Mentale Modelle

Dipl. Inform. (Univ.) Gerd Ziegler

Plettstr. 73
81735 München
Tel: 0171 - 359 75 74
http://www.gerdziegler.de
kontakt@gerdziegler.de

Abstract. Dieser Artikel untersucht die Bedeutung von Modellen, insbesondere Mentalen Modellen auf einen Softwareentwicklungsprozess. Es wird deren Einfluss auf Änderung, Redundanz und Kommunikation betrachtet. Der Artikel befasst sich mit dem dynamischen Aspekt eines Softwareentwicklungsprozesses und betrachtet die Gründe warum es so schwierig ist den Prozess am Leben zu erhalten. Er betrachtet den Kampf zwischen Ordnung und Chaos, Chaos das ein Softwaresystem so träge machen kann dass die Lösungssuche und der Lernprozess (d.h. die Fähigkeit die Modelle an neue Erkenntnisse anzupassen und synchron zu halten) zum erliegen kommen.

Modelle in der Softwareentwicklung

Softwareentwicklung ist Informationsverarbeitung. Die Informationsverarbeitung beginnt nicht erst mit Inbetriebnahme einer Software, sondern mit deren Erstellung. Die Informationen werden codiert und in verschiedenen an der Softwareentwicklung beteiligten Systemen gespeichert. Auch die am Softwareentwicklungsprozess beteiligten Menschen speichern Informationen.

Dieser Artikel verwendet als zentralen Begriff das mentale Modell (MM). Mental deshalb, weil das Modell in der Gedankenwelt eines Menschen gespeichert ist. Es gibt auch andere Modelle, die nicht in der Gedankenwelt eines Menschen gespeichert sind, das wichtigste Beispiel hierzu ist das Endprodukt des Softwareentwicklungsprozesses, die Software selbst. Diese ist ein Modell das im Computersystem gespeichert ist und dessen Codierung digital ist. Viele Eigenschaften der MM gelten auch für die anderen Modelle.

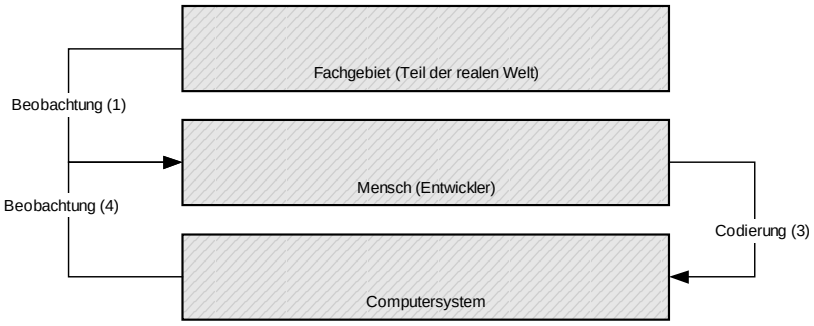
Der Prozess der mentalen Modellbildung ist mit der Softwareentwicklung untrennbar verbunden. Er ist die Voraussetzung für das Verstehen. Ohne ein Verständnis des Fachgebiets, der Software, des Umfelds, der Anforderungen ist keine Softwareentwicklung möglich. Verstehen bedeutet, ein mentales Modell zu besitzen oder in der Lage zu sein, eines zu erstellen. Auf den Softwareentwicklungsprozess angewandt: wir verstehen den Softwareentwicklungsprozess nur wenn wir ein mentales Modell des Prozesses haben, oder in der Lage sind eines zu erstellen.

Dieser Artikel versucht ein für die Praxis der Softwareentwicklung nützliches Modell der Informationsverarbeitung zu skizzieren. Das Modell erhebt nicht den Anspruch formal korrekt zu sein, es ist ein Werkzeug zum Verständnis des Softwareentwicklungsprozesses.

Ein einfacher Softwareentwicklungsprozess

Betrachten wir den einfachen, idealisierten Fall eines Softwareentwicklungsprozesses (SEP), an dem nur ein Entwickler beteiligt ist. Der Entwicklungsprozess besteht aus der Interaktion von 3 Systemen:

- Ein Ausschnitt der realen Welt, das Fachgebiet
- Ein Mensch, der Entwickler
- Ein Computersystem

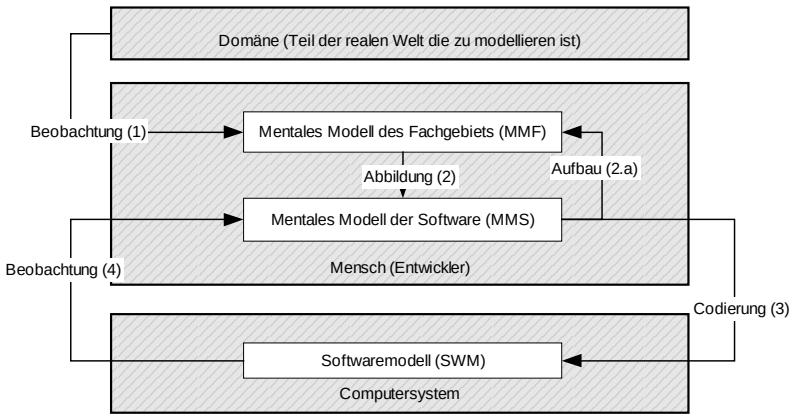


Der Entwickler verwendet Informationen aus dem Fachgebiet (1) und aus dem Computersystem (4) um das Computersystem zu verändern (3).

Modelle

Alle Informationen, die über einen Weg (1, 3, 4) übertragen werden müssen codiert und gespeichert werden. Ein Modell im Sinne des SEP sind in erster Näherung Informationen die in einer bestimmten Form codiert sind und innerhalb eines Systems gespeichert werden.

Im Falle des einfachen SEP gibt es mindestens 3 Modelle, wie in der folgenden Graphik dargestellt.



Der Mensch bildet anhand von Informationen über das Fachgebiet ein mentales Modell des Fachgebiets. Anhand dieses MMF bildet er ein mentales Modell der Software (MMS). Dieses Modell überträgt er durch Codierung in das Softwaremodell (SWM). Der Mensch kann geistige Operationen nur auf MM anwenden, d.h. alles was wir denken erfolgt innerhalb mentaler Modelle. Dies ist wichtig, da meistens das MM mit dem modellierten System verwechselt wird. Diese sind verschieden.

Das mentale Modell des Fachgebiets

Das MMF beinhaltet Begriffe und Konzepte aus dem jeweiligen Fachgebiet. Es ist ein Modell des Fachgebiets, so wie es der Entwickler sieht. Das Modell ist in der Gedankenwelt des Entwicklers gespeichert. Es ist ein mentales Modell (MM).

Das MMF entsteht durch Beobachtung des Fachgebiets (Weg 1) oder wenn sich ein Entwickler in bestehenden Code einarbeitet aus dem MMS (Weg 2.a).

Das mentale Modell der Software

Das MMS beinhaltet Begriffe und Konzepte aus der Informatik bzw. der jeweiligen Softwareumgebung. Es ist ein Modell des Softwaresystems, so wie es der Entwickler sieht. Es ist ein mentales Modell.

Das MMS entsteht aus dem MMF (Weg 2) oder aus dem SWM (Weg 4), wenn schon bestehender Code verarbeitet wird.

Das Softwaremodell

Das Softwaremodell (SWM) ist die Codierung der Software im Quelltext. Die Codierung ist digital und das Modell ist im Computer gespeichert.

Das SWM entsteht aus dem GMS (Weg 3).

Mentale Modelle

Ein mentales Modell ist eine Menge von Begriffen, die miteinander in Beziehung stehen und sich alle in derselben Abstraktionsebene befinden. Jeder Begriff gehört zu genau einem MM. Jeder Begriff hat einen Namen. Der Name eines Begriffs ist eindeutig. Die Identität eines Begriffs ist durch seinen Namen und das MM gegeben. In verschiedenen MM können Begriffe mit demselben Namen auftauchen. Diese Begriffe sind verschieden. Begriffe mit demselben Namen haben in verschiedenen MM verschiedene Bedeutungen/Eigenschaften. Ein MM liegt immer innerhalb der Gedankenwelt eines Menschen. Eine Gedankenwelt (GW) kann mehrere MM enthalten.

Beispiel:
- Die Begriffe eines Fachgebiets können meistens in einem oder mehreren in sich abgeschlossenen MM organisiert werden.
- Die Begriffe aus der GUI-Welt, wie Fenster, Konsole, Dialogfenster, Auswahlbox etc. können in einem MM der GUI (MMGUI) organisiert werden.
- Die Begriffe aus einer speziellen Softwareumgebung können in einem MM der Software (MMS) organisiert werden. In der Java – Welt enthält das MMS Begriffe wie Klasse, Objekt, Methode, Variable, Aufruf, Schleife etc.

Innerhalb eines MM kann argumentiert werden. Eine Argumentation ist eine Folge von Aussagen die Begriffe aus dem MM verwenden. Im Kontext der Softwareentwicklung sind damit weder formal korrekte Aussagen gemeint, noch stellt die Folge von Argumenten einen mathematischen Beweis dar. Sie soll nur nach Einschätzung des Argumentierenden bei seinem aktuellen Kenntnistand als korrekt bzw. plausibel angesehen werden. Sie stellt sozusagen eine Arbeitshypothese dar. Wichtig ist, dass die Argumentation nur Begriffe aus genau einem MM verwendet. Sobald die Argumentation Begriffe eines anderen MM verwendet kann der Argumentation an dieser Stelle keine Bedeutung mehr zugeordnet werden, da die verwendeten Begriffe im aktuellen Kontext (MM) nicht definiert sind. Dies ist für die Praxis bedeutsam, denn meist vermischen wir bei der Diskussion und beim Nachdenken ständig Begriffe aus verschiedenen MM.
Da die Fähigkeiten eines Menschen ein mentales Modell zu bilden begrenzt sind hat ein MM in der Praxis eine Reihe unangenehmer Eigenschaften.

Ein MM ist:
- unvollständig, es erfasst nur einen kleinen Ausschnitt des modellierten Systems
- fehlerhaft, das MM bildet das modellierte System nur unvollkommen ab
- dynamisch, jede neue Informationen kann zur Veränderung des MM führen
- subjektiv, die Veränderung des MM durch neue Informationen hängt vom vorhandenen MM ab
- erfahrungsabhängig, das MM wird aufgrund der vorhandenen Erfahrung erstellt
- nicht eindeutig, zu jedem modellierten System gibt es mehrere MM
- widersprüchlich, oft passen verschiedene Teile eines MM nicht zusammen, da dieses nicht vollständig aufgebaut wurde

- inert, es besitzt eine Art natürlicher Trägheit. Da die Erstellung eines neuen MM aufwendig ist wird der Mensch zu lange an einem schlechten MM festhalten
- irrational, Dinge werden bewusst falsch modelliert um schmerzhafte Erkenntnisse zu vermeiden

Trotz dieser Einschränkungen wird der Mensch das modellierte System durch die Brille des MM wahrnehmen. Eine Änderung oder ein neues MM hat auch eine Änderung der Wahrnehmung des modellierten Systems zur Folge. Oft wird das MM mit dem modellierten System verwechselt.

Beziehungen zwischen mentalen Modellen

MM können in Beziehung zueinander stehen. Eine Beziehung zwischen zwei MM wird hergestellt indem Begriffe des Quell-MM auf Begriffe des Ziel-MM abgebildet werden. Die Abbildung ist im allgemeinen nicht eindeutig und nicht umkehrbar. Ein Begriff im Quell-MM kann auf keinen, einen oder mehrere Begriffe des Ziel-MM abgebildet werden.
In der Praxis gibt es mindestens die Abbildungen MMF $\rightarrow$ MMS $\rightarrow$ SWM.

Gemeinsamkeitsmetrik

Die Gemeinsamkeitsmetrik (G-Metrik) ist eine in der Praxis anwendbare Metrik um die Güte der Abbildung zwischen zwei MM zu messen. Die Metrik misst die Gemeinsamkeiten (logische Nähe) zwischen Begriffen eines MM. Sie sagt aus, dass zwei Begriffe a und b mehr gemeinsam haben als die Begriffe a und c. Die G-Metrik muss nicht für alle Tripel a, b und c definiert sein.
Zwei MM auf denen jeweils eine G-Metrik definiert ist werden mit Hilfe einer Abbildung zueinander in Bezug gesetzt. Dann ist die Abbildung umso besser umso mehr die Gemeinsamkeiten von Begriffen im Quell-MM (nach der Quell-G-Metrik) nach deren Abbildung ins Ziel-MM (unter der Ziel-G-Metrik) erhalten bleiben. Wenn im Quell-MM gemessen mit der Quell-G-Metrik a und b mehr gemeinsam haben als a und c so sollte dies auch für die Abbilder von a, b und c im Ziel-MM unter der Ziel-G-Metrik gelten.
Die Definition ist mit Absicht ungenau gehalten um praxistauglich zu sein. In der Praxis sind bei komplexen MM die Gemeinsamkeiten zwischen Begriffen nicht explizit formulierbar, ja noch nicht einmal die Begriffe eines MM sind vollständig bekannt geschweige denn definiert. Ebenso sind die Abbildungen zwischen MM nicht wohldefiniert.
Auf jedem MM lassen sich mehrere G-Metriken definieren. Eine G-Metrik ist erfahrungsabhängig, nicht vollständig und nicht wohldefiniert. Verschiedene Entwickler (ja sogar derselbe Entwickler) können je nach Standpunkt unterschiedlicher Meinung sein, welche Begriffe mehr gemeinsam haben.
Bleibt eine G-Metrik nach der Abbildung nicht erhalten so ist dies trotzdem ein untrügliches Zeichen dafür dass die Abbildung oder das Modell verbessert werden können.

Beispiel. Sind bestimmte Funktionalitäten im mentalen Modell des Fachgebiets (MMF) einander nahe (im Sinne der G-Metrik) z.B. Druckfunktionalitäten (Begriff aus MMF) so sollte sich diese Nähe auch im dazugehörigen MMS zeigen, etwa indem die Funktionen (=Begriff aus MMS) die diese Funktionalität (Begriff aus MMF) implementieren sich wahlweise in derselben Klasse befinden (Begriff aus MMS) oder in demselben Package befinden (Begriff aus MMS) oder dieselbe Namenskonvention einhalten (z.B. Namen der Form printXxx) (Begriff aus MMS) oder alles zusammen.
Wenn Sie eine GUI für Ihre Anwendung bauen so werden Sie auch ein MM der GUI (MMGUI) haben. Dieses MMGUI wird Begriffe aus der GUI-Welt verwenden, wie Fenster, Menü, Toolbar, Auswahlbox, Benutzerauswahl etc. Hier kann wieder eine G-Metrik definiert werden. Z.B. haben Aktionen die im selben Menü enthalten sind mehr gemeinsam als solche die in verschiedenen Menüs enthalten sind. Wenn Sie nun das MMF auf das MMGUI abbilden so wird eine beispielhafte Abbildung die Druckfunktionalitäten (MMF) auf Auswahlen in demselben Menü (MMGUI) abbilden. Eine Abbildung die die Druckfunktionalitäten (MMF) in dasselbe Menü abbildet wird im Sinne der oben genannten G-Metrik besser sein als eine die die Druckfunktionalitäten in verschiedene Menüs abbildet.
In der objektorientierten Welt kann man auch sagen, daß die Funktionalität (MMF) mit dem Objekt (MMF) auf das sie angewendet wird mehr Gemeinsamkeiten hat als mit einem anderen Objekt (MMF). In diesem Sinne wird die Funktionalität (MMF) dann auf ein Popup-Menü (MMGUI) abgebildet, das nahe an der GUI-Darstellung (MMGUI) des Objekts (MMF) liegt. In der 2-dimensionalen GUI-Welt korreliert Gemeinsamkeit gut mit der Nähe der GUI - Elemente auf dem Bildschirm.

Änderung von Mentalen Modellen

MM sind dynamische Gebilde. Mit jeder neuen Information kann sich ein MM ändern. MM ändern sich indem Begriffe neu erzeugt werden, entfernt werden oder sich die Beziehungen zwischen den Begriffen ändern.
Die Operation hinzufügen eines Begriffs ist indirekt definiert. Einen Begriff B1 zu einem mentalen Modell MM2 hinzufügen setzt voraus, dass es den Begriff B1 gibt. Da Begriffe ausschließlich in MM definiert sind muss es auch ein MM1 geben in dem der Begriff B1 definiert ist. Den Begriff B1 aus MM1 zu MM2 hinzufügen heißt einen Begriff B2 in MM2 zu erzeugen und eine Abbildung zwischen MM1 und MM2 vorauszusetzen, die B1 auf B2 abbildet. Die Bedeutungen von B1 und B2 sind nur innerhalb des jeweiligen MM definiert, also verschieden! Es ist ein Fehler der immer wieder gemacht wird, den ursprünglichen Begriff B1 mit dem neu erzeugten B2 zu verwechseln, bzw. die Eigenschaften eines Begriffes aus einem MM in das andere „mitzunehmen“.
MM können durch Erweiterung bestehender MM gebildet werden. Sie umfassen alle Begriffe eines bestehenden MM und fügen weitere Begriffe hinzu. Wenn Sie z.B. ein MMS bilden so erzeugen Sie das MM nicht komplett neu sondern verwenden ihr Basiswissen über die Software wieder. Trotzdem erzeugen Sie ein oder mehrere neue MM, die voneinander verschieden sind.

Mentale Modelle, Redundanz und Kommunikation

Kommunikation dient dazu, MM aus verschiedenen Gedankenwelten (GW) zueinander in Beziehung zu setzen und zu synchronisieren. Sie dient dazu, MM in anderen GW zu erzeugen, zu löschen und zu verändern.
In diesem Sinne ist Kommunikation, die nicht zur Veränderung von MM führt nutzlos. Oft werden Dokumente zur Kommunikation eingesetzt. Wenn diese aber nicht zur Veränderung von MM führen (z.B. weil keiner sie liest) sind sie nutzlos. Ebenso gefährlich ist die fehlerhafte Kommunikation, die das MM des Kommunikationspartners auf ungewollte Art und Weise verändert. Vielleicht besitzt der Kommunikationspartner das zu verändernde MM gar nicht, oder dieses sieht ganz anders aus als bei Ihnen.
Bei der Kommunikation ist es notwendig, vor dem Verwenden von Begriffen das MM zu definieren in dem sich der Begriff befindet, mit dem man gerade argumentiert, und einen Wechsel des MM erstens sich selber bewusst zu machen und zweitens dem Gesprächspartner mitzuteilen. Der Wechsel findet öfter statt als man denkt. Manche Diskussionen sind ein einziger Wirrwarr aus Begriffen aus verschiedenen MM.

Zahllose Missverständnisse können mit einer im Prinzip einfachen Technik vermieden werden:
Es hat sich bewährt, das MM als Kästchen auf ein Blatt Papier zu zeichnen. Die wichtigsten verwendeten Begriffe mit den wichtigsten Beziehungen sollten ebenfalls in das Kästchen gezeichnet werden. Trifft man bei einer Argumentation oder einem Gedankengang auf Begriffe die nicht zum Kästchen gehören so muss das Modell zumindest gedanklich um Begriffe erweitert werden. Hier sollte man konsequent sein: entweder wird das Modell erweitert oder die Argumentation abgebrochen. Sie werden überrascht sein, wie schnell sich die Begriffe klären.
MM aufzuzeichnen ist besonders dann wichtig wenn verschiedene Alternativen erläutert werden. Hier birgt die Tendenz zu schnell und zu oft zwischen den alternativen MM zu wechseln eine große Verwechslungsgefahr. Jede Alternative sollte ein MM und ein Kästchen bekommen und der Wechsel des MM sollte sich bei allen an der Diskussion beteiligten Personen und auf der Zeichnung vollziehen. Hier ist eiserne Disziplin nötig, und die aufgezeichneten MM sind da eine gewisse Hilfe.

Redundanz zwischen mentalen Modellen

Sind MM redundant, so hat dies den Vorteil, dass die MM schon synchron sind und daher keine Kommunikation mehr notwendig ist um diese zu synchronisieren. Redundanz ist zwischen sich selten und wenig ändernden MM sinnvoll.
Sind MM redundant so hat dies den Nachteil, dass die Redundanz gepflegt werden muss. Eine Änderung in einem MM hat eine Änderung im redundanten MM zur Folge. Jede Änderung verursacht Aufwand. Wenn die Änderung nicht erfolgt, entsteht ein inkonsistenter Zustand. In der Praxis tritt dieser Fall häufig ein. Der Umgang mit Redundanz ist ein wichtiger Aspekt bei der Softwareentwicklung, da die Pflege der Redundanz entscheidend für die Korrektheit der Software und für den Aufwand der

Entwicklung ist. Dank dieser negativen Eigenschaften ist es erstrebenswert, die Redundanz zwischen sich oft und stark ändernden MM gering zu halten.
Redundanz zwischen MM kann auftreten zwischen MM in derselben GW und zwischen MM in verschiedenen GW.

Redundanz zwischen mentalen Modellen in derselben Gedankenwelt

Wie schon gesehen besitzt jeder Entwickler ein oder mehrere mentale Modelle der Software die er gerade erstellt (MMS). Hier verwendet er Begriffe die im jeweiligen Zielsystem definiert sind. Bei Java wären dies Klassen, Objekte, Funktionen, Variablen etc. Ebenso hat jeder Entwickler MM von Teilbereichen des Fachgebiets das zu modellieren ist (MMF). Hier tauchen Begriffe auf, die dem jeweiligen Fachgebiet angehören.
Diese MM sollten redundant sein. Je besser des MMS das MMF widerspiegelt umso besser ist die Software. Im Idealfall kann das MMS direkt aus dem MMF mit Hilfe einer Abbildung erzeugt werden und ist damit voll redundant. Leider ist dies in der Praxis nicht der Fall. Die Abbildung die das MMS erzeugt gibt es im Regelfall nicht, bzw. sie ist je nach Entwickler verschieden. Ebenso entsteht das MMS nicht nur durch Abbildung des MMF sondern auch durch Beobachtung der Software. Das MMS muss auch zur Software redundant sein. Da die Software aber durch mehrere Entwickler entsteht, deren MMF verschieden sind, und sich die Strukturen des jeweiligen MMF über die Abbildung MMF $\rightarrow$ MMS im MMS wiederfinden stammt die Software von verschiedenen MMF ab! Sogar wenn nur ein Entwickler beteiligt ist so entsteht die Software über einen längeren Zeitraum, in dem sich das MMF mehrfach verändert. Dadurch wird es immer mehr oder weniger große Teile der Software geben, die nicht dem aktuellen MMS entsprechen, geschweige denn dem aktuellen MMF. Der Prozess der Softwareentwicklung ist ein ständiges Synchronisieren der redundanten Modelle MMF, MMS und des SWM.
Diese Betrachtungsweise ist vereinfachend. In Wirklichkeit gibt es nicht nur die Abbildungen MMF $\rightarrow$ MMS $\rightarrow$ SWM sondern weitere MM dazwischen (auf verschiedenen Abstraktionsebenen, z.B. GUI, Datenbank etc.), mit jeweils mehr oder weniger genau definierten Abbildungen dazwischen. Je klarer die Trennung der Modelle und je genauer die jeweiligen Abbildungen dazwischen definiert sind umso leichter fällt die Pflege der Redundanz und umso korrekter ist auch der entstehenden Code.

Redundanz und Kommunikation zwischen mentalen Modellen in verschiedenen Gedankenwelten

Arbeiten mehrere Entwickler an derselben Software, so ist Redundanz zwischen den MM der verschiedenen Entwickler unvermeidlich, ja um Kommunizieren zu können geradezu notwendig. Der Zweck der Kommunikation ist ja gerade die Erhöhung der Redundanz (Synchronisation) zwischen MM in verschiedenen GW. Redundanz in diesen MM ist geradezu eine Voraussetzung, die Kommunikation überhaupt erst ermöglicht. Entwickler die sehr verschiedene MM haben werden auch kaum

miteinander kommunizieren können, da sie zuerst ihre MM weitgehend synchronisieren müssen, bevor sie „eine gemeinsame Sprache“ finden. Dagegen werden sich Entwickler mit hochredundanten MM ohne viele Worte verstehen, da die MM schon weitgehend synchron sind und gar nicht viele Worte notwendig sind, um die MM zu synchronisieren.
Daher sind die besten Teams diejenigen bei denen die MM angeglichen sind. Dies betrifft sowohl die fachlichen MM, als auch die technischen, persönlichen etc. Techniken die die MM angleichen (wie z.B. Pair-Programming, gemeinsame Ziele, gleicher Ausbildungsstand usw.) tragen dazu bei die Kommunikation zu verbessern und gleichzeitig die Notwendigkeit zu Kommunizieren zu reduzieren.
Kommunikation zwischen Entwicklern findet auch über den Weg MMF1 → MMS1 → SWM → MMS2 → MMF2 statt.
Um den Synchronisationsaufwand gering zu halten sollten die schnell veränderlichen MM nicht redundant zu anderen MM sein.

Beispiel:
Die Probleme mit schnell veränderlichen Modellen kennt jeder, der schon einmal in einer Sitzung im Team ein Konzept ausgearbeitet hat. Hier werden viele Modelle schnell geändert. Wenn diese Situation nicht vermieden werden kann so sollten an solchen Sitzungen möglichst wenige Entwickler teilnehmen, deren MM möglichst synchron sein sollten. Die Entwickler sollten sich auch der Problematik der redundanten MM bewusst sein und den Wechsel der MM sich selbst und auch allen anderen klarmachen.

Es ist nützlich, die Redundanz gering zu halten. Dies gelingt dadurch, dass die Anzahl der Entwickler gering gehalten wird. Wenige gute Entwickler mit synchronen MM sind weitaus effizienter als viele mittelmäßige Entwickler die nicht synchrone MM haben. Wenn in einem Team Menschen mit sehr unterschiedlichen Ausbildungsständen zu finden sind so wird dies wahrscheinlich kein gutes Team werden wenn die MM der Mitglieder synchron sein müssen, da die Begriffe die die verschiedenen Teammitglieder verwenden gänzlich verschiedenen MM entstammen.
Dies ist natürlich ab Aufgaben einer gewissen Größenordnung nicht möglich. Eine geschickte Aufteilung der MM mit Hilfe von Information-Hiding und divide-and-conquer (Schnittstellen, Module, Kapselung) hilft die notwendige Redundanz zwischen den MM der Entwickler gering zu halten. Dabei wird der Umfang der gemeinsamen MM reduziert. Die Gemeinsamkeit bezieht sich nur noch auf die Schnittstellen. Schnittstellen haben die Tendenz, sich langsamer zu verändern als ihre Implementierung, so dass die Redundanz seltener zu pflegen ist. Beides reduziert den Kommunikationsaufwand. Kommuniziert werden muss nur über die redundanten MM.
Generell sollten sich schnell verändernde MM in einer GW bleiben. Ist Synchronisation mit MM in einer anderen GW notwendig, so sollten sie hinter kleineren, sich langsam verändernden MM verborgen werden (Fassade, Schnittstelle), für die der Kommunikationsaufwand geringer ist. Denn die Kommunikation zwischen MM innerhalb einer Gedankenwelt ist um Größenordnungen schneller als jene zwischen MM in unterschiedlichen Gedankenwelten.

Modelle, Redundanz und Unordnung

MM sind Modelle die in einer GW liegen. Modelle können im allgemeinen aber auch an anderen Stellen gespeichert sein. Eine Software ist im Grunde ein Modell, das Softwaremodell (SWM), das digital codiert ist und in einem Computersystem gespeichert ist. Die Dokumentation ist auch ein digitales Modell (DOCM), das in Dokumenten gespeichert ist. Kommunikation ist in diesem Sinne der Prozess der Pflege der Redundanz zwischen den Modellen. Codierung ist Kommunikation zwischen dem MMS des Entwicklers und dem SWM.
Je mehr untereinander redundante Modelle am Prozess beteiligt sind um so größer ist der Kommunikationsaufwand (hier Prozess der Pflege der Redundanz zwischen allen Modellen, auch zwischen MM und SWM, DOCM usw.). Der Entwicklungsaufwand ist dem Kommunikationsaufwand direkt proportional. Alle Techniken die den Kommunikationsaufwand verringern, verringern den Entwicklungsaufwand.
Wie wir gesehen haben besteht der Softwareentwicklungsprozess aus der Veränderung verschiedener dynamischer Modelle, die in Beziehung zueinander stehen und aus der Kommunikation die notwendig ist um die Modelle synchron zu halten. Die natürliche Tendenz der Modelle bei Veränderung ist die Unordnung zu vergrößern. Ohne weitere Maßnahmen nimmt die Unordnung des Systems bestehend aus den Modellen zu. Eine wichtige Aufgabe bei der Softwareentwicklung ist dieser Tendenz entgegenzuwirken und die Ordnung zu vergrößern.
Dies geschieht durch Kommunikation (Erhöhung der Redundanz) zwischen den Modellen und durch klare Strukturen (Verringerung der Redundanz) innerhalb eines Modells. Da der Aufwand der mit einem Modell verbunden ist von dessen Komplexität abhängt sollte ein Modell so wenig Informationen wie möglich enthalten (wenig komplex sein).
Ein Modell enthält zwei Arten von Informationen: Informationen die aus der Komplexität des modellierten Systems stammen und Informationen zufälliger Natur.
Um den Informationsgehalt eines Modells zu reduzieren kann man beide Arten von Informationen reduzieren.
Durch die Entfernung von Systeminformationen sinkt die Modellierfähigkeit und Genauigkeit des Modells. Das Modell beschreibt das modellierte System weniger gut. Trotzdem kann das Modell gut genug sein. Es bedarf einer Abwägung ob und welche Systeminformationen aus dem Modell entfernt werden sollen. Die Nachteile eines ungenaueren Modells können durch die Vorteile eines kleineren Modells mehr als aufgehoben werden. Oft werden Informationen vorsorglich in ein Modell aufgenommen. Dies widerspricht der Forderung nach einem minimalen System. Eine bessere Strategie ist, nur die wirklich benötigten Informationen aufzunehmen.
Durch die Entfernung von zufälligen Informationen nimmt die Ordnung des Systems zu und gleichzeitig wird das Modell kleiner. Diese Art von Informationsreduktion hat nur Vorteile und sollte wenn immer möglich erfolgen. Um zufällige Information zu entfernen muss zunächst die Redundanz des Modells erhöht werden. In einem zweiten Schritt wird dann die Redundanz entfernt um ein kleineres System zu erhalten.

Beispiel aus dem SWM:
Man kann einen Algorithmus auf viele verschiedene Arten codieren. Wenn dies geschieht, so ist das SWM nicht redundant, aber auch nicht minimal. Es besitzt durch

die verschiedenen „zufälligen“ Implementierungen des gleichen Algorithmus eine gewisse Unordnung. Wenn der Algorithmus nun an verschiedenen Stellen auf immer die gleiche Art implementiert wird so ist das SWM an diesen Stellen redundant, aber die Unordnung ist gesunken. Gleichzeitig ist das SWM aber nicht minimal, denn durch Ersetzen des gleichen Codes durch Referenzen (z.B. Funktionsaufrufe) auf eine Implementierung erhält man ein SWM das nicht redundant ist, und gleichzeitig weniger Informationen enthält als das ursprüngliche SWM, daher kommt es dem minimalen Modell näher und ist diesem vorzuziehen.
Weitere Beispiele zufälliger Information in einem SWM sind verschiedene Codierungen desselben Sachverhalts, beliebige Codeformatierung, beliebige Namenskonventionen, zufällige Verzeichnisstrukturen, ungenutzter Code (Variablen, Funktionen, Klassen).

Die Vorgehensweise ist immer gleich: identifiziere gleiche oder ähnliche Stellen im Modell, ersetze die ähnlichen durch gleiche Stellen, ersetze die gleichen Stellen durch eine Referenz. Dadurch erhält man ein Modell das weniger zufällige Informationen enthält und das gleichzeitig nicht redundant ist. Dies gilt für alle Modelle.
Weitere Beispiele sind Codegeneratoren, Programmierkonventionen, verschiedene Refactoring-Techniken, Design-Pattern, Namenskonventionen usw. Alle diese Techniken erhöhen die Ordnung des Modells indem zufällige Informationen aus dem System entfernt werden.
In einem zweiten Schritt kann die Redundanz aus dem Modell entfernt werden. So erhält man ein kleineres, weniger redundantes Modell, das dem ursprünglichen vorzuziehen ist. Oft bietet das bereinigte Modell durch seine veränderte Struktur überraschende, neue Möglichkeiten.
Durch weitere Schritte wird im Laufe der Entwicklung die Struktur des Systems wieder verschlechtert, indem zufällige Informationen aufgenommen werden, so dass wieder eine Bereinigung erfolgen muss.

Eigenschaften des Softwareentwicklungsprozesses

Entwicklungsprozess als Lernprozess

In der heutigen Zeit der stürmischen Entwicklung der Softwaretechnologie kann niemand von sich behaupten er kenne eine Technologie vollständig und kenne alle Varianten ein Problem zu lösen. Noch weniger kann jemand eine Technologie beurteilen die er nicht selbst in der Praxis angewendet hat. Da niemand die Zeit hat alles auszuprobieren um dann mit voller Kenntnis aller Vor- und Nachteile die optimale Lösung zu finden muss eine andere Strategie verwendet werden. Diese besteht darin ein System zu bauen das nicht redundant und minimal wie oben besprochen ist. Da meistens die Eigenschaften des Systems erst dann verstanden werden, wenn dieses schon da ist (erst dann kann ein realistisches MMS gebildet werden um die Eigenschaften der Software zu bestimmen, erst dann sind die MM der beteiligten Entwickler synchron genug um eine gemeinsame Sprache zu haben) muss

der SEP die unvollständigen MM der Entwickler berücksichtigen. Im besten Fall lernen alle an der Entwicklung Beteiligten bis zum Ende des SEP und verbessern ihre MM laufend. Oft ist dies aber nicht der Fall. Bricht der Lernprozess erst kurz vor Ende des Projekts zusammen so kann das Projekt durchaus noch zur Fertigstellung kommen, ohne ein großer Erfolg zu werden. Bricht der Lernprozess früh zusammen, so ist dies meist das Todesurteil.

Entwicklungsprozess als Suchprozess mit Backtracking

Da die Lösung (die Software) am Anfang des SEP nicht bekannt ist muss sie gesucht werden. Daher hat der SEP Eigenschaften eines Suchprozesses. Da jede getroffene Entscheidung Vor- und Nachteile hat, und es sich im Nachhinein herausstellen kann dass die Nachteile überwiegen kommt es in der Praxis oft vor dass mehrere Varianten ausprobiert werden müssen und ein bestehendes System immer wieder verändert wird bis seine Eigenschaften den Eigenschaften einer Lösung nahe kommen. Gelegentlich muss auf einen alten Stand zurückgesetzt werden wenn die Suche in eine Sackgasse geführt hat.
Der SEP erfordert ein bewegliches (leicht zu veränderndes) Software-System. Ein System ist umso beweglicher umso weniger komplex und umso weniger redundant es ist.
Des weiteren kann die Erfahrung der beteiligten Entwickler helfen, Irrwege bei der Suche zu vermeiden. Je besser die Entwickler, umso mehr können sie nach Gefühl (Erfahrung die nicht explizit formuliert werden kann) die verschiedenen Varianten beurteilen. Die besten Entwickler kennen meist eine gute Lösung (nicht unbedingt die beste, wie auch immer die Beste definiert sein mag), sie lenken die Suche in die richtige Richtung oder vermeiden sie ganz. Doch niemand kann in jedem Fall eine gute Lösung im vorhinein kennen und so ist der SEP auch ein Lern- und Suchprozess.
Die Suche und das Lernen widersprechen der aus ökonomischen Gründen berechtigten Forderung des Projektmanagements nach Determinismus und Planbarkeit. Den SEP aber per Beschluss als „wir bauen ein Haus nach Plan“ zu definieren und die Such- und Lerneigenschaften des SEP durch Druck, „endgültige Festlegungen“ und Kontrolle zu unterbinden führt unweigerlich zu einem aufgeblähten System, das die erste Lösung die durch Zufall gefunden wird implementiert und das die falschen MM der Entwickler am Anfang des SEP widerspiegelt. Diese Systeme sind unweigerlich strukturell kaputt, redundant und davon geprägt, dass eine Krücke die andere stützt, da eben der Such- und der Lerneffekt fehlen.
Die Suche kann auch durch die Unordnung und Komplexität der Software zum Erliegen kommen. Wenn niemand mehr den Code versteht, weil er zu komplex geworden ist, wenn niemand alten, unbenutzten Code aufgeräumt hat und dieser nicht von echtem Code zu unterscheiden ist, wenn kein System im Code zu erkennen ist so kann der Aufwand zur Änderung so groß werden dass niemand mehr eine wichtige Änderung machen kann oder will. Der Such- und Lernprozess kommt zum Erliegen, da das SWM nicht mehr verändert werden kann. Die MMS und die MMF können nicht mehr verändert werden, da sie in Beziehung zueinander und zum SWM stehen. Das Projekt kommt zum Stillstand.

Entwicklungsprozess als Kommunikationsprozesses

Lernen bedeutet verändern von MM, also Kommunikation. Wenige Dinge tun einem Projekt so schlecht wie fehlende Kommunikation. Fehlende Kommunikation verhindert das Lernen und sorgt dafür dass die MM nicht synchron sind. Folglich wird die Unordnung des Systems zunehmen, mit allen negativen Folgen.
Kommunikation muss dazu führen dass bestehende MM verändert werden, und nicht dazu dass weitere redundante Modelle aufgebaut werden (etwa Dokumente).
Der Kommunikationsaufwand begrenzt die effektiv einsetzbare Anzahl von Entwicklern. Hier gilt wieder: wenige gute Entwickler sind besser als viele mittelmäßige. Holen Sie sich die Besten ins Team, auch wenn sie teuer sind. Im Endeffekt wird die Software dadurch schneller und besser erstellt.

Entwicklungsprozess als Optimierungsprozesses

Wie schon gesagt benötigt ein *lernendes* Team, das eine Lösung *sucht* einen sauberen Umgang mit Redundanz und Komplexität. Die natürliche Tendenz zur Komplexität und Unordnung muss durch gezielte, andauernde Aktionen zur Strukturierung und Minimierung der Software bekämpft werden, ohne jedoch die Suche zu behindern. Auch wenn diese keine neue Funktionalität bringen so ermöglichen sie erst die Beweglichkeit des Systems auf seinem Weg zum Ziel. Wenn aus Termingründen nicht mehr optimiert wird so wird das System immer tiefer im Sumpf der Unordnung versinken. Lieber ein schlankes System mit wenig Funktionalität das funktioniert als ein fettes, dauernd „fast“ fertiges System. Auch hier gilt: je erfahrener und besser die Mitarbeiter, umso eher können sie entscheiden wann und wo optimiert werden muss. Es besteht die Gefahr, durch Reglementierung die Spontaneität der Entwickler einzuschränken. Diese ist jedoch für eine Lösungs*suche* unbedingt notwendig. Es reicht schon, wenn jeder Entwickler in seinem Code nach seinem Stil für Ordnung sorgt, wenn nur ein System dahintersteckt, das für andere schnell ersichtlich ist.

Mentale Modelle in der Praxis

Es folgt eine lose Sammlung von Tipps für die Praxis.

- MM gibt es und jeder verwendet sie, wenn auch unbewusst. Verwenden Sie sie bewusst.
- Machen Sie sich klar, dass es viele MM gibt. Ihr Gesprächspartner verwendet vielleicht gerade ein ganz anderes MM als Sie. Vielleicht besitzt er gar keine Entsprechung zu Ihrem MM. Klären Sie dies früh im Gespräch.
- Begriffe sind nur innerhalb eines MM definiert. Machen Sie sich und Ihren Gesprächspartner bewusst in welchem MM sie sich gerade befinden. Sollten Sie das MM z.B. zur Erkundung von Varianten wechseln so machen Sie es sich und den anderen bewusst. Am besten malen Sie für jedes verwendete MM ein Kästchen

auf ein Papier, mit den wichtigsten Begriffen und deren Beziehungen zueinander und markieren das MM dessen Begriffe Sie gerade verwenden.

- Bleiben Sie bei der Argumentation im selben MM. Wenn das MM die Argumentation nicht erlaubt, weil die notwendigen Begriffe darin nicht vorkommen müssen Sie das MM ändern oder erweitern. Brechen Sie Argumentationen die das MM verlassen ab und überlegen Sie erneut.
- Machen Sie sich immer wieder bewusst dass beim Wechsel des MM Begriffe mit demselben Namen eine neue Bedeutung bekommen. Schalten Sie im Kopf auf die neue Bedeutung um. Blenden Sie alle Begriffe die im aktuellen MM nicht definiert sind mental aus. Blenden Sie nur die Begriffe des aktuellen MM mit ihrer aktuellen Bedeutung ein. Zwingen Sie sich dazu, beim Wechsel des MM den Begriff des neuen MM zu verwenden, auch wenn es eine 1:1 Abbildung zwischen den Begriffen der verschiedenen MM gibt.
- Jeder Entwickler verwendet bewusst oder unbewusst mehrere MM. Jeder hat mindestens ein MM des Fachgebiets (MMF) das es zu modellieren gilt. Ebenso hat jeder ein MM der Software die erstellt wird (MMS). Ebenso hat der Entwickler meistens mehrere MM der Software so wie sie werden soll. Trennen Sie das MMF und die verschiedenen MMS. Für verschiedene in sich abgeschlossene Welten kann es weitere MM geben. So ist z.B. für die GUI einer Anwendung ein eigenes MM sinnvoll, ebenso für das Datenbankmodell.
- Bilden Sie das MMF im Kopf explizit auf das MMS ab. Machen Sie sich den Vorgang der Abbildung bewusst. Sie verwenden in jedem Modell Begriffe mit unterschiedlichen Eigenschaften. Durch die Abbildung findet eine Veränderung statt!
- Werden Sie sich klar dass die MM einem steten Wandel unterworfen sind. Das gilt auch für Ihre Kollegen! Deren MM ändern sich auch ständig. Synchronisieren Sie ihre MM, wenn erforderlich auch mit Ihren Kollegen.
- Wenden Sie eine Gemeinsamkeits-Metrik bei ihren Abbildungen an. Bleibt die Gemeinsamkeits-Metrik bei der Abbildung nicht erhalten so ist dies ein untrügliches Zeichen dafür, dass die Abbildung oder die Modelle verbessert werden können.
- Versuchen Sie die Abbildung laufend zu verbessern. Machen Sie ein Refactoring immer wenn Sie eine Stelle im MMS finden bei dem die Gemeinsamkeits-Metrik nicht erhalten bleibt. Sie werden die Struktur und Qualität des Programms verbessern.
- Definieren Sie die Gemeinsamkeits-Metrik bei redundanten Modellen zusammen mit Ihren Kollegen. Die dabei entstehende Diskussion führt zu einem besseren gegenseitigen Verständnis.
- Synchronisieren Sie Ihr MMS mit der aktuellen Software. Verstehen Sie wie diese funktioniert indem Sie ein oder mehrere MMS (für Teilbereiche) davon anlegen und aktuell halten.
- Verwechseln Sie nicht das MMS mit der aktuellen Software (SWM). Diese ist meistens verschieden von Ihrem MMS.
- Die Struktur des MMF sollte sich im MMS und dieses im SWM widerspiegeln.
- Managen Sie die Redundanz durch geschickte Aufteilung des Systems, und den Einsatz kleiner Teams mit Entwicklern die auf dem selben Wissensstand sind.

- Software ist dynamisch. Die Dynamik birgt die Gefahren der Unordnung und der zufälligen Informationen.
- Managen Sie den Informationsgehalt der Software indem Sie das System ordnen und es minimal halten.
- Halten sie nicht an Festlegungen fest, wenn die Dynamik des Systems diese obsolet gemacht hat.
- Nichts ist hinderlicher als veraltete Dokumentation. Sie erhöhen dadurch die Inflexibilität des Systems, indem sie Redundanz hinzufügen. Dokumentieren Sie hauptsächlich redundante, langsam veränderliche Teile der Software.
- Fragen Sie sich, was Sie da gerade dokumentieren. Dokumentation wird von Menschen erstellt. Als solche könne sie nur ein MM auf ein DOCM abbilden. Wenn das MM falsch oder veraltet ist, so ist auch die Dokumentation falsch oder veraltet. Das DOCM ist ein Abbild des MMS des Erstellers der Dokumentation, nicht ein Abbild des SWM.
- Unüberlegte Kommentare im Code sorgen für mehr Redundanz. Wer sich schon einmal von veralteten Kommentaren verwirren ließ weiß wovon ich spreche. Fügen Sie hauptsächlich Kommentare ein die nicht redundant zum Code sind (also zusätzliche Informationen enthalten).
- Kommentare sollten immer das MM des Lesers als Ziel haben, nicht einen Formalismus.
- Protokolle haben keine Alibifunktion sondern dienen der Kommunikation.
- MM befinden sich in der GW der Entwickler. Die MM können nicht oder nur sehr unvollkommen in ein anderes Modell (z.B. Dokumentation, Software) übertragen werden. Daher können Sie einen Entwickler mit guten MM nicht wirklich ersetzen. Ein guter Entwickler ist unersetzlich, da er seine MM mitnimmt wenn er geht.
- Ein guter Entwickler ist besser als zwei mittelmäßige, da er seine MM mit viel größerer Geschwindigkeit ändern kann als die zwei Entwickler, die mit den Problemen der Redundanz und Kommunikation zu kämpfen haben. Die mittelmäßigen Entwickler werden ihre MM auch nicht so klar ausarbeiten können wie der gute Entwickler (das macht ja gerade den Unterschied zwischen ihnen aus).