Tabellarischer Vergleich der Prozessmodelle für modellbasiertes Testen aus Managementsicht

Dominik Beulen, Baris Güldali, Michael Mlynarski
Software Quality Lab, Universität Paderborn
dbeulen@upb.de, {bguldali,mmlynarski}@s-lab.upb.de

1 Einleitung

Zahlreiche Forscher und Praktiker auf dem Bereich des Softwaretestens beschäftigen sich derzeit mit der Thematik des modellbasierten Testens (MBT) [1]. Als MBT wird die Generierung von Testfällen, Testorakeln und Testrahmen aus abstrakten Modellen (*Testmodellen*) verstanden [2].

Testmodelle spielen also eine zentrale Rolle beim MBT und müssen zuerst erstellt werden. Das bedarf oft zusätzlicher Kosten. Ein Testmanager, der MBT einsetzen möchte, muss nicht nur die Erstellung, sondern auch die Verwaltung von Testmodellen in der Kostenplanung berücksichtigen. Die Schulung des Testpersonals und Beschaffung von Werkzeugen stellen weitere Kostenpunkte dar. Die Zusatzkosten, die durch den Einsatz MBT entstehen, können Testmanager durchaus entmutigen. In Model-driven Development (MDD) werden Modelle bereits lange zur Unterstützung der Entwicklungsaktivitäten (wie z.B. Entwurf, Codegenerierung) erfolgreich eingesetzt. Ähnlich wie beim MDD, auch in MBT können Testmodelle sinnvoll und gewinnbringend eingesetzt werden [12].

Im MBT können die Eigenschaften, die Herkunft der Testmodelle und der Umgang mit ihnen je nach Projektkontext stark variieren. Diese Aspekte können direkten oder indirekten Einfluss auf die Kosten in Testen haben. In der MBT-Literatur wird zwischen unterschiedlichen Szenarien (hier *Prozessmodellen*) unterschieden, die diese Aspekte betrachten [2, 3]. Pretschner und Philips definieren in [2] folgende vier Prozessmodelle: (1) gemeinsames Modell für Entwicklung und Testen, (2) automatische Generierung des Testmodells aus existierendem System, (3) manuelle Erstellung des Testmodells, (4) separate Modelle für Entwicklung und Testen. Die Diskussion in [2] zeigt, dass diese Prozessmodelle sich in der Nützlichkeit und Aufwand unterscheiden, welche wichtigen Aspekte in Testmanagement darstellen.

Zusätzlich zu den oben aufgeführten Prozessmodellen konnten wir in der Literatur zwei weitere Prozessmodelle identifizieren, die sich bei der Erstellung und Nutzung von Testmodellen unterscheiden und andere Kosten/Nutzen Eigenschaften aufweisen: (5) Generierung von Testmodellen aus Testfällen [9, 10], (6) Transformation von Testmodellen [8, 11].

In dieser Publikation wollen wir zeigen, wie die unterschiedlichen Prozessmodelle aus Managementsicht miteinander verglichen werden können. Dafür stellen wir basierend auf die Literatur Vergleichskriterien [2-5] auf. Unser Ziel ist es, mit Hilfe von objektiven Kriterien eine Vergleichbarkeit von MBT-Prozessmodellen zu ermöglichen und den Testmanagern ein Hilfsmittel in die Hand zu geben, mit dem sie einschätzen können, mit welchen Kosten sie bei der Auswahl eines Prozessmodells rechnen können. Bei dem Vergleich adressieren wir auch den für die Prozessmodelle benötigten Reifegrad des Testprozesses nach Test Process Improvement (TPI) [7] und die benötigten Modellierungskenntnisse des Testteams, die mit Hilfe von Modeling Maturity Levels (MML) [8] gemessen werden können.

Als nächstes möchten wir die unterschiedlichen Prozessmodelle kurz skizzieren. Im Kapitel 3 stellen wir die Vergleichskriterien vor. Im Kapitel 4 zeigen wir den tabellarischen Vergleich.

2 Prozessmodelle des MBT

In [2] werden vier Szenarien für modellbasierten Testprozess definiert und ihre Unterschiede hauptsächlich bzgl. der Herkunft und des Nutzens der Testmodelle werden diskutiert. Die

Hauptaussage dieses Papiers ist es, dass Testen eine gewisse *Redundanz* zwischen dem Prüfling und der Testfällen zur Aufdeckung von Fehlern benötigt. Zusätzlich zu diesen Prozessmodellen haben wir in der MBT-Literatur weitere Prozessmodelle identifiziert, die in [2] nicht berücksichtigt worden sind. Im Folgenden charakterisieren wir diese Prozessmodelle, die wir im Kapitel 4 vergleichen. Bei der Charakterisierung geben wir aus Platzgründen nicht sehr viele Details und verweisen auf die Literatur [2, 9-11]. Die Abbildungen geben eine vereinfachte Visualisierung der Prozessmodelle basierend auf [2].

2.1 Gemeinsames Modell

Beim „Gemeinsames Modell“ wird ein Modell sowohl für die Code- als auch die Testfallgenerierung verwendet (Abbildung 1).

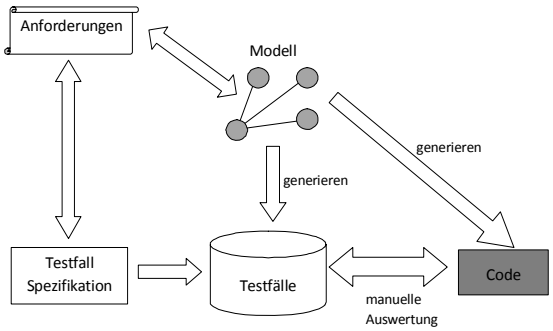

Abbildung 1: Gemeinsames Modell nach [2]

Bei diesem Prozessmodell werden die Modelle, die unter Berücksichtigung der Anforderungen erstellt werden, sowohl für Entwicklungs- und Testzwecke benutzt. Somit benötigt dieses Prozessmodell keine zusätzlichen Kosten für die Erstellung der Testmodelle.

Dieses Prozessmodell weist einen hohen Automatisierungsgrad, so dass Testfälle und Code automatisch aus demselben Modell generiert werden. Die Generierung der Testfälle berücksichtigt dabei eine Testfallspezifikation, die die Kriterien zur Testauswahl stellt. Durch die automatische Generierung des Codes und der Testfälle aus derselben Quelle fehlt jedoch die Redundanz zwischen dem Code und den Testfällen, die zur Aufdeckung von Fehlern benötigt wird [2]. In dieser Szenario wird also das Modell gegen sich selbst testet. Allerdings können durch dieses Vorgehen Fehler im Codegenerator oder Fehler in Zusammenhang mit den Annahmen über die Umgebung durch manuelle Auswertung der Testergebnisse entdeckt werden.

2.2 Generierung des Testmodells aus dem Code

Beim zweiten Prozessmodell wird das Testmodell (TM) durch Reverse-Engineering erstellt, wobei der Code oder die Systemlogs als Quelle für das Testmodell verwendet werden (Abbildung 2).

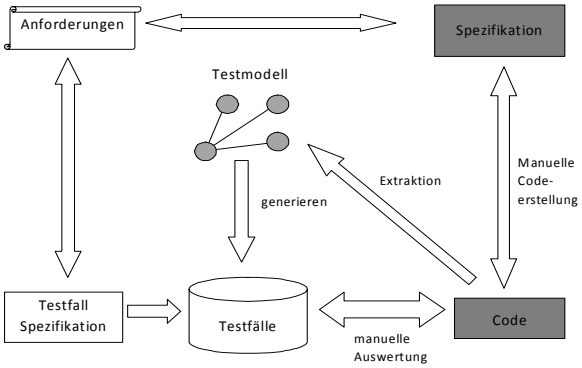

Abbildung 2: Generierung TM aus SUT nach [2]

Durch die automatisierte Wiederverwendung von Code entfällt die kreative Aktivität zur Erstellung der Testmodelle. Allerdings auch in diesem Szenario fehlt die für das Testen benötigte Redundanz, so dass der Code gegen sich selbst getestet wird.

Dieses Prozessmodell eignet sich gut für Regressionstests. Die späteren Versionen einer Software werden durch die Testfälle nach Konformität mit früheren Versionen getestet. Dabei kann das Verhalten einer früheren Version im Testmodell abgebildet werden. In diesem Fall kann das Testmodell als Orakel eingesetzt werden.

2.3 Manuelle Erstellung des Testmodells

Bei der manuellen Erstellung des Testmodells werden für die Entwicklung von Code und Testfällen verschiedene Quellen eingesetzt, die in Beziehung zueinander stehen (Abbildung 3).

Die Beziehung zwischen den Modellen kommt meist dadurch zustande, dass Entwicklungsmodelle in der Spezifikation für Testzwecke verändert werden und damit Testmodelle bilden.

Je weniger die Beziehung zwischen der Spezifikation und dem Testmodell ist, desto besser können Fehler im Code aufgedeckt werden. In

dem Fall, wo der Code sogar manuell aus der Spezifikation erstellt wird, kann das Testmodell als Orakel eingesetzt werden.

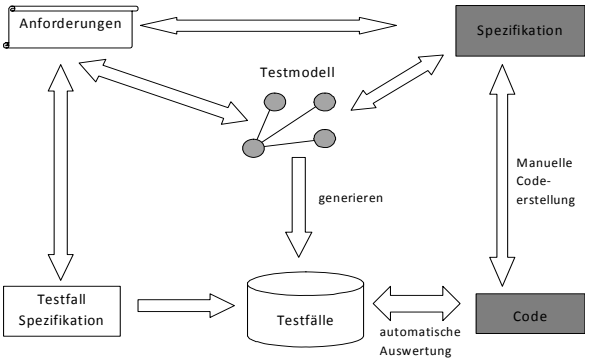

Abbildung 3: Manuelle Erstellung TM nach [2]

2.4 Separate Modelle

Der vierte Ansatz ist vergleichbar mit dem dritten Szenario. Allerdings werden bei diesem Ansatz zwei völlig voneinander gettrente Modelle verwendet, die idealerweise von zwei unabhängigen Abteilungen erstellt werden (Abb. 4).

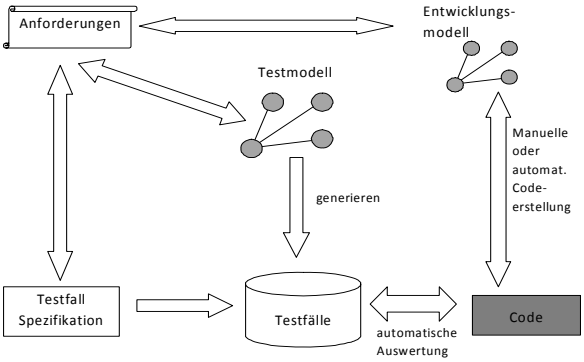

Abbildung 4: Separate Modelle nach [2]

Die beiden separaten Modelle können zur automatischen Generierung verwendet werden. Die Modellunabhängigkeit gewährleistet die gewünschte Redundanz zwischen Test und Entwicklung, so dass hier der Einsatz des Testmodells als Orakel sinnvoll ist. Die zweifache Modellerstellung ist zwar teurer als bei allen anderen Ansätzen, jedoch ist der weitere Entwicklungs- und Testprozess durch die Automatisierung deutlich effizienter.

2.5 Generierung aus den Testfällen

Dieses Prozessmodell ist vergleichbar mit dem zweiten Prozessmodell, in dem das Testmodell durch Reverse-Engineering erstellt wird (vgl. Abschnitt 2.2). Auch in diesem Ansatz wird das Testmodell durch Revers-Engineering gewonnen, dieses Mal aber aus alten Testfällen (Abbildung 5).

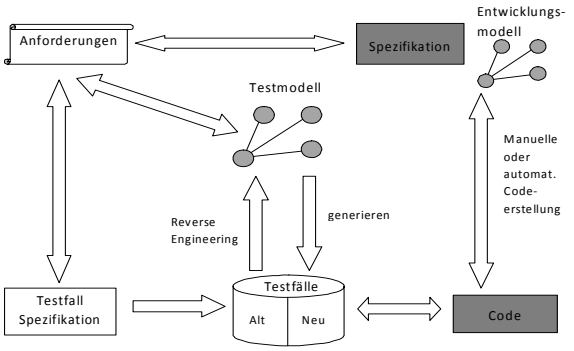

Abbildung 5: Generierung TM aus Testfällen

Bereits existierende Testfälle enthalten wertvolle Informationen über das Verhalten des Prüflings und seiner Umgebung inkl. Testeingaben und erwartetes Verhalten [9, 10]. Das aus diesen Informationen erstellte Testmodell kann zur Generierung neuer Testfällen oder zum Testen neueren Versionen des Prüflings verwendet werden.

Dieses Prozessmodell ist für Migrationsprojekte von klassischen Testprozessen zu modellbasiertem Testprozess sehr gut geeignet. Die bereits existierende Testfälle oder Testskripte können in Form von Testmodellen dokumentiert und wiederverwendet werden.

2.6 Transformation von Testmodellen

Bei diesem Ansatz geht es um die Erstellung des Testmodells durch Modelltransformationen. Dabei können unterschiedliche Modelle als Quelle dienen, z.B. alte Testmodelle oder Entwicklungsmodelle (Abbildung 6).

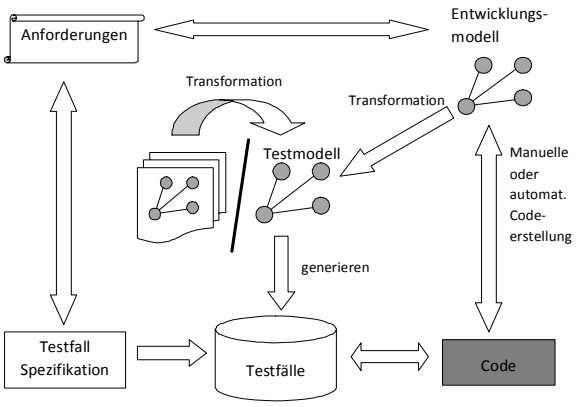

Abbildung 6: TM nach Transformation

Bei der Transformation können neue testrelevante Aspekte in das Testmodell einfließen, wodurch ein Mehrwert an Informationen gegenüber bestehenden Modellen entsteht.

Ein Beispiel für die Anwendung einer solchen Transformation ist die Mutation der Testmodelle. In [11] werden die Testmodelle, die zur Generierung der positiven Testfälle benutzt werden, mutiert, um daraus Negativtestfälle zu erzeugen. In einem anderen Beispiel [8] werden aus den Entwicklungsmodellen testrelevante Teile durch Modelltransformationen extrahiert. Dadurch wird eine gezielte Auswahl der Testfälle möglich.

3 Vergleichskriterien

Für den Vergleich der Prozessmodelle haben wir basierend auf der Literatur [2, 4, 5, 7, 13] mehrere Kriterien definiert, an die wir folgende Anforderungen stellen:

Relevanz zum Testmanagement: Kriterien sollen eine direkte oder indirekte Relevanz zu Managementsicht haben. Durch die Auswertung der Ansätze bzgl. der Kriterien soll ein Testmanager ein Gefühl für die potenziellen Aufwände und das Nutzen der Ansätze bekommen.

Objektivität: Die Prozessmodelle sollen bezüglich eines Kriteriums objektiv bewertbar sein, damit eine Reproduzierbarkeit und Nachvollziehbarkeit des Vergleichs gegeben ist.

Vergleichbarkeit: Die Bewertungen der Prozessmodelle bezüglich eines Kriteriums sollen auf einer aufsteigenden Werteskala (Beispiel: *keine*, *wenig* oder *hohe* Redundanz) verteilt werden, damit die einzelnen Bewertungen miteinander gegenüberstellt werden können.

Übersichtlichkeit an Anzahl: In der Literatur werden viele Dimensionen zur Charakterisierung insb. technischer Aspekte (wie z.B. Modellierungsparadigma, zu modellierende Subjekt) von MBT-Ansätzen definiert [3, 4, 6]. Diese Vielzahl von Dimensionen ist sicherlich hilfreich, wenn eine tiefgehende Analyse durchgeführt wird, die eher technisch motiviert ist. In unserem Vergleich möchten wir eine leichtgewichtige Heuristik anbieten, die mit wenigen Kriterien einen schnellen aber trotzdem aussagekräftigen Vergleich ermöglicht.

Folgende sechs Vergleichskriterien erfüllen oben vorgestellte Anforderungen und ermöglichen einen Vergleich der unterschiedlichen Prozessmodelle in MBT insbesondere aus Managementsicht:

Wiederverwendung: Durch die Erstellung und Wartung von Testmodellen entstehen zusätzliche Aufwände im Testprozess. Utting und Legeard diskutieren in [3] über die Vor- und Nachteile der Wiederverwendung von Entwicklungsmodellen für Testmodellierung. Einerseits würden durch die Wiedeverwendung keinen zusätzlichen Aufwand entstehen, andererseits würde die für das Testen benötigte Redundanz fehlen. Mit diesem Kriterium unterscheiden wir zwischen zwei Ausprägungen der Wiederverwendbarkeit der Entwicklungsmodelle: wiederverwendbar (✓) oder nicht wiederverwendbar (×).

Redundanz: Pretschner und Philips diskutieren in [2] über die Wichtigkeit der Redundanz in den Entwicklungs- und Testmodellen zur Aufdeckung von möglichen Fehlern. Je mehr Redundanz in den Modellen existiert, desto wahrscheinlicher ist es, die Unstimmigkeiten in der Spezifikation und den Testmodellen zu finden. Dieses Kriterium steht mit dem vorangegangenen Kriterium zu gewissermaßen in Beziehung. Je mehr Entwicklungsmodelle wiederverwendet werden, desto weniger ist die Redundanz. Für dieses Kriterium definieren wir drei Werte: *Keine* Redundanz (↓), *wenig* Redundanz (→), *hohe* Redundanz (↑).

Automatisierungslevel: Die einzelnen Aktivitäten für die Erstellung von Testmodellen, Generierung der Testfälle und Testauswertung weisen unterschiedlichen Bedarf an Automatisierung auf [4]. Für jede automatisierte Aktivität weisen wir den Ansätzen einen Punkt zu, so dass wenn alle drei Aktivitäten automatisiert werden, der Ansatz dann drei Punkte bekommt. Hier unterscheiden wir zwischen den folgenden möglichen Wertebelegungen: 3 (↑), 2 (→) und 1 (↓).

Reihenfolge zwischen Code- und Testmodellerstellung: Die inhaltliche Abhängigkeit der Test- und Entwicklungsmodelle bestimmt auch die zeitliche Beziehung des Erstellung dieser Modelle und des Codes. Bei den Prozessmodellen mit den Reverse-Engineering-Ansätzen müssen die Artefakte bereits existieren, bevor die Testmodelle aus ihnen gewonnen werden [5].

Hierfür unterscheiden wir zwischen drei Ausprägungen: Die Testmodelle entstehen in einem *späteren* Zeitpunkt im Entwicklungsprozess oder *parallel* dazu. Der Zeitpunkt kann auch *egal* sein, weil die Erstellung von Testmodellen vollkommen unabhängig geschieht.

Reifegrad des Testprozesses (TML): MBT benötigt Kenntnisse in systematischem und automatisiertem Vorgehen für Testen und ein fundiertes algorithmisches Wissen. D.h. für die Einführung von MBT muss der aktuelle Testprozess einen hohen Reifegrad besitzen. Der Reifegrad eines Testprozesses kann nach Testing Maturity Level (TML) bestimmt werden. Für TML wurde in Test Process Improvement (TPI) [7] Reifegradstufen von 0 bis 13 definiert. Je höher die Stufe, desto strukturierter ist der Testprozess. Mit diesem Kriterium möchten wir den Ansätzen einen Reifegrad zuweisen.

Reifegrad des Modellierungsprozesses (MML): wie mehrfach erwähnt benötigt MBT Modelle, die entweder durch den Entwickler oder Tester erstellt und gewartet werden müssen. Diese Aktivitäten erfordern Modellierungskenntnisse der Beteiligten und einen bestimmten Reifegrad des Entwurfsprozesses. Der Reifegrad eines Entwurfsprozesses kann mit Hilfe von Modeling Maturity Levels (MML) [13] bestimmt werden. Dieser bezieht sich auf die Modelle selbst und charakterisiert, wie detailliert die Modelle sind. Modelle niedrigeren Levels enthalten viele informelle textuelle Beschreibungen. Modelle höheren Levels werden formeller definiert und können für automatisierte Codegenerierung verwendet werden. Der Reifegrad wird in fünf Stufen gegliedert: (0) no specification, (1) textual, (2) text with diagrams, (3) models with text, (4) precise models, (5) models only [13].

4 Tabellarischer Vergleich

Tabelle 1 stellt den Vergleich der Prozessmodelle für MBT aus dem Abschnitt 2 in Zeilen bzgl. der Vergleichskriterien aus Abschnitt 3 in Spalten auf. Die Ausprägungen der Kriterien für einen Prozessmodell haben wir basierend auf die Beschreibungen in der Literatur [2, 4, 6, 7, 13] bestimmt, die wir als nächstes erläutern wollen, ohne zu sehr in Detail gehen zu wollen.

Bzgl. der Wiederverwendbarkeit bilden die Ansätze zwei Gruppen: Ansätze 1, 2, 5 und 6 sehen eine Wiederwendung der Entwicklungs- oder Testmodelle, der Testfälle oder des Codes vor. Bei den Ansätzen 3 und 4 werden die Testmodelle hauptsächlich neu explizit für Testzwecke erstellt.

Parallel zu Wiederverwendbarkeit unterscheidet sich die Redundanz zwischen den Testmodellen und den restlichen Artefakten. Eine hohe Redundanz hat einen wesentlichen Einfluss in Aufdeckung der Fehler im Code. Allerdings nur im Ansatz 4 ist eine hohe Redundanz zu erkennen, wobei bei den Ansätzen 1 und 2 fast keine Redundanz zu erkennen ist. Bei den Ansätzen 3, 5 und 6 entsteht durch die manuellen Aktivitäten oder durch die direkte Ergänzung der testrelevanten Aspekte eine gewisse Redundanz, die wir in der Tabelle mit *wenig* ausdrücken wollen.

Die Ausprägung der Automatisierung im gesamten Testprozess inkl. der Erstellung der Testmodelle, Testfälle und Testauswertung unterscheidet die Ansätze sehr stark. Alle Ansätze enthalten mindestens die automatisierte Testfallerstellung, so der Ansatz 1 mit einem Punkt. Die Ansätze 2, 3 und 4 bekommen einen mittleren Automatisierungslevel zugewiesen. Während bei Ansatz 2 die Testmodellerstellung automatisiert geschieht, können die Ansätze 3 und 4 die Testauswertung automatisieren. Bei den Ansätzen können alle drei Aktivitäten automatisiert werden, angenommen aus den alten Testfällen bzw. durch die Transformation erwartete Ergebnisse abgeleitet werden können.

Bei dem Kriterium wird die zeitliche Reihenfolge der Erstellung der Testmodelle und der Entwicklungsmodelle betrachtet. Bei den Ansätzen müssen 1 und 3 die Modelle in Parallel erstellt werden müssen, da sie in Beziehung zueinander stehen. Bei dem Ansatz 2 kann die Erstellung der Testmodelle erst nach der Erstellung des Codes erfolgen, d.h. zu einem späteren Zeitpunkt im Vergleich mit der Erstellung der Entwicklungsmodelle. Bei den Ansätzen 4 und 5 läuft die Erstellung von Testmodellen vollkommen unabhängig von Entwicklungsaktivitäten. Bei dem Ansatz 6 hängt die Bewertung davon ab, ob bei der Transformation Entwicklungsmodelle als Quelle eingesetzt werden. Wenn ja, müssten die Testmodelle auch später erstellt werden.

Tabelle 1: Vergleich der MBT-Prozessmodelle

	Wieder-verwendung	Redundanz	Automatisie-rungslevel	Reihenfolge	TML	MML
1) Gemeinsames Modell	✓	↓	↓	parallel	7	5
2) Generierung TM aus SUT	✓	↓	→	später	7	1
3) Manuelle Erstellung TM	×	→	→	parallel	7	3
4) Separate Modelle	×	↑	→	egal	7	4/5
5) Generierung TM aus Testfällen	✓	→	↑	egal	7	1
6) TM nach Transformation	✓	→	↑	später bzw. egal	7	5

Utting und Legeard erklären in [3], dass MBT erst dann einsetzbar ist, wenn ein stabiler Testprozess vorhanden ist. Dort wird die TML Stufe 7 als mindeste Stufe angesehen, die wir in unserem Vergleich in der Form übernommen haben. Allerdings wir vermuten, dass die adressierten Ansätze sich insbesondere bzgl. der Key Areas in TPI „test strategy“, „moment of involvment“, „test automation“, „communication“ und „testware management“ stark unterscheiden werden.

Des Weiteren kann man sich bei dem Vergleich der Ansätze an dem MML orientieren. Utting und Legeard empfehlen in [3] MBT ab Level 3 einzusetzen. Wir sehen für die einzelnen Ansätze unterschiedliche Ausprägungen bzgl. MML. Bei dem Ansatz 1 ist eine automatisierte Code- und Testfallgenerierung vorgesehen, was einen hohen MML benötigt (Level 5). Auch die Ansätze 4 und 6 benötigen einen hohn MML. Bei den Ansätzen 2 und 5 werden gar keine Modelle manuell erstellt, sondern durch Reverse-Engineering aus dem Code oder Testfälle gewonnen (Level 1). Bei dem Ansatz 3 benötigt man für die manuelle Modellierung der Testmodelle einen stabilen Kenntnisstand (Level 3).

5 Zusammenfassung

In diesem Papier haben wir einen Vorschlag für einen Vergleichsschema für MBT-Prozessmodelle gemacht, der diese Prozessmodelle primär aus Managementsicht betrachtet. Durch dieses Schema soll einem Testmanager geholfen werden,

- die durch MBT entstehenden Aufwände und zu erfahren und sich für die Kosten/Nutzen der Ansätze zu sensibilisieren,
- die organisatorischen Aspekte (zeitliche Reihenfolge, Kommunikation, Wiederverwendung) zu verstehen,
- die Eignung von MBT für eigenen Kontext bzgl. des Reifegrades des Testprozesses und der Modellierungskenntnisse des Teams zu evaluieren.

Primäre Beiträge dieser Arbeit sind die Identifizierung weiterer Prozessmodelle für MBT und die Erweiterung der Prozessmodelle in [2], die Sichtung und Aufstellung von Vergleichskriterien für Management basierend mehrerer Literaturquellen [2, 4, 6, 7, 13] und die Bewertung der Prozessmodelle anhand dieser Kriterien.

In der Zukunft möchten wir unseren Vergleich nach Testprozessreifegrad TML insbesondere unter Betrachtung der identifizierten TPI Key Areas verfeinern.

6 Literatur

[1] Bertolino, A.: Software Testing Research: Achievements, Challenges, Dreams *FOSE '07: 2007 Future of Software Engineering, IEEE Computer Society*, **2007**, 85-103

[2] Pretschner, A., Philips, J.: Methodological Issues in Model-Based Testing. In M. Broy, et.al. (Eds.), Model-Based Testing of Reactive Systems, LNCS no. 3472, Springer-Verlag, 2005, pp. 281-291.

[3] Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.

[4] Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing. Technical report 04/2006, Department of Computer Science, The University of Waikato (New Zea-land).

[5] Beulen, D.: Seminararbeit : Ansätze des Modellbasierten Testens, Universität Paderborn, 2009

[6] A. C. Dias Neto, R. Subramanyan, M. Vieira, G. H. Travassos: A Survey on Model-based Testing Approaches: A Systematic Review. In Proc. Of WEASELTech'07, November 5, 2007, Atlanta Georgia, USA

[7] Pol, M.; Koomen, T. & Spillner, A. Management und Optimierung des Testprozesses, *dpunkt.verlag*, 2002

[8] Mlynarski, M., Güldali, B., Späth. M., Engels, G.: From Design Models to Test Models by Means of Test Ideas. In Proc. of Modevva 2009 at MODELS 2009 (to be published)

[9] A. Jääskeläinen, A. Kervinen, M. Katara, A. Valmari, and H. Virtanen. Synthesizing Test Models from Test Cases. In Proc. of the Haifa Verification Conference 2008, IBM Haifa Labs, October 2008. Number 5394 in Lecture Notes in Computer Science, pages 179-193. Springer, 2009.

[10] Zeiss, B., Ulrich, A., Grabowski, J.: Constructing Test Behavior Models using simulated system answers fort he analysis of test behavior anomalies. In GI Jahrestagung (1) 2008: 177-182

[11] Chen Z., Hollmann, A.: Positive and negative testing with mutation-driven model checking. In GI Jahrestagung (1) 2008: 187-192

[12] Pretschner, A., Prenninger, W., Wagner, S., Kühnel, C., Baumgartner, M., Sostawa, B., Zölch, R. und Stauner, T.: One evaluation of model-based testing and its automation, ICSE '05: Proceedings of the 27th international conference on Software engineering, ACM, 2005, 392-401

[13] A. Kleppe, J. Warmer, W. Bast. MDA Explained: The Model Driven Architecture(TM): Practice and Promise, *Addison-Wesley Professional*, 2003