren.
- Um sicherzustellen, dass die Kunden einen Mehrwert erhalten.
- Fokussierung auf ausführbare Software.
- Veränderungen in frühen Phasen des Projekts vornehmen.
- Frühzeitiger Entwurf einer ausführbaren Architektur.
- Wiederverwendung vorhandener Komponenten.
- Enge Zusammenarbeit - alle sind ein Team.
- Die Projektqualität wird von all ihren Proportionen bestimmt, nicht nur von einem Teil (Testing).

Ursprüngliche Anforderungen an die Software sind für die Qualitätsentwicklung des Produktes unerlässlich. RUP erstellt ein **Anforderungsmanagementsystem**. Es verwaltet deren Erfassung, Dokumentation und überwacht Änderungen der Anforderungen. Anschließend wird dieses System von den Entwicklern in der Kommunikation mit den Kunden eingesetzt. Änderungen in der Softwareentwicklung werden auf die gleiche Weise verwaltet.

Viele Teile des Softwareprodukts sind sich ähnlich. Daher ist ihre Neuerfindung Zeitverschwendung. RUP führt die **komponentenbasierte Entwicklung** ein. Sobald wir über die notwendigen Komponenten verfügen, beginnt die Entwicklung des Produkts mit der Integration der entsprechenden Komponenten. Diese Integration kann mit Lego oder dem Aufbau eines Desktop-Computers verglichen werden.

Aufgrund der Komplexität der entwickelten Programme ist die **Softwarevisualisierung** für eine bessere Idee sehr wichtig. RUP arbeitet mit der UML-Sprache.

Die **Überprüfung der Softwarequalität** ist ein sehr wichtiger Aspekt für das Feedback an die Entwickler, um die Bedürfnisse der Kunden dank der funktionierenden Software zu erfüllen. An dieser Überprüfung sollten alle Produktentwickler beteiligt sein. RUP spezifiziert bestimmte Verfahren zur Beurteilung der Softwarequalität.

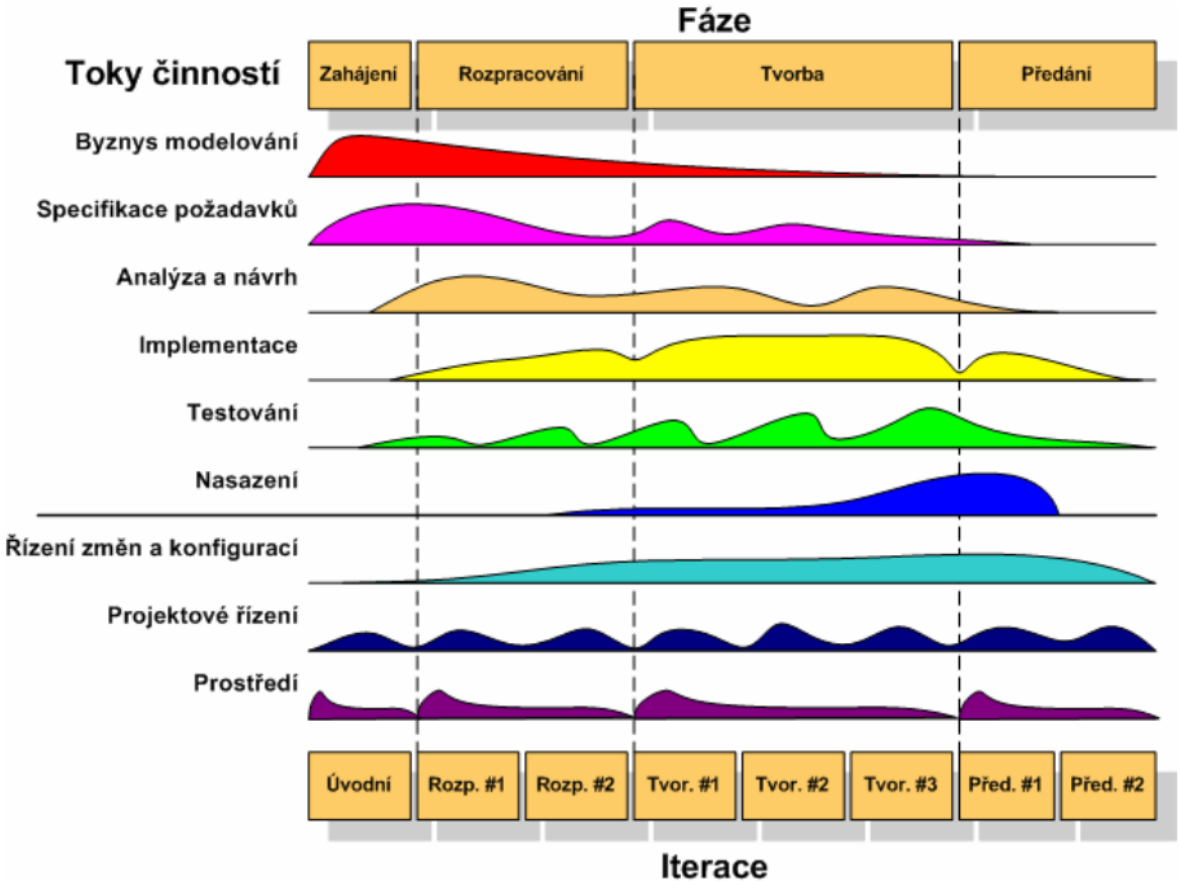
Die Softwareentwicklung von RUP gliedert sich in vier Phasen. RUP-Phasen können als einzelne Projektphasen und deren zeitliche Veränderungen bezeichnet werden. Jede Phase besteht oft aus mehreren Iterationen.

4.1. Schematisches Modell

Das schematische Modell besteht somit aus sechs Ingenieurdisziplinen und drei Hilfsdisziplinen. Zu den ingenieurwissenschaftlichen Disziplinen gehören:

- Geschäftsprozessmodellierung
- Anforderungsspezifikation
- Design und Analyse
- Implementierung
- Prüfung
- Bereitstellung
- Zu den Hilfsdisziplinen gehören:
- Konfigurations- und Änderungsmanagement
- Projektleitung
- Vorbereitung der Projektumgebung

Eine relative Häufigkeit von Aktivitäten einzelner Disziplinen im Zusammenhang mit einzelnen Phasen und Projektiterationen wurde von prof. Vondrák (2002) erarbeitet:



Eine relative Zeit- und Quellverteilung sollte wie folgt aussehen:

	Beginn	Ausarbeitung	Errichtung	Übergang
Quellen	Ca 5%	20%	65%	10%
Zeit	10%	30%	50%	10%

Žáček (2017) schlägt die Ergebnisse der einzelnen Phasen vor:

- Das Ergebnis von Inception ist das Verständnis der wichtigsten Themen, das Ziel des Projekts, die Bewertung der Risiken.
- Der Output von Elaboration ist eine ausführbare und verifizierte Architektur (=Funktionierender Teil der Anwendung).
- Der Output von Construction ist eine Beta-Release-Anwendung - eine relativ stabile, wiederausführbare und nahezu vollständige Anwendung.

- Das Ergebnis von Transition ist ein Produkt, das für den endgültigen Einsatz bereit ist, einschließlich der gesamten Dokumentation und Hardware. Der wesentliche Unterschied besteht auch darin, dass jede Phase wie folgt charakterisiert ist:

  Die einzelnen Phasen umfassen eine unterschiedliche Anzahl von Mitarbeitern und Stellen. Es ist wichtig, dass die Mitarbeiter immer in einem oder mehreren Teams zusammenarbeiten. Bei Bedarf helfen und beraten sie sich gegenseitig.

4.2. Beginn der Tätigkeit

Zu Beginn ist es notwendig, die Projektgröße, den Bedarf, die Zeit- und Kosten der gesamten Projektdurchführung zu beurteilen. Žáček (2017) schlägt 5 Ziele dieser Phase vor:

- Den Prozess verstehen - die Vision vermitteln, die Systemgröße und -grenzen definieren; die Person verstehen, die das System will und wie sie davon profitieren könnte.
- Identifizierung der wichtigsten Systemfunktionalitäten - Identifizierung der kritischsten Use Cases.
- Vorschlag mindestens einer möglichen Lösung (Architektur), (d.h. ob es möglich ist, in der Entwicklung fortzufahren Hinweis: Autor)
- Über Kosten, Projektkonzeption und Risiken informiert zu sein.
- Definition/Modifikation des Prozesses; Werkzeugauswahl und -einstellung.

Definition und Auferlegung von Anforderungen

Im Rahmen der gestellten Anforderungen versuchen wir, maximale Informationen darüber zu erhalten, was die Benutzer von dem neuen System erwarten und welche wesentlichen Funktionen das System erfüllen soll. Zu den Techniken der Datenerhebung gehören:

- Dokumente und Aufzeichnungen
- Interviews mit zukünftigen Anwendern
- Fragebögen
- Beobachtung
- Aufzeichnung der Aktivitäten der Benutzer

Diese Anforderungen lassen sich unterteilen in:

- **Funktionale Anforderungen** Dies sind Anforderungen an die Systemfunktionalität aus dem System; alternativ sollten die Funktionsfähigkeit und Unfähigkeit des Systems definiert werden. Ein Anwendungsfalldiagramm ist ein effektives Werkzeug zur Veranschaulichung der funktionalen Anforderungen.

- **Nichtfunktionale Anforderungen** Dies sind Anforderungen an die Systemeigenschaften als Ganzes. Sie können sich auf Effizienz, Zuverlässigkeit, Sicherheit und Systemkapazität beziehen; alternativ können sie sich auf organisatorische Anforderungen wie die Einhaltung von Standards und die Einhaltung etablierter Verfahren beziehen. Alternativ können sie auch Anforderungen an die gültige Gesetzgebung beinhalten.

Ziele

Das Hauptziel ist es, eine Vision des Projekts zu vermitteln. Diese Vision sollte in Bezug auf den Benutzer definiert werden. Sie sollte jedoch gleichzeitig eine grundlegende technische Sicht auf die angewandten Technologie- oder Architekturkomponenten enthalten. In dieser Vision sollten vor allem die Anforderungen an die zukünftige Software festgelegt werden. Diese Phase findet nur einmal statt. Dennoch kann sie in komplexeren Fällen mehrfach wiederholt werden.

Gleichzeitig sollte in bestimmten Bereichen der Grad der Risiken bewertet werden; diese Risiken stellen sich wie folgt dar:

- Risiken technischer Bereiche (Technologien, Kompatibilität, Kommunikation mit anderen Systemen)
- Finanzielle Risiken - Entwicklungskosten und anschließende Wartung
- Zeitrisiken - je mehr Quellen vorhanden sind, desto kürzer ist wahrscheinlich die eigentliche Entwicklung.

Am Ende dieser Phase ist es notwendig zu beurteilen, ob die Entwicklung fortgesetzt werden kann. Je früher wir einer obskuren Projektentwicklung ein Ende setzen, desto geringer sind die Kosten der Projektentwicklung. Zu diesem Zweck wird der Lifecycle Objective Milestone (LOM) mit folgenden Bewertungskriterien verwendet:

- Interessierte Parteien stimmen über den Projektumfang, die Kosten und den Zeitplan überein.
- Vereinbarung, dass die richtigen Anforderungen an das System gestellt wurden und dass es ein gemeinsames Verständnis dieser Anforderungen gibt.
- Vereinbarung, dass die Kosten- und Terminschätzungen, Prioritäten und Risiken dem Entwicklungsprozess entsprechen.
- Alle Risiken wurden identifiziert und Minderungsstrategien festgelegt.

4.3. Ausarbeitung

Diese Phase beginnt mit der Bildung des Projekts. Ziel dieser Phase ist es, die Systemarchitektur zu etablieren, um in der nächsten Phase einen Großteil der gesunden Funktionalitäten zu produzieren. Diese Architektur basiert auf Erkenntnissen aus der vorherigen Phase. Wesentliche Fragen des Architekturdesigns der entwickelten Software sollen hier behandelt werden. Die Funktionalität des Systems muss nicht perfekt sein, dennoch sollte der Großteil der wesentlichen Funktionalitäten bereits in das System integriert sein. Das Forschungsteam sollte sich darüber im Klaren sein, dass sich das Architekturdesign in dieser Phase stabilisieren muss.

An dieser Stelle wird das entscheidende Thema analysiert und die Projektarchitektur erhält ihre Grundform. Mit dem Architekturvorschlag bemühen wir uns die Risiken im Zusammenhang mit der Softwareentwicklung zu minimieren. Diese Risiken betreffen im Wesentlichen:

- Zeit und Finanzen
- Genauigkeit der Architektur
- Prozesse und Technologien
- Ziel der Softwareentwicklung

Eine Reihe kleinerer Iterationen können den Grad der oben genannten Risiken an dieser Stelle mindern und helfen, das Hauptziel der Entwicklung zu überwachen. Wenn wir das System mit ähnlichen Technologien wie bisher aufbauen, gibt es offensichtlich ein geringeres Maß an potenziellen Risiken. Daher sind wir in der Lage, die Ziele dieser Phase innerhalb der Iteration zu erreichen. Andererseits, wenn wir mit der verwendeten Technologie nicht vertraut sind oder das Projekt komplexer ist (z.B. strengere Sicherheitsanforderungen), benötigen wir zwei oder mehr Iterationen. Für den Fall, dass in dieser Phase eine wesentlichere Änderung der Architektur stattfindet, sollte eine weitere Iteration hinzugefügt werden, die die notwendige Funktionalität und Stabilität der modifizierten Architektur überprüft. Hier gilt der Grundsatz "je später das Fundament der Konstruktion geändert wird, desto teurer wird der gesamte Umbau".

Um das Ziel der Softwareentwicklung zu überprüfen, sollte eine Beta-Version der Benutzeroberfläche erstellt werden, auf der die wichtigsten Systemfunktionalitäten getestet werden.

Am Ende dieser Phase muss die Systemarchitektur stabilisiert werden. Die ausführbare Architektur muss die Systemarchitektur überprüfen und dient gleichzeitig als Mittel zur Überprüfung kritischer Systemfunktionalitäten. Diese kritischen Funktionalitäten müssen auch modelliert werden, z.B. mittels UML-Sequenzdiagramm. Wir versuchen auch, potenzielle Probleme und Risiken zu identifizieren. Anschließend sollten wir ausgewählte Objekte zu Paketen gruppieren, die Sichtbarkeitsregeln und zukünftige Produktkonfiguration betreffen. Wir sollten auch eine Vorstellung davon haben, wie mit den Daten in der Da-

umgegangen wird, weshalb wir die Komponenten regelmäßig integrieren sollten. Das bedeutet, dass die Reihenfolge und Art und Weise, in der die ausgewählten Komponenten integriert werden, festgelegt wird.

Schließlich stellt das Testen kritischer Szenarien einen sehr wichtigen Teil dieser Phase dar. Das Testen kritischer Szenarien kann unseren Fortschritt in der Erarbeitungsphase belegen. Darüber hinaus sollten kritische Szenarien unter Aspekten wie hohe Systembelastung, auf ausreichende Systemarchitekturleistung oder Zusammenarbeit mit externen Systemen getestet werden.

An dieser Stelle testen wir die Systemvorgaben mittels Lifecycle Architecture Milestone (LCA), während es sich bei den Bewertungskriterien um LCA handelt:

- Die Produktvision und die Anforderungen sind stabil.
- Die Architektur ist stabil.
- Die wichtigsten Ansätze und Verfahren werden getestet und verifiziert, so dass sie sich als anwendbar erwiesen haben.
- Tests und Evaluierungen von ausführbaren Prototypen haben wichtige Projektrisikoelemente gezeigt, die erfolgreich gelöst wurden.
- Die Iterationspläne für die folgende Phase werden erstellt, damit die Arbeiten fortgesetzt werden können.
- Die Iterationspläne werden durch glaubwürdige Schätzungen unterstützt.
- Alle Teilnehmer sind sich einig, dass die aktuelle Vision erfüllt werden kann, wenn der aktuelle Plan im Kontext der aktuellen Architektur ausgeführt wird.
- Istkosten im Vergleich zu Plankosten sind akzeptabel.
- Das Projekt kann abgebrochen oder anderweitig überdacht werden, wenn es den Meilenstein nicht erreicht.

4.4. Konstruktion

Das Hauptziel dieser Phase ist es, ein glaubwürdig funktionierendes Softwaresystem aufzubauen und gleichzeitig die Kosten zu minimieren.

Diese Phase konzentriert sich auf die Entwicklung von Komponenten und anderen Systemelementen. Diese Phase ist weiterhin dadurch gekennzeichnet, dass sie eine Verschlüsselung schafft, die sehr zeitaufwendig und menschlich ist (es erfordert eine Menge Programmierer und Tester). Der Rest der Funktionalitäten wird hier entworfen, d.h. die restlichen Hauptfunktionen (Kundenwünsche) und die meisten anderen. Wenn ein Eingriff in die Systemarchitektur behandelt werden soll, ist es wahrscheinlich, dass die vorherige Phase falsch abgeschlossen ist.

Wesentliche Voraussetzungen für den erfolgreichen Abschluss dieser Phase sind: Architekturkompaktheit, parallele Entwicklung einzelner Teams, Konfigurations- und Änderungsmanagement sowie automatisiertes Testen.

Größere Projekte können mehrere Iterationen durchführen (üblicherweise 2 - 4). Diese Phase führt in der Regel die meisten Iterationen durch. Die Iterationsplanung entspricht der Anzahl und Art der noch nicht implementierten Funktionalitäten. Jede Iteration befasst sich mit einem separaten Projektsegment. Im Idealfall entsteht die Architektur, die weiterentwickelt, genutzt oder wiederverwendet wird, aus früheren Phasen. Das System wird integriert und getestet. Im Hinblick auf die effektive Organisation ist ein Team für die Architektur verantwortlich, während andere Teams parallel an der Entwicklung des Subsystems arbeiten. Diese "parallelen Teams" kommunizieren hauptsächlich mit dem Team, das für die Softwarearchitektur verantwortlich ist.

Dank der iterativen Entwicklung werden viele Dateien und deren Versionen entwickelt, die anschließend getestet und integriert werden. Aus diesem Grund muss ein Konfigurations- und Änderungsmanagement eingeführt werden. Dieses Management registriert individuelle Konfigurationen und Änderungen. Funktioniert ein solches System, können sich die Entwickler ganz auf die Weiterentwicklung konzentrieren und so ihre Effektivität steigern. Am Ende dieser Phase muss eine Beta-Version dieses Programms entwickelt werden, die Hilfe bei der Anwendung, Installationsanweisungen, Benutzerhandbücher und Tutorials enthält. Auf diese Weise können zukünftige Anwender die Software ausprobieren und uns ein konstruktives Feedback geben.

Am Ende dieser Phase sollten wir uns mehrere Fragen gemäß dem IOC-Meilenstein (Initial Operational Capability) stellen.

- Die Beta-Version ist bereit für die Freigabe durch die ersten Benutzer (Tester).
- Die Benutzer sind vorbereitet und können unsere Beta-Version nutzen.
- Istkosten versus Plankosten sind akzeptabel.

4.5. Übergang

Diese Phase zielt auf den Übergang des Systems von der Entwicklung zur regelmäßigen Nutzung und die Behebung der wichtigsten Mängel ab (sie betreffen in der Regel die Leistung, Benutzerfreundlichkeit und Funktionalität). Zu diesen Phasenaktivitäten gehören:

- Schulung von Administratoren und Endanwendern
- Systemtests zur Überprüfung der Erfüllung der Erwartungen der Endanwender
- Erstellung von Marketingmaterialien
- Umgebung und Datenaufbereitung, z.B. Datenmigration vom alten zum neuen System

- Das Produkt wird in Bezug auf die zu Beginn des Projekts festgelegten qualitativen Rahmenbedingungen genauestens überwacht.
- Interne Projektbewertung, Lernen aus Fehlern und Problemen aus der Projektarbeit.

Englisches Wikipedia vom 24.6.18 weiters erwähnt: "Das System durchläuft auch eine Evaluierungsphase; jeder Entwickler, der keine sinnvolle Arbeit verrichtet, wird ersetzt oder entfernt.

Auch diese Phase sollte mit einem Meilenstein abgeschlossen werden. PRM (Product Release Milestone) Kriterien sind:

- Die Benutzer sind zufrieden.
- Die Endkosten im Vergleich zu den geplanten Kosten sind akzeptabel. Wenn die Kosten überstiegen wurden, was hätte man tun müssen, um das Problem zu umgehen?

5.GESCHÄFTSPROZESSMODELL UND NOTATION

Ziel war es, ein grafisches Werkzeug zu entwickeln, das für alle Geschäftsanwender (von Analysten über Entwickler bis hin zu Geschäftsprozessverantwortlichen) verständlich ist, was eine effektivere Kommunikation zwischen Entwickler und Kunde ermöglicht. Geschäftsprozessmodell und Notation (nachfolgend BPMN genannt) besteht aus einer Reihe einfacher grafischer Elemente, aus denen ein Geschäftsprozessmodell modelliert werden kann. BPMN wird seit 2005 entwickelt. Die Version 2.0. ist im Jahr 2011 entstanden. Das Prozessablaufdiagramm basiert auf einem Flussdiagramm, das dem UML-Aktivitätsdiagramm sehr ähnlich ist, das im weiteren Text erläutert wird.

Flow-Objekte

Es ist eng mit dem Informationsfluss im Prozess verbunden. Flow-Objekte sind die wichtigsten grafischen Elemente. Sie können in drei Gruppen eingeteilt werden - Events, Activities und Gateways.

Events

Ereignisse werden durch einen Kreis gekennzeichnet, in dem ein Symbol angezeigt werden kann. Es markiert ein Ereignis, das den Prozessablauf direkt steuert. Ereignisse werden in Startereignisse unterteilt, die den Prozess auslösen und mit einem Kreis mit schmalem Rand - manchmal mit grüner Farbe - markiert sind.

Darüber hinaus gibt es Endereignisse, die das Ende des Prozesses oder das Ergebnis der Aktivität anzeigen. Diese sind mit einem fetten Rand oder einem fetten Symbol im Kreis markiert - manchmal mit roter Farbe. Einige Quellen erwähnen auch Intermediate Event, das mit einem doppelten Rand markiert ist.

Aktivität

Aktivitäten werden durch ein abgerundetes Rechteck markiert. Dieses Element beschreibt Arbeit oder Tätigkeit. Die Aktivität kann entweder atomar (d.h. Aufgabe) sein, was als eine einzige Einheit betrachtet wird und immer als Ganzes ausgeführt werden muss.

Alternativ kann es auch auf einen einzelnen Prozess heruntergebrochen werden, der als Subprozess bezeichnet wird. Diese Art von Aktivität wird in Prozessen eingesetzt, die wir auf einer bestimmten Ebene nicht offenbaren wollen. Teilprozessaktivitäten werden durch ein Pluszeichen in der unteren Zeile des Rechtecks mit abgerundeten Ecken gekennzeichnet.

Gateway

Gateways sind mit einer quadratischen oder diamantförmigen Form gekennzeichnet. Es bestimmt die Verzahnung und Zusammenführung von Prozessabläufen, z.B. bei Entscheidungen oder Parallelverarbeitung.

Objekte verbinden

Sie verbinden Flussobjekte oder Artefakte. Durch die Verbindung entsteht eine neue Struktur (Framework) des Geschäftsprozesses. Die verbindenden Objekte werden in drei Grundtypen unterteilt:

- Sequence Flow - ist mit einer durchgezogenen Linie und einer Pfeilspitze gekennzeichnet. Es zeigt die Reihenfolge, in der die einzelnen Flussdiagrammeinheiten ausgeführt werden sollen.

- Nachrichtenfluss - ist mit einer gestrichelten Linie und offener Pfeilspitze gekennzeichnet. Es zeigt den Nachrichtenfluss über die Unternehmensgrenzen des Prozesses hinweg an.

- Assoziation - ist mit einer gestrichelten Linie gekennzeichnet. Es ordnet Objekten zusätzlichen Informationen zu. Es wird auch für die Anzeige von Ein- und Ausgängen verwendet.

Schwimmstrecken

Sie zeigen die Teilnehmer des Prozesses an und beziehen sich auf die Organisation und Kategorisierung der Aktivitäten im Prozess. Sie bestehen aus zwei Arten.

- Pool - repräsentiert die Teilnehmer des Prozesses, oder alternativ trennt er verschiedene Organisationen. Es kann mehr Spuren enthalten. Ein Pool enthält einen separaten Prozess.

- Lane - befindet sich unter dem Pool. Es wird für die Organisation und Kategorisierung von Aktivitäten verwendet.

Artefakte

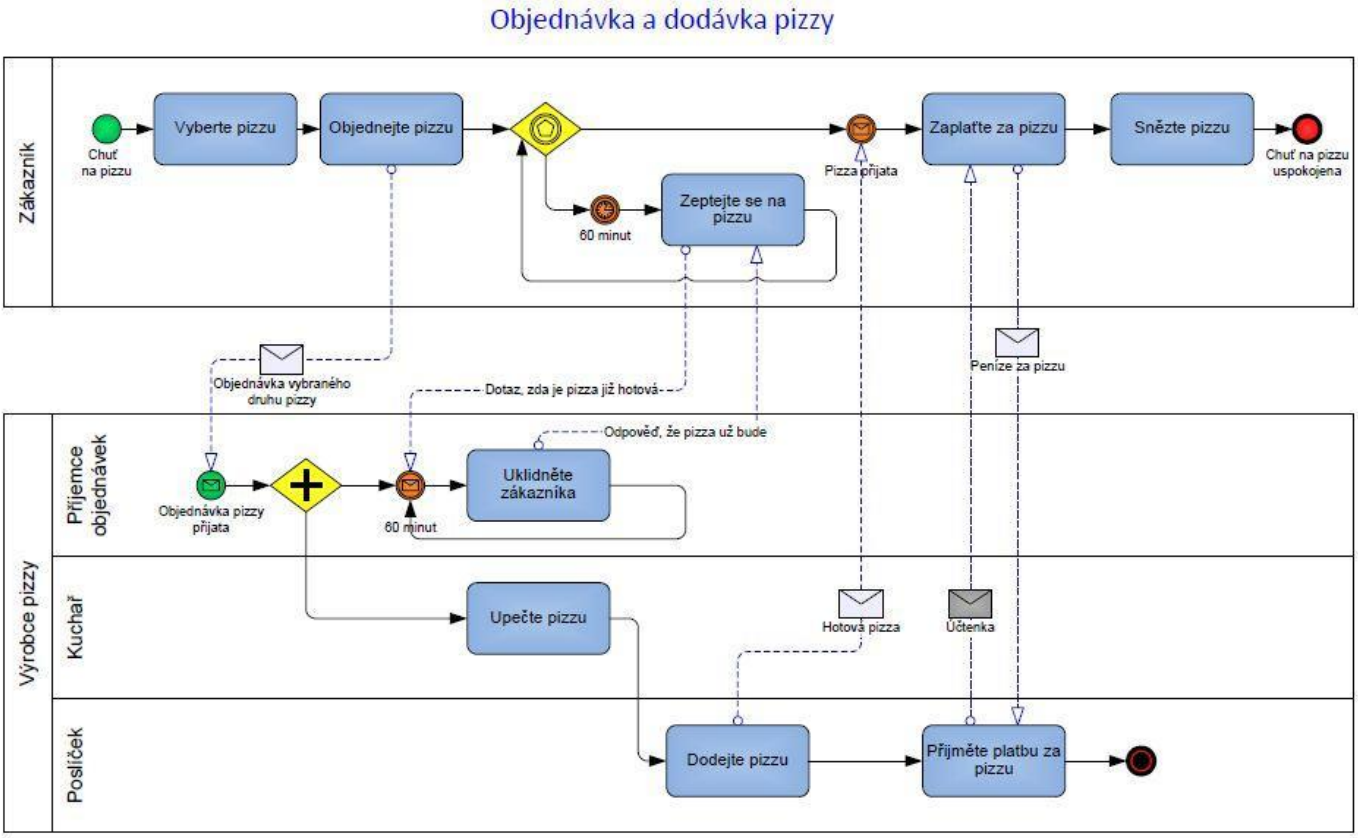
Sie bringen einige neue Informationen in den Prozess ein. Sie sind unabhängig von Bewegungen.

- Datenobjekt - ist mit einem Rechteck in der Biegeecke (ein Blatt Papier) markiert. Es zeigt die in einer Aktivität produzierten Daten an, oder es sagt uns, welche Daten für eine Aktivität benötigt werden.

- Gruppe - ist mit einem abgerundeten Rechteck und gestrichelten Linien markiert. Es wird verwendet, um verschiedene Objekte zu gruppieren.

- Annotation - vermittelt dem Leser einen klaren und verständlichen Eindruck.

Wir stellen ein Modell der Bestellung und Lieferung von Pizza als Beispiel zur Verfügung.



6.UML

Tatsächlich sind viele Softwaresysteme außergewöhnlich komplex. So liegt ihre Entwicklung mit strengen Anforderungen in den Händen und in der Phantasie der Entwickler. Wenn an einem Projekt mehrere Menschen beteiligt sind, muss das zukünftige Softwaresystem genau definiert - modelliert - werden. Es gibt viele Ansätze für eine solche Modellierung. Alle haben ihre Vor- und Nachteile. Um eine effektive Zusammenarbeit zu fördern, ist es wichtig, dass jeder das Modellierungssystem und seine Terminologie kennt, verwendet und versteht. Aus diesem Grund wurde die UML (Unified Modeling Language) entwickelt. Die Standard-UML wird von der Object Management Group (OMG) definiert.

UML bezieht sich auf eine einheitliche Sprache zum Zeichnen von Diagrammen. UML ermöglicht Spezifikationen (Struktur und Modell), Visualisierung (Flussdiagramme), Konstruktion (Softwareentwicklungsmethoden) und Dokumentation von Software-Systemartefakten. Es gibt eine große Anzahl von Diagrammen. Jedes dieser Diagramme wird für ein anderes Ziel und in verschiedenen Projektphasen verwendet. UML bietet insgesamt 13 Arten von Flussdiagrammen. In diesem Text werden die Diagramme in Strukturdiagramme und Klassendiagramme unterteilt.

Strukturdiagramme

Strukturdiagramme beschreiben die Projektstruktur. Sie werden oft verwendet, um die Architektur der entwickelten Software zu beschreiben. Wir werden uns mit mehreren ausgewählten Diagrammen dieser Kategorie befassen.

Klassendiagramm

Dieses Diagramm zeigt die Projektklasseneinteilungen, ihre Beziehungen, Operationen und Methoden oder Methoden der Methoden. Diese Klassen sind in einem rechteckigen, vertikal in 4 Teile gegliederten Rechteck geschrieben. Dieses Diagramm ist der Grundstein der objektorientierten Modellierung. Es wird auch für die Datenmodellierung verwendet. Darüber hinaus stellt das Diagramm eine statische Struktur des Systems dar und hängt teilweise von einer bestimmten Programmiersprache ab. Ziel dieser Aktivität ist es, die Art und Weise vorzuschlagen, wie das Produkt in der Implementierungsphase implementiert wird.

Beispiele für Klassendiagramme finden Sie unten.

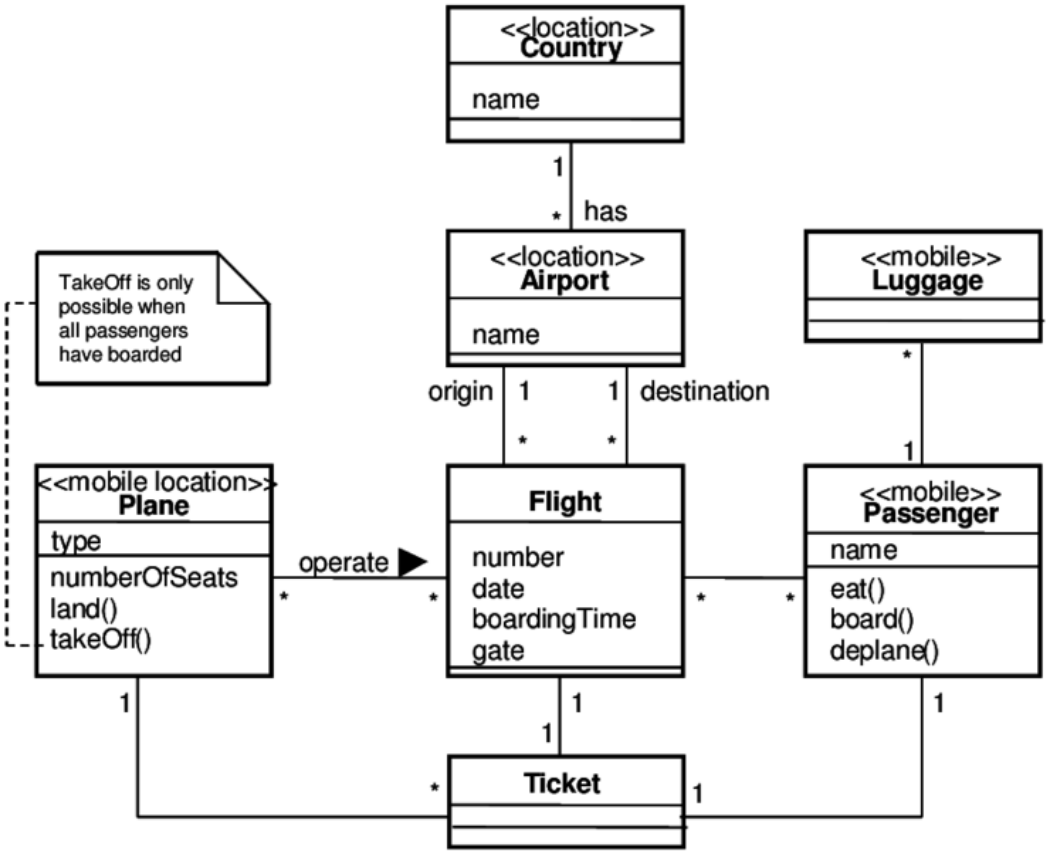


Abb.: Quelle - Andrade et al, Recent Trends in Algebraic Development Techniques—16th International Workshop, WADT 2002, Frauenchiemsee, Germany. Vol. 2755 of LNCS.

Komponentendiagramm

Dieses Diagramm arbeitet mit den im System verwendeten Komponenten und beschreibt, wie die Komponenten zu größeren Komponenten von Softwaresystemen verbunden werden. Es wird verwendet, um die Struktur verschiedener komplexer Systeme zu beschreiben. Das Diagramm wird hauptsächlich für den Entwurf von Software nach Komponenten verwendet. Es hat auch eine höhere Abstraktionsebene als Klassendiagramme. Eine oder mehrere Klassen sind innerhalb einer einzigen Komponente implementiert.

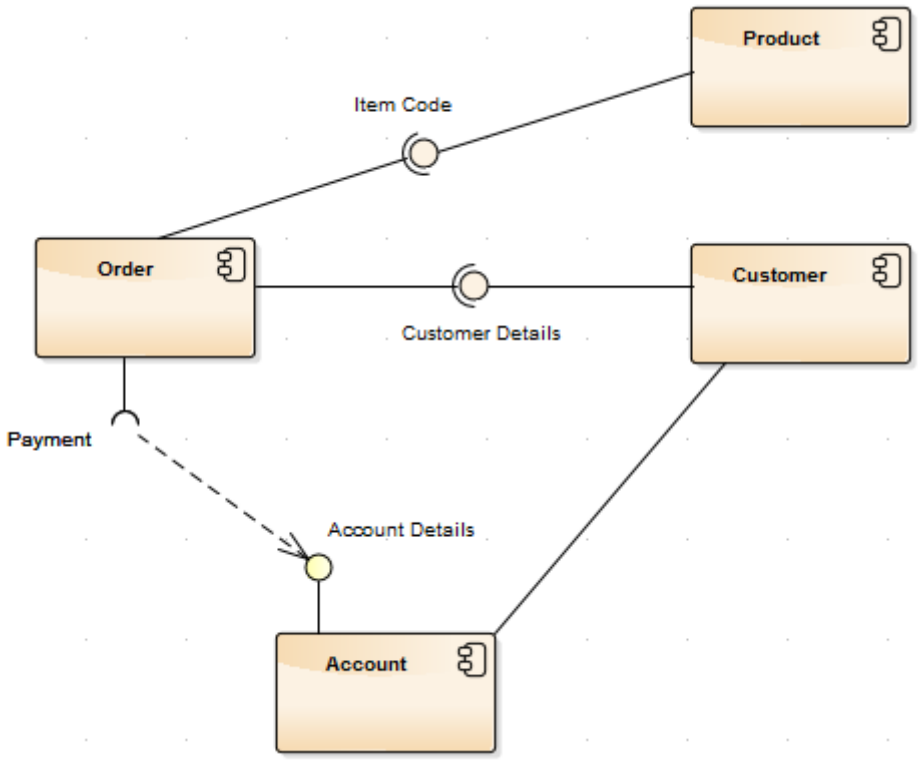
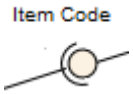


Abb.: Quelle - http://sparxsystems.com/enterprise_architect_user_guide/13.0/model_domains/componentdiagram.html

Das Bild zeigt insgesamt 4 Komponenten.



Montagestecker verbinden die Auftragsschnittstelle mit dem Produkt und dem Kunden. In diesem Fall benötigen die Komponenten ID-Produktinformationen und Angaben zum Kunden.

Eine einfache Linie zwischen Kundenkomponente und Kundenkonto veranschaulicht die Beziehung zwischen den Komponenten.

Objektdiagramm

Das Objektdiagramm stellt Objekte (Klasseninstanzen) und deren Beziehungen (Links - Assoziationsinstanzen) in einem Moment dar. Dieses Diagramm sieht aus wie ein Klassendiagramm und kann tatsächlich als sein spezifisches Beispiel betrachtet werden. Sie wird hauptsächlich verwendet, wenn wir Beispiele für Datenstrukturen in einem bestimmten Moment zeigen wollen.

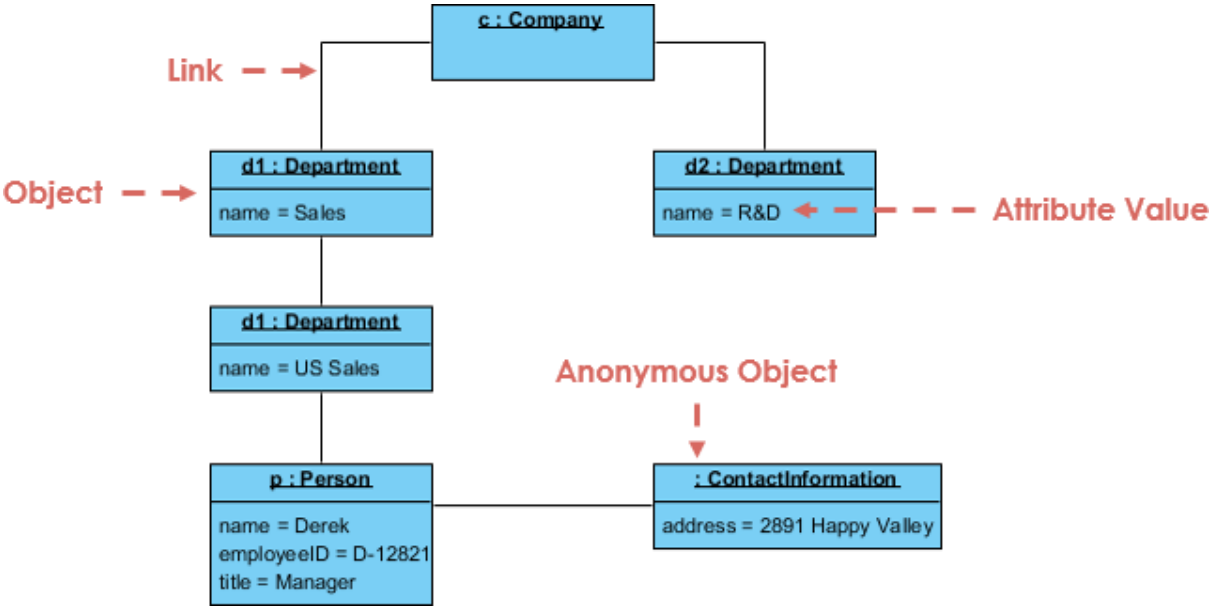


Abb.: Quelle - https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/

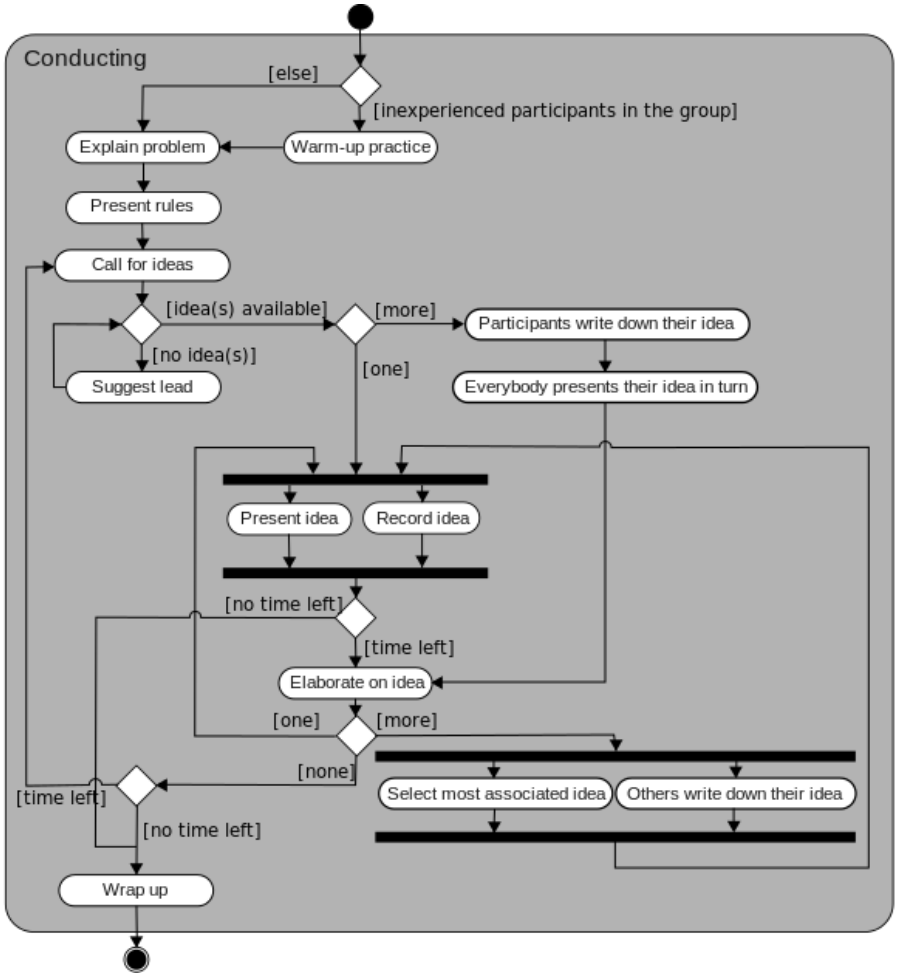
7. VERHALTENSDIAGRAMME

Diese Diagramme beschreiben einzelne Prozesse durch Aktivitäten, die ihre Zustände und Übergänge darstellen. In Anbetracht der Tatsache, dass Aktivitätsdiagramme das Verhalten des Systems beschreiben, werden Verhaltensdiagramme verwendet, um Funktionalitäten des Softwaresystems zu beschreiben.

Aktivitätsdiagramm

Dieses Diagramm ist für die Modellierung von Rechen- und Organisationsprozessen (z.B. Arbeitsprozess) konzipiert und kann zur Modellierung von Datenflüssen verwendet werden. Die Diagrammmodelle verarbeiten Prozesse mittels Aktivitäten, die ihre Zustände darstellen, und Übergängen zwischen einzelnen Zuständen. Dadurch ist es eines der Diagramme, das das Verhalten des Systems erkennen kann.

- Abgerundete Rechtecke stellen Aktionen dar.
- Diamanten stellen Entscheidungen dar.
- Balken stellen einzelne Aktivitäten dar.
- Der schwarze Kreis stellt den Anfang (Anfangsknoten) dar.
- Der umlaufende schwarze Kreis stellt das Ende (Endknoten) dar.



Sequenzdiagramme

Sequenzdiagramme zeigen Objektinteraktionen, die an einer bestimmten Systemfunktionalität beteiligt sind. Neben Objekt und Klassen zeigt es die Abfolge der Nachrichten an, die notwendig sind, um die jeweilige Funktionalität zwischen den einzelnen Objekten in Bezug auf ihre Zeitreihenfolge zu erreichen. Diese Diagramme werden häufig bei der Implementierung von Anwendungsfällen zur Logik des Entwicklungssystems verwendet. Ziel dieser Aktivität ist es, die Art und Weise vorzuschlagen, wie das Produkt in der Umsetzungsphase umgesetzt wird.

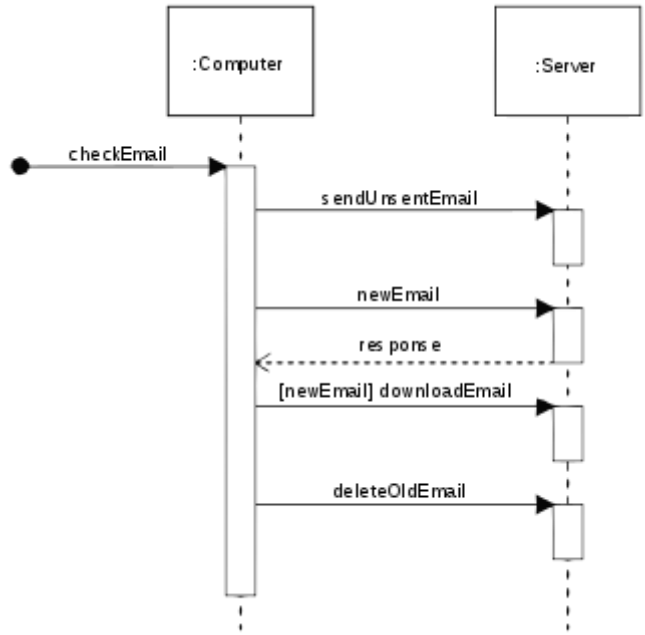


Abb.: Quelle - https://en.wikipedia.org/wiki/Sequence_diagram#/media/File:CheckEmail.svg

Die Zeitlinie im Diagramm ist vertikal (Zeitflüsse von oben nach unten). Die Platzierung der Objekte erfolgt horizontal. Unser Bild zeigt 2 Objekte (Server und Computer); die gestrichelte vertikale Linie wird auch als Lebenslinie bezeichnet. Unser Fall bezieht sich auf Objekte, die vor der Zeichnung des Diagramms existierten und auch nachträglich existieren werden. Rechtecke auf der Lebenslinie veranschaulichen eine Aktivität. Horizontale Pfeile stellen Meldungen dar. Die Pfeilrichtung zeigt den Absender und den Empfänger an. Die volle Zeile zeigt die obligatorische Nachricht, während die gestrichelte Linie eine nicht obligatorische darstellt.

Anwendungsfall-Diagramm

Es stellt in der Regel die Beziehung zwischen dem Kunden und dem System dar, wo es verschiedene Anwendungsfälle in der Kunden-System-Beziehung demonstriert, die oft mündlich durch Szenarien beschrieben wird. Um Szenarien auf Anwendungsfalldiagramme abzubilden, werden Anwendungsfall-Implementierungen angewendet. Hier soll der Algorithmus der Transkription von Szenarioeinträgen in das Diagramm behandelt werden.

So kann beispielsweise BPMN der erste Impuls für die Erstellung dieses Diagramms sein. Das Ergebnis ist eine Liste von Aktivitäten, die von der Entwicklungssoftware anstelle von z.B. Personen durchgeführt werden können. Diagrammfiguren werden als Akteure bezeichnet. Akteure sind mit relevanten Anwendungsfällen verknüpft. Die einfache Linie wird als Assoziation bezeichnet. Die horizontale Richtung zeigt weiter die Struktur der Akteure. Der untere Akteur erbt die Eigenschaften von den Akteuren über ihm. Diese Art von

Beziehung wird als Generalisierung bezeichnet. Die Include-Beziehung wird implementiert, wenn der Anwendungsfall, aus dem sich diese Beziehung entwickelt, implementiert wird.

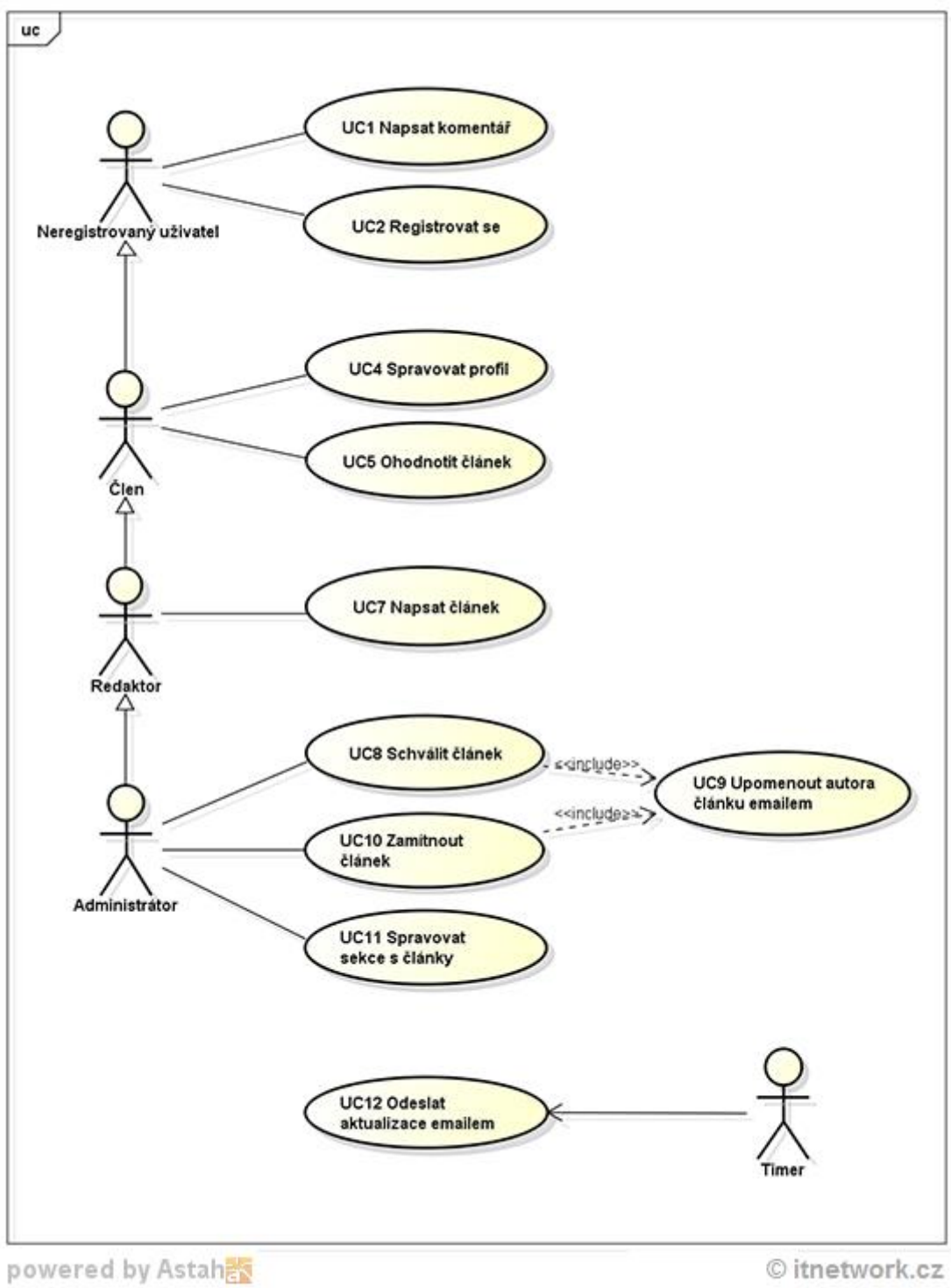


Abb.: Quelle - https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram
Softwaretest und -installation

Es gibt viele Methoden des Softwaretestens, während sich jede Testmethode in der Regel auf einen anderen Testbereich konzentriert. Die einzelnen Methoden lassen sich in funktionale und nicht-funktionale Tests unterteilen. Der Funktionstest konzentriert sich auf die Erfüllung aller Anforderungen der Anwender. Die Ziele der Tests sind in der Norm ISO/IEC 12207 definiert oder können aus den Ergebnissen, die mit Hilfe von FURPS+ erzielt wurden, gewonnen werden. Jede Anforderung wird mit einer Prüfung geliefert.

Informelle Testfälle hingegen testen nicht direkt die Systemfunktionalität, sondern basieren auf normal erwarteten Operationen, die das System normalerweise durchführen kann. Diese sind wie folgt:

- Sicherheitstests - testet die Systemsicherheit
- Lasttests - Tests, bei denen ein System gefordert wird und dessen Reaktion gemessen wird.
- Usability-Tests - messen die Bedingungen, unter denen das System eingesetzt werden kann. Zum Beispiel die Verbindung der Internetgeschwindigkeit.
- Dokumentationsprüfung - Die Benutzerdokumentation wird auf ihre Verständlichkeit und Konsistenz geprüft. In der Entwicklungsdokumentation wird die Deckungshöhe getestet.

Auch Softwaresysteme werden hinsichtlich ihrer Verifikation und Validierung getestet.

- Die **Verifizierung** beantwortet die Frage, ob das entwickelte Produkt alle technischen Anforderungen erfüllt.
- Die **Validierung** hingegen beantwortet die Frage, ob die Software dem Verwendungszweck entspricht.

Wenn ein Kunde zum Beispiel ein E-Shop-System wünscht, und unser entwickeltes System viele unterhaltsame Funktionen hat, diese aber versteckt sind, so dass der Kunde sie normalerweise nicht findet, ist die Folge, dass er das Produkt nicht kauft.

Manchmal sprechen wir über Black Box und White Box Tests.

- **Black Box Testing** nähert sich dem entwickelten Softwaresystem als Black Box. Es geht also nicht um das Innere. Es untersucht die Funktionalität nur aus der Sicht des Endverbrauchers. In der Regel wird geprüft, ob die vorgegebenen Eingänge mit den erforderlichen Ausgängen übereinstimmen. Es versucht auch, ungewöhnliche oder unerwartete Eingaben zu finden, die möglicherweise nicht angemessen reagieren.

- Der **White Box Test** hingegen ist mit der Programmstruktur vertraut. Es wird geprüft, ob einzelne Programmfunktionen ordnungsgemäß funktionieren. Es versucht, jede Funktion einzeln zu testen.

Vondrák (2002) empfiehlt folgende Aktivitäten zur Bereitstellung des Softwaresystems für den Benutzer:

- Entwicklung des Endprodukts oder seiner Versionen
- Fertigstellung des Softwaresystems
- Vertrieb des Softwaresystems
- Installation des Softwaresystems beim Benutzer
- Assistenzdienste für Benutzer
- Planungsmanagement für Beta-Tests
- Migration bestehender Daten- und Softwareprodukte

8. MESSUNG DER QUALITÄT VON SOFTWARESYSTEMEN

Wenn es uns gelingt, die Anforderungen der Anwender/Kunden am Ende des gesamten Prozesses zu erfüllen, ist es wahrscheinlich, dass eine hochwertige Software entwickelt wurde. Dennoch ist dieser Indikator sehr subjektiv. Tatsächlich hat jeder Benutzer/Kunde unterschiedliche Erwartungen. Daher ist es notwendig, objektivere Kriterien für die Bewertung von Qualität und Nutzen von Softwaresystemen festzulegen.

Diese Kriterien werden als Metriken bezeichnet. (Es ist ein etwas unglücklicher Ausdruck da es den gleichen Namen hat wie die Verallgemeinerung des physikalischen Abstandes in der Mathematik)

Žáček (2017) definiert den Begriff der Metrik wie folgt:

Eine Metrik kann als solche definiert werden: "Eine Kennzahl ist ein klar definierter Finanzindikator oder nichtfinanzieller Indikator oder Bewertungskriterium, das zur Beurteilung des Effizienzgrades im jeweiligen Bereich des Geschäftsergebnisses und seiner effektiven Unterstützung durch IS/ICT verwendet wird." Die Gruppe von Metriken, die für einen bestimmten Zweck (d.h. bezogen auf einen bestimmten Bereich, Prozess oder ein bestimmtes Projekt) zugeordnet sind, werden als "Portfoliokennzahlen" bezeichnet.

Metriken werden in der Regel in harte und weiche Metriken unterteilt.

- Harte Kennzahlen - sind objektiv messbar, z.B. durchschnittliche Reaktionszeit, Anzahl der Fehler pro Zeiteinheit oder Garantiezeit.

- Weiche Kennzahlen - sind subjektiver und sehr schwer objektiv zu messen. Sie können auf einer Skala von typischerweise 0 - 100 in der Reihenfolge der bewerteten Produkte bewertet werden oder alternativ kann eine mündliche Bewertung durchgeführt werden.

Darüber hinaus gibt es eine Vielzahl von Systemen zur Qualitätskontrolle der Softwaresystementwicklung. Im Großen und Ganzen mögen Kunden, wenn ihre Produkte das Qualitätszertifikat haben. Tatsächlich zahlen sie gerne zusätzliches Geld dafür. Diese Systeme helfen, Fehler zu minimieren und die Effizienz des Entwicklungsprozesses zu steigern. Einige davon werden im folgenden Text kurz vorgestellt.

8.1. CMMI

Fähigkeits-Reifegradmodell (Integration). In den USA und Japan ist dieses Modell bemerkenswert weit verbreitet. Es ist für Entwicklungsteams konzipiert. Dieses Modell ist relativ aufwendig

und kann daher als Leitfaden für die Verbesserung der Qualität des Entwicklungsteams dienen. Es besteht aus einer Reihe von Regeln und Empfehlungen, die für eine schnelle Entwicklung und Gestaltung neuer Produkte zu beachten sind. Darüber hinaus konzentriert es sich auf die Organisation, Planung und Überwachung von Entwicklungsprozessen. Sie wird durch die Aufteilung der Entwicklungsteams in fünf Reifegrade und Kapazitäten festgelegt. Auf diese Weise können sich einzelne Teams innerhalb dieser Ebenen bewegen. Die Reifegrade werden von einem gut ausgebildeten Gutachter in einem genauen Verfahren bewertet.

Die Reifegrade sind (laut Wikipedia)

- Initial: Auf dieser Ebene führen Teams diese Prozesse entweder gar nicht oder nur teilweise durch.
- Verwaltet: Das Projektmanagement ist definiert und die Aktivitäten sind geplant.
- Definiert: Die Verfahren werden definiert, dokumentiert und verwaltet.
- Quantitativ verwaltet: Produkte und Prozesse werden quantitativ verwaltet.
- Optimieren: Das Team optimiert kontinuierlich seine Aktivitäten.

Kapazitätsstufen

- Unvollständig: Einige Aktivitäten werden nicht ausgeführt.
- Durchgeführt: Es werden Aktivitäten durchgeführt.
- Verwaltet: Die Tätigkeiten werden entsprechend ihrem Management durchgeführt.
- Definiert: Die Aktivitäten werden gemäß ihrer Definition durchgeführt.

8.2. ISO 9000:2001

Im Gegensatz zu CMMI ist es nur lose definiert. Es definiert das Qualitätsmanagementsystem. Diese Norm ermöglicht es spezifischen Unternehmen, ihre Kapazitäten für die Produktion und den Vertrieb von Produkten nachzuweisen. Sie ist dabei nur lose definiert. Der Entwicklungsprozess ist daher sehr spezifisch. Aus diesem Grund wurde die Interpretation des englisch-schwedischen TickIT oder der ISO/IEC 90003 durchgeführt.

8.3. Six Sigma

Es handelt sich um eine Reihe von Techniken für das Prozessmanagement, deren Ziel es ist, seine Effizienz zu verstehen und kontinuierlich und regelmäßig zu verbessern oder die Anforderungen des Kunden durch Verständnis der Kundenbedürfnisse, Prozessanalyse, Standardisierungsmessverfahren und deren Analyse mit statistischen Methoden zu erfüllen.

Grundlegende philosophische Prinzipien dieser Methodik sind:

- Eine nachhaltige Anstrengung, um stabile und vorhersehbare Ergebnisse des Prozesses zu erzielen, ist entscheidend für den wirtschaftlichen Erfolg.
- Produktions- und Geschäftsprozesse haben Eigenschaften, die definiert, bewertet, analysiert, verbessert und kontrolliert werden können.
- Um eine ständige Qualitätsverbesserung zu erreichen, muss das gesamte Unternehmen, einschließlich des Top-Managements, eingebunden werden.
- Zu den wichtigsten Techniken gehören beispielsweise der DMAIC-Zyklus - er dient dem Verständnis und der Steuerung von Prozessen.

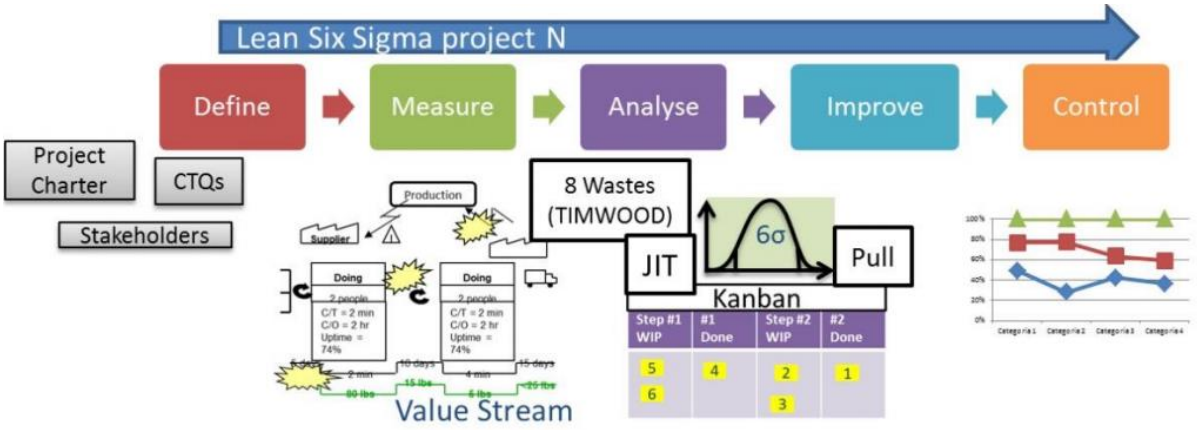


Abb.: *Quelle - http://www1.osu.cz/~zacek/infs2/02.pdf*

DMAIC besteht aus 5 Phasen:

- **Definieren** - Definieren eines tatsächlichen und idealen Prozesses. Es schlägt einen Weg vom eigentlichen zum idealen Prozess vor.

- **Messen** - Messung des aktuellen Prozesses. Es bedeutet zum Beispiel die durchschnittliche Wartezeit, die uns mit "Hard Data" über die "objektive Realität" versorgt.

- **Analyse** - die Messergebnisse werden statistisch ausgewertet, um ein Arbeitsmodell für ein bestimmtes Projekt zu erstellen; alternativ versucht die Analyse herauszufinden, welche Teile des Prozesses verbessert werden müssen.

- **Verbessern** - die Umsetzung von Gedanken aus der vorherigen Phase. Zu den wichtigsten Zielen gehören die Zufriedenheit des Kunden, die Steigerung der Effizienz und die Senkung der Kosten.

- **Kontrolle** - wenn eine Änderung erfolgreich eingeführt wurde, muss sie standardisiert und kontrolliert werden, ob sie eine Verbesserung gebracht hat oder nicht.

Diese Phase wurde ursprünglich von der Motorola Company eingebracht. Diese Phase wird heute unter anderem von Honeywell, HP, Texas Instruments, NASA und anderen Unternehmen genutzt. Sie basiert auf angewandten statistischen Verfahren, die mit qualitativen Werkzeugen ausgestattet sind.

9. WARTUNG UND BETRIEB VON SOFTWARESYSTEMEN

Bisher haben wir uns nur mit der Softwareentwicklung beschäftigt. Dennoch markiert die Softwareentwicklung erst den Beginn des Software-Lebenszyklus. Es ist vor allem der Betrieb, der den entscheidenden Teil ausmacht. Genauer gesagt, wenn wir ein leistungsfähiges Softwaresystem entwickeln würden, das nach einem Monat seines Betriebs aufgrund seiner schlechten Wartung regelmäßig ausfallen würde, würde der Kunde sicherlich das Gefühl haben, dass das gesamte System schlecht entwickelt ist. Daher ist es notwendig, eine regelmäßige Wartung durchzuführen, damit die Software effektiv funktioniert.

ITIL konzentriert sich auf die Techniken der Wartung und des Betriebs von Informationstechnologien. Es handelt sich um eine Reihe von Konzepten und Techniken für eine effektivere Nutzung von IT-Services. IT-Services umfassen IT-Systeme, deren Hauptziel es ist, Geschäftsprozesse zu stimulieren (d.h. die Unternehmen müssen ihre Arbeit richtig machen).

ITIL ist eigentlich ein Band von Buchpublikationen, die wertvolle Erfahrungen im Bereich des Service Managements von Informationstechnologien enthalten. Damit wird ein Rahmen für das IT-Service-Management geschaffen, der uns sagt, wann und was zu tun ist. Das ITIL-Konzept ist eigentlich sehr einfach; oder genauer gesagt, warum wir den gesamten Prozess von Anfang an gestalten und entwickeln sollten, wenn viele andere Unternehmen ihn bereits eingeführt und regelmäßig verbessert haben.

ITIL verwendet einen proaktiven Ansatz. Es geht darum Probleme im Voraus und nicht rückwirkend festzustellen. Wenn ein Problem bereits aufgetreten ist, versucht es, es durch aktive Erkennung herauszufinden, für welche es korrekte Verfahren festlegt. Der gesamte Prozess wird regelmäßig gesteuert, überwacht, bewertet und ständig verbessert. Alle diese Prozesse sollen dem Kunden einen Mehrwert bieten. Die Terminologie kann jedoch in einigen Fällen, in denen Unternehmen unterschiedliche Fachbegriffe verwenden, ein Problem darstellen. ITIL strebt dabei eine Standardisierung der Terminologie an. Diese Verfahren sind plattformneutral etabliert.

Der größte Vorteil dieser Ansätze ist wahrscheinlich ihre Effizienz, dank derer beispielsweise die Dauer von IT-System-Ausfällen reduziert werden kann.

Aus diesem Grund wurden zwei grundlegende Prozesse Service Support und Service Delivery eingeführt. Innerhalb dieser Prozesse wurde eine Kundenkontaktstelle - Service Desk - eingeführt. Dieser Punkt sammelt die Anforderungen der Kunden oder Benutzer. Außerdem werden IT-Prozesse erfasst, so dass sie reagieren und die Anforderungen erfüllen können. Es bietet auch grundlegende IT-Unterstützung.

Im Problemfall findet der Incident Management Prozess statt, der versucht, unangenehme Folgen für Kunden bei Problemen mit IT-Systemen zu minimieren. Eines der wichtigsten Kriterien ist die schnelle Lösung des Problems.

Darüber hinaus wird das Vorfallmanagement durch das Problem Management unterstützt, das das Register zur Problembehandlung verwaltet. Gleichzeitig analysiert es bestehende Probleme und deren Charakter und führt gegebenenfalls strukturelle Veränderungen der IT-Systeme ein.

Einzelne Änderungen werden im Change Management erfasst. Release-Management plant und verwaltet individuelle Änderungen von IT-Services (Release)

ITIL Version 3 besteht aus 5 Teilen und einem Einführungsbuch; (Žáček, 2017) beschreibt sie wie folgt:

- Service-Strategie - befasst sich mit der Harmonisierung von Business und IT, Management-Strategie von IT-Services, Planung.
- Service Design - befasst sich mit IT-Services, Prozessplanung (Erstellung und Pflege von IT-Architekturen und -Verfahren).
- Service Transition - überträgt IT-Services in die Geschäftsumgebung.
- Servicebetrieb - liefert und verwaltet Prozessaktivitäten, Anwendungsmanagement, Änderungen, Betrieb und Kennzahlen.
- Kontinuierliche Serviceverbesserung - befasst sich mit IT-Services, Verbesserungskompetenzen, Methoden, Praktiken und Metriken.

10. LITERATUR

Eysenck, M. W., & Keane, M. T. (2008). Kognitivní psychologie. Praha: Academia.

Nolen-Hoeksema, S. (2012). Psychologie Atkinsonové a Hilgarda (Vyd. 3., přeprac.). Praha: Portál.

Sklenář, V. (2007). SOFTWAROVÉ INŽENÝRSTVÍ [Online]. Retrieved June 29, 2018, from https://phoenix.inf.upol.cz/esf/ucebni/syspro.pdf

Softwarové inženýrství [Online]. (2001-). In Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation. Retrieved from https://cs.wikipedia.org/wiki/Softwarov%C3%A9_in%C5%BEen%C3%BDrstv%C3%AD#Softwarov%C3%A1_krize

Plháková, A. (2004). Učebnice obecné psychologie. Praha: Academia.

Rational Unified Process [Online]. (2001-). In Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation. Retrieved from https://en.wikipedia.org/wiki/Rational_Unified_Process

Rychta, A. (2011). Softwarové inženýrství [Online]. Retrieved June 29, 2018, from http://www.ksi.mff.cuni.cz/~richta/NSWI026/NSWI026-1-Uvod.pdf

Říčan, J. (2016). Používané metakognitivní strategie žáků pátých tříd ve specifické doméně čtení (Disertační práce). Praha.

US Department of Justice (2003). INFORMATION RESOURCES MANAGEMENT Chapter 1. Introduction.

Vondrák, I. (2002). Úvod do softwarového inženýrství [Online]. Retrieved June 29, 2018, from http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf

WHITE, Stephen A. Business Process Modeling Notation [online]. [cit. 2018-06-26]. Dostupné z: https://is.muni.cz/el/1433/jaro2014/PV165/um/46771256/pr_06_bpmn.pdf

Žáček, J. (2017). SOFTWAROVÉ INŽENÝRSTVÍ [Online]. Retrieved June 29, 2018, from http://www1.osu.cz/~zacek/sweng/skripta_sweng.pdf