

2 In-situ Nutzungsinformationserfassung und beteiligungsorientierte Softwareentwicklung

Dieses Kapitel stellt die Erkenntnisse aus aktuellen Arbeiten zu Usability-Engineering-Prozessen in Theorie und Praxis sowie zu Community-Innovation (Abschnitt 2.1) und zu Remote-Usability- und Nutzerfeedback-Verfahren (Abschnitt 2.2 und 2.3) dar. Zudem werden in Abschnitt 2.4 zum einen Methoden und Werkzeuge aus dem Participatory-Design und zum anderen Motivationsfaktoren und -mechanismen zu Nutzerbeteiligung sowie Konzepte zur Aneignungsunterstützung vorgestellt. Am Ende jedes Abschnitts werden stichpunktartig die wesentlichen Erkenntnisse in einem Kasten aufgelistet. Eine Zusammenfassung aller Erkenntnisse wird in Abschnitt 2.5 gegeben und die identifizierten Forschungslücken, die diese Arbeit adressiert, werden in Abschnitt 2.6 beschrieben.

2.1 Usability-Engineering-Prozesse

In diesem Abschnitt werden etablierte Usability-Engineering-Prozesse und -Vorgehensmodelle betrachtet, die in den letzten zehn Jahren entstanden sind. Dabei ist für diese Arbeit von besonderem Interesse, wie der Usability-Diskurs die konkrete Nutzungsphase und heterogene und verteilte Nutzer im Entwicklungsprozess berücksichtigt. So sollen in diesem Abschnitt folgende Fragen beantwortet werden:

- Wie wird die evolutionäre Softwareentwicklung berücksichtigt?
- Wie werden Endnutzer nach der Auslieferung der Software in den Weiterentwicklungsprozess eingebunden?
- Wie werden verschiedene Verteilungsarten von Nutzern und Entwicklern berücksichtigt?

Usability-Engineering (UE) ist die Disziplin in der Mensch-Maschine-Interaktion-Forschung, die die Sicherstellung der Gebrauchstauglichkeit als Ziel hat. UE wird in der Literatur folgendermaßen definiert:

„Usability Engineering is a discipline that provides structured methods for achieving usability in user interface design during product development.“ (Mayhew, 1999, S. 2)

Dabei sind die Hauptaufgaben des Usability-Engineerings im Wesentlichen herauszufinden, wer überhaupt die Nutzer sind, deren Arbeitstätigkeiten zu ana-

lysieren, geeignete Funktionalitäten zu bestimmen und passende Nutzerschnittstellen zu entwickeln.

Will man die relevantesten Modelle vorstellen, müssen insbesondere zwei Disziplinen betrachtet werden. Traditionell beschäftigt sich Software-Engineering (SE) mit der Entwicklung von Softwareanwendungen. In etablierten und renommierten SE-Vorgehens- und -Entwicklungsmodellen, wie z.B. dem *Rational-Unified-Prozess* (RUP) oder der *agilen Softwareentwicklung*, haben Tätigkeiten im Bereich Anforderungsanalyse und -management bereits einen hohen Stellenwert. Da sie für diese Arbeit sehr interessant sind, werden sie in der Folge genauer betrachtet. Auf der anderen Seite existieren seit gut zehn Jahren nutzer-orientierte Vorgehensmodelle, die die Anwendung von UE-Methoden und -Werkzeugen in Entwicklungsprozessen beschreiben. Es ist die Disziplin des *User Centered Design* (UCD), die sich mit der systematischen Einbindung von späteren Nutzern und entsprechenden Werkzeugen in den Softwareentwicklungsprozess beschäftigt und versucht, die UE- und SE-Welt näher zusammenzubringen (Richter & Flückiger, 2013). Nun folgend werden SE- und UE-Modelle beschrieben und aus den oben genannten Blickwinkeln betrachtet.

2.1.1 Usability- und Software-Engineering-Prozesse

Begonnen wird mit der Beschreibung von zwei Vorgehensmodellen aus dem Software-Engineering.

2.1.1.1 Rational Unified Process (RUP)

RUP ist ein in der Softwareentwicklung sehr etabliertes Vorgehensmodell und eines der wenigen, das ein iteratives Vorgehen vorschreibt. Der Prozess ist im Wesentlichen eine Zusammensetzung aus vielen verschiedenen Softwareentwicklungsprozessen (Jacobson et al., 1999). Die Anforderungsanalyse ist eine von sechs primären „Disziplinen“, wie in Abbildung 1 zu sehen ist. Dabei geht es um das Erfassen, die Dokumentation und die Verwaltung von Anforderungen der verschiedenen Akteure. Als besondere Interessensgruppe spielen Nutzer dabei eine wesentliche Rolle.

Der Prozess berücksichtigt zwar nicht direkt Usability und fokussiert eher auf Anforderungsmanagement, aber eines der wichtigen Merkmale des Prozesses sind Use-Cases. Use-Cases werden hier als Instrument der Anforderungsermittlung genutzt (Kruchten, 1999). In der praktischen Anwendung allerdings spielen Use-Cases nur eine untergeordnete Rolle im Vergleich zur Systemarchitekturentwicklung (Göransson et al., 2003). Use-Cases werden nur zu Beginn eines jeden Zyklus zur Planung eingesetzt. Doch nach dem Start einer neuen Iteration sind sie nur als Vorversion der Elemente der internen Funktionsbeschreibung zu sehen. Seit 2002 existiert ein User- Experience-Plugin

(UX) für RUP (Gornik, 2004). Dabei werden die Use-Case-Beschreibungen mit UX-Elementen erweitert. Dazu gehören z.B. eine erweiterte Beschreibung von Nutzercharakteristiken, Angaben zu Usability-Anforderungen oder die Definition von UX-Elementen. Obwohl das UX-Plugin schon einen Schritt in Richtung der Integration von Usability-Maßnahmen geht, deckt dieser Zusatz nicht den gesamten Prozess ab.

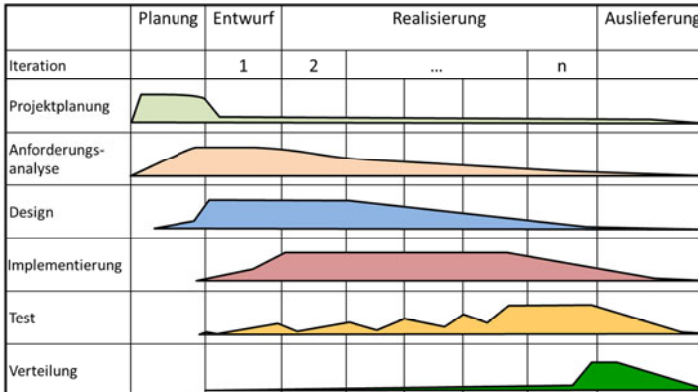


Abbildung 1: Disziplinen im Rational Unified Process (RUP) (Kruchten, 1999)

Gulliksen et al. (Gulliksen et al., 2003) verglichen ein RUP basiertes Entwicklungsprojekt mit User-Centered-Design-Prinzipien. Dabei wurde deutlich, dass der Fokus von RUP nicht auf einer langfristigen Betrachtung des kompletten Softwarelebenszyklus liegt. Es stehen nur kurzfristige Ziele wie die Entwicklung von Modellen oder Spezifikationen im Vordergrund. Langfristige Ziele und Bedürfnisse der Nutzer hinsichtlich der zukünftigen Entwicklungen in der Arbeitspraxis werden ignoriert. Der Term „Iteration“ wird in diesem Prozess anders als im UCD-Diskurs gedeutet. Bei RUP ist mit Iteration eine bestimmte Abfolge von Aktivitäten gemeint, die in einer neuen Version mündet. Die Aktivitäten innerhalb einer Iteration werden wasserfallartig abgearbeitet. Dabei sind innerhalb eines Zyklus keine Rücksprünge vorgesehen. Usability-Ergebnisse sind bei RUP Teil des Anforderungsmanagements, spielen nur in den Anfangsphasen eine Rolle und werden im Laufe eines Projektes nicht weiter verifiziert. Des Weiteren ist die Use-Case-Dokumentation sehr umfangreich und zeitaufwendig. Außerdem wird die Spezifizierung der Nutzerschnittstelle in *Unified Modelling Language* (UML) beschrieben, was für Nutzer schwer zu verstehen und somit schwer zu validieren ist.

2.1.1.2 V-Modell XT

Das V-Modell XT bildet einen Standard zum systematischen Vorgehen bei der Entwicklung von komplexen Systemen. V-Modell XT ist ein etablierter Entwicklungsstandard für IT-Vorhaben in Unternehmen, aber auch bei kleinen und mittelständischen Unternehmen und ist bei zivilen und militärischen Vorhaben des Bundes empfohlen oder gar verbindlich vorgeschrieben. Das V-Modell XT stellt dabei drei Projekttypen bereit:

1. Vorgehensmodell für Auftraggeber *für die Durchführung von Projektausschreibungen und Abnahmen;*
2. Vorgehensmodell für Auftragnehmer *für die Durchführung von Systementwicklungen und Abnahmen;*
3. Vorgehensmodell zur Unterstützung bei der Einführung und Pflege eines organisationsspezifischen Vorgehensmodells.

Für weitere Details siehe (Broy & Rausch, 2005)

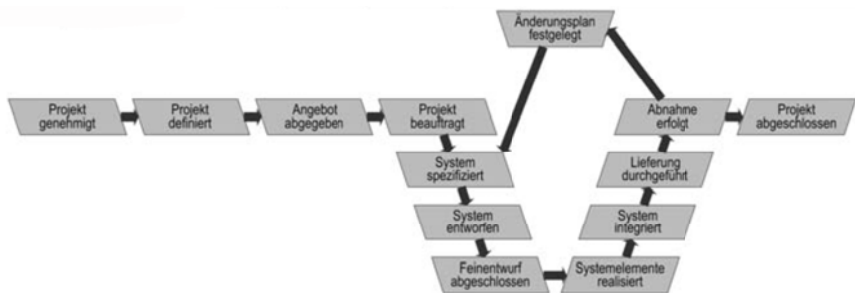


Abbildung 2: V-Modell XT aus Sicht eines Auftragnehmers (Broy & Rausch, 2005)

Die aktuelle Version des Modells ging aus dem V-Modell 97 hervor und wurde insbesondere hinsichtlich der Anpassbarkeit und Qualitätssicherung verbessert. Abbildung 2 zeigt neben den Projektanbahnungsphasen auch die Phasen der Systementwicklung. Die Systementwicklung folgt einer hierarchischen Zerlegung des Systems in kleinere Einheiten, dass diese eine Realisierung ermöglichen. Dementsprechend werden während der Systementwicklung die Spezifikation und die Zerlegung vorgenommen. Für jeden dieser Zerlegungsschritte wird ein präzises Vorgehen vorgegeben, das auf einem einheitlichen Muster basiert, um eine lückenlose Verfolgung der Umsetzung der Anforderungen sicherzustellen. Die Anforderungen aus den vorherigen Schritten werden zunächst übernommen, daraufhin wird die Zerlegung vorgenommen, die Realisierung der Systemelemente spezifiziert und schließlich werden die Anforderungen für die nächsten Schritte definiert. Die Realisierung und Integration des Systems erfolgt

im Vergleich zu Spezifikation und Zerlegung in umgekehrter Reihenfolge. Die Verifikation und Validierung der Ergebnisse werden auf jeder Stufe durchgeführt.

Im Gegensatz zum V-Modell 97 sind im V-Modell XT die Produkte und Aktivitäten, die die Qualitätssicherung betreffen, nicht mehr nur im Vorgehensbaustein (so werden die einzelnen Phasen im Modell genannt) *Qualitätssicherung* enthalten, sondern integrieren sich in die jeweiligen anderen Vorgehensbausteine, z. B. *Anforderungsfestlegung*, *Systemerstellung*, *SW-Entwicklung*, sodass die Qualitätssicherung fortlaufend im gesamten Projekt durchgeführt wird.

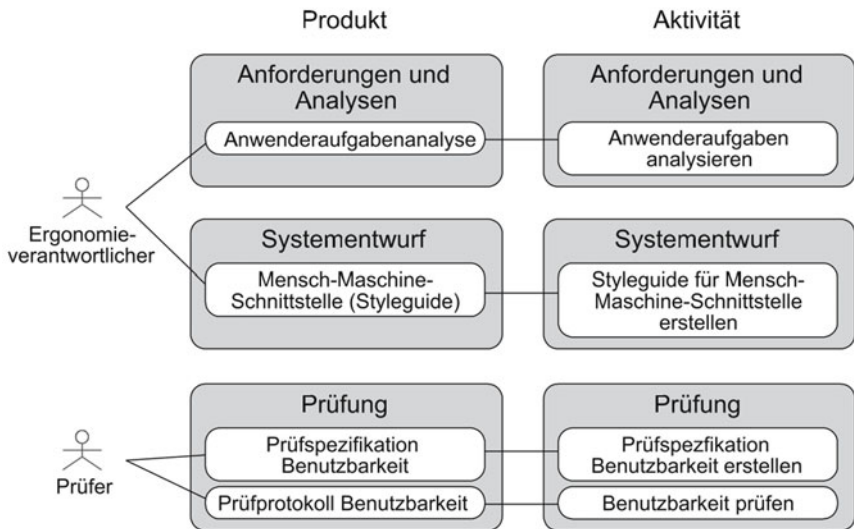


Abbildung 3: Qualitätssicherung im Vorgehensbaustein Benutzbarkeit und Ergonomie (Höhn, 2008)

Konkret wird seit der neuen Version des V-Modells auch das Thema Usability in der Qualitätssicherung adressiert. Ein neuer Vorgehensbaustein *Benutzbarkeit und Ergonomie* (s. Abbildung 3) und eine neue Rolle, die des *Ergonomieverantwortlichen*, wurden geschaffen. Diese Rolle ist für die Umsetzung ergonomischer Forderungen im Gesamtsystem, d.h. für System, Software, Hardware etc., verantwortlich und stellt ein wesentliches Bindeglied zwischen Nutzer und Auftragnehmer dar. Außerdem muss der Ergonomieverantwortliche die Durchführung der Anwenderaufgabenanalyse und die Gesamtgestaltung der Benutzeroberfläche sicherstellen. Dies tut er, indem er Regeln und Gestaltungskri-

terien in Form von Styleguides definiert. Auf Basis der Anwenderanalyseergebnisse wird eine Prüfspezifikation „Benutzbarkeit“ erarbeitet, die als Grundlage für die Überprüfung der Produkte dient.

Vergleicht man das V-Modell XT nun mit den Anforderungen einer evolutionären und verteilten Softwareentwicklung, ist zwar die neue Berücksichtigung von Usability im Vorgehensmodell herauszustellen, doch Maßnahmen sind nur bis zur Auslieferung bzw. der Abnahme der Software vorgesehen. Eine Unterstützung über den gesamten Lebenszyklus einer Software hinweg ist nicht vorgesehen. Des Weiteren werden bis jetzt keine kleineren Einzelprojekte und -schritte im V-Modell berücksichtigt, da sich der Einsatz des Modells bis jetzt nur auf größere Softwareprojekte konzentrierte. Durch die Berücksichtigung von kleineren Projektschritten kommt der Synchronisation und inhaltlichen Integration eine wichtigere Rolle zu. Bei der Systementwicklung ergibt sich daraus ein noch stärkerer Bezug auf die Komponentenstruktur von Software. Der Bedarf begründet sich ebenfalls durch die immer populärer werdenden agilen Softwareentwicklungsansätze, die eine stark flexible Systemarchitektur benötigen. Das bedingt, dass einzelne Komponenten auch mit eigenen Prozessen entwickelt und die daraus resultierenden vielfältigen nebenläufigen (Software-) Prozesse synchronisiert werden müssen. Der Skalierbarkeit der Prozesse, d.h. die Anwendbarkeit auf ganz kleine bis ganz große Software-Einheiten wird eine der wichtigsten Forderungen sein. Ein weiteres wichtiges Defizit betrifft die Kooperation zwischen Entwicklern und Nutzern (Höhn, 2008). Es müssen korrespondierende Prozesse zwischen diesen beiden Rollen integriert und gezeigt werden, wie diese harmonisiert und synchronisiert werden können.

2.1.1.3 Agile Softwareentwicklung

Agile Softwareentwicklungsmethoden werden im Software-Engineering immer populärer. Dies ist besonders auf stark organisierte und dokumentationsintensive Vorgehensmodelle, wie des zuvor dargestellte Rational Unified Process (RUP) oder auch das V-Modell XT, zurückzuführen. Die agile SE-Community argumentiert, dass insbesondere die aufwendige Dokumentation und die starren Prozesse die Entwicklung ineffizient machen und wesentlich verlängern (Richter & Flückiger, 2013). Gerade die stetige Veränderung der Nutzeranforderung setzt eine reaktionsschnelle Entwicklungsinfrastruktur voraus, die hier zur Verfügung gestellt wird. Für die agile Entwicklungsszene ist die Berücksichtigung von Nutzerfaktoren im Entwicklungsprozess von entscheidender Bedeutung. Das zeigt sich, da sich zwei der vier Säulen im agilen Softwareentwicklungsmanifest allein drauf beziehen: 1) Individuen und Interaktionen vor Prozesse und Werkzeuge und 2) Zusammenarbeit mit dem Kunden vor Vertragsverhandlungen („Manifesto for Agile Software Development,” n.d.; Seffah et al., 2005). Somit

kommt der agile Software Entwicklungsansatz schon sehr nah an die Grundsätze der nutzerzentrierten Software-Entwicklung.

Auch wenn die agile Softwareentwicklung die aktuell existierenden Rahmenbedingungen der Praxis schon gut berücksichtigt, sind auch hier noch wichtige Grundsätze nicht konsequent genug umgesetzt (Blomkvist, 2005; Sutcliffe, 2005). Da ist bspw. die nicht deutliche Trennung von Kunden und Endnutzern. In Softwareprojekten sind Kunden meist Vertreter des Managements, der Marketingabteilung. Also meist nicht Personen, die als spätere Endnutzer zu sehen sind und somit nur wenig mit der operativen Praxis zu tun haben (Ferre et al., 2005). Zudem stellt der Entwicklungsansatz keine Hilfestellungen zur Verfügung, wie es z.B. bei der ISO Norm 9241-210 der Fall ist, die helfen die Umsetzung von Nutzeranforderungen im Laufe der Entwicklung zu überprüfen. Nach Seffah et al. (Seffah et al., 2005) sind Hilfestellungen zur regelmäßigen Evaluation von Nutzeranforderungen gerade bei sozial eingebetteten System wichtig, da soziale Interaktionen und Geschäftsprozesse nur schwer vorhersehbar sind.

Nun folgend werden etablierte Vorgehensmodelle aus dem Usability-Engineering und aus dem User Centered Design vorgestellt und auf die zuvor definierten Fragestellungen hin untersucht.

2.1.1.4 Star Lifecycle

Der Star Lifecycle entworfen von Hix & Hartson (Hix & Hartson, 1993) ist ein nutzerzentriertes Prozessmodell, das die wesentlichen Usability-Aktivitäten beschreibt. Im Zentrum des Modells steht die Usability-Evaluation, wie in Abbildung 4 zu sehen ist. Umringt wird die Usability-Evaluation von verschiedenen Usability-Aktivitäten: (1) Aufgaben- und funktionale Analyse, (2) Spezifikation der Anforderungen, (3) Entwicklung eines Konzeptes, (4) Entwicklung von Designentwürfen, (5) Einführung des Systems. Auch wenn es die Nummerierung suggeriert, setzt das Vorgehensmodell keine bestimmte Reihenfolge voraus. Die Resultate aus jeder dieser Aktivitäten werden evaluiert, bevor zu einer weiteren Prozessaktivität fortgeschritten wird. Die bidirektionale Verbindung zwischen der zentralen Usability-Evaluation und den einzelnen Prozessaktivitäten resultiert in dem charakteristischen Sterngebilde des Modells. Darüberhinaus beschreiben Hix & Hartson Kommunikationspfade die zwischen Usability- und Software-Design-Aktivitäten existieren sollten. So wird die Entwicklung des Nutzerinterfaces strikt von der Entwicklung des restlichen Systems getrennt und ist nur durch zwei Aktivitäten *Systemanalyse* und *Usability Evaluation* mit einander verbunden. Das Modell ist auf spezifische Anwendungskontexte anpassbar.

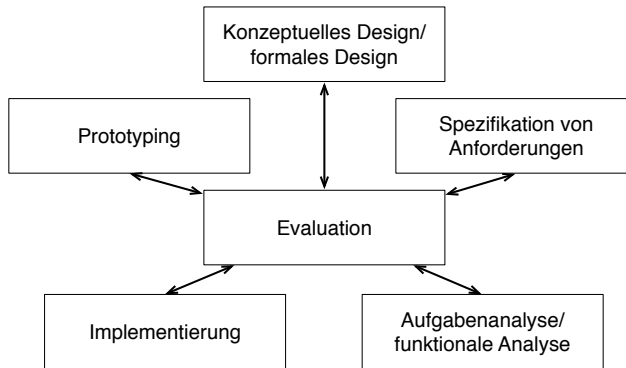


Abbildung 4: Star Lifecycle (Hix & Hartson, 1993)

Dennoch ist die Anwendung des Modells in der Praxis schwierig (Seffah et al., 2005). Zwar werden spezielle Methoden und Notationen, wie die *User Action Notation* (UAN) angeboten, das auf solche Art und Weise Nutzerinteraktionen spezifiziert, dass diese auf einfache Weise für die Gestaltung des Nutzerinterfaces verwendet werden können. Zudem ist es auch wegen dem iterativen Vorgehen sehr schwierig, den Endpunkt eines Entwicklungszyklus überhaupt zu bestimmen. Eine Definition oder ein Endpunkt wird hier nicht bereitgestellt. Das macht die Ressourcenplanung sowie die generelle Planung eines Entwicklungsprojektes schwierig. Hix und Hartson (Hix & Hartson, 1993) empfehlen, hier einen Top-Down-Ansatz zu wählen, sofern das Projektmanagement oder das Entwicklerteam schon Vorerfahrungen hinsichtlich der Struktur des Zielsystems besitzen. Laut den Autoren hängt dies aber von der Projektgröße, der Anzahl der involvierten Personen und dem Managementstil ab. Sie empfehlen ebenfalls Usability-Engineering als Prozess zu sehen, jedoch sagen sie selbst, dass die Designphase die am wenigsten verstandene Entwicklungsaktivität ist und hier nur die zuvor genannte *User Action Notation* zur Verfügung gestellt wird. Wie sonst die Evaluation und die Entwicklung zusammen interagieren wird nicht beschrieben.

2.1.1.5 Usability Engineering Lifecycle

Deborah Mayhews Usability Engineering Lifecycle (Mayhew, 1999) ist der Versuch die Softwareentwicklung dahingehend umzugestalten, dass sich die gesamte Entwicklung einzig um Wissen, Methoden und Aktivitäten des Usability-Engineerings dreht. Der Prozess wird dabei in drei Phasen aufgeteilt (s. Abbildung 5): Eine Anforderungsanalyse zu Beginn. Darauf folgenden Entwicklung und Tests, der Prozess endet mit der Installation des Systems. Der Prozess folgt dem klassischen Wasserfallmodell. Rückschritte sind nur dann

möglich, wenn geforderte Anforderungen nicht adressiert wurden. Ist dies der Fall, ist die Anwendung von verschiedenen Usability-Methoden wie das Erstellen von Papier-Mockups, Prototyping oder Usability-Testing vorgesehen, bis die zuvor definierten Usability-Ziele erreicht wurden. Das Vorgehensmodell wurde bereits in verschiedenen Projekten erfolgreich angewendet (Mayhew, 1999), denn es gilt aus SE-Sicht als eines der kompletten Usability-Prozessmodelle (Ferre et al., 2005). Zwar zielt der Usability Engineering Lifecycle im eigentlichen Sinne nur auf die Gestaltung des Nutzerinterfaces ab, doch beinhaltet es neben anforderungsbezogenen Aktivitäten wie *Contextual Task Analysis* auch Verbindungen zu Objekt-orientiertem Software-Engineering (OOSE) und Rapid Prototyping-Methoden. Dennoch hat Mayhew selbst, durch die Analyse der Anwendung des Prozessmodells in verschiedenen Projekten, noch Defizite entdeckt. Die sind zum einen darauf zurückzuführen, dass die strikte Fokussierung auf Usability-Maßnahmen im Softwareentwicklungsprozess nicht angemessen ist. Gut etablierte Softwareentwicklungsprozesse in Unternehmen müssen berücksichtigt werden und können nicht direkt auf nutzerzentrierte Entwicklungsverfahren umgestellt werden. Zudem bestehen immer noch fehlende Kenntnisse über die Anwendung von Usability-Maßnahmen bei den Entwicklern, was es schwierig macht, UCD-Aktivitäten in Software-Engineering-Prozessen zu etablieren. (Mayhew, 1999)

Im Bezug auf die Unterstützung der evolutionären Softwareentwicklung stellt der Usability Engineering Lifecycle weiterhin ein klassisches mit der Auslieferung einer Software endendes Vorgehensmodell dar. Auch wenn die stetige Überprüfung der Nutzeranforderungen mittels gut etablierter Usability-Verfahren durchgeführt wird, bis eine gewünschte Qualität erreicht ist, ist die Abfolge der einzelnen Entwicklungsstufen nach dem klassischen Wasserfallmodell aufgebaut. Ist ein Softwareprodukt einmal in der letzten Phase, der Installation, angelangt, sind zwar Nutzerfeedbacks aus der Praxis eingebunden, ein Rückschritt in die Designphase ist hingegen nicht mehr vorgesehen.

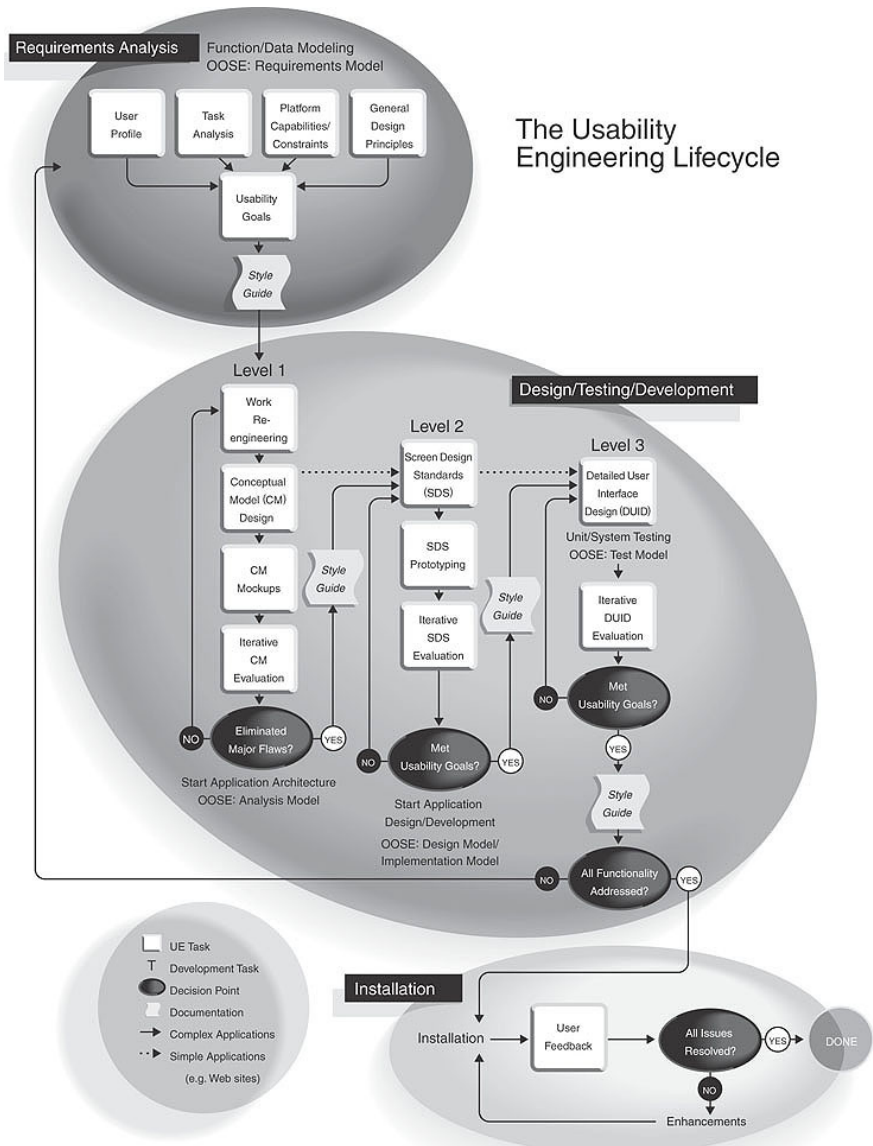


Abbildung 5: Usability Engineering Lifecycle (Mayhew, 1999)



<http://www.springer.com/978-3-658-10861-8>

Community Usability Engineering
Prozesse und Werkzeuge zur In-situ
Feedbackunterstützung
Wiedenhöfer, T.
2015, XVII, 254 S. 45 Abb., Softcover
ISBN: 978-3-658-10861-8