Rational Quality Requirements for Medical Software

Barbara Paech
Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326
D-69118 Heidelberg
Germany
paech@informatik.uni-heidelberg.de

Thomas Wetter
Institute of Medical Biometry and Medical Informatics
University of Heidelberg
Im Neuenheimer Feld 305
D-69120 Heidelberg
Germany
thomas.wetter@urz.uni-heidelberg.del

ABSTRACT
In this paper we discuss the challenges of software quality for medical software and present some ideas for improving medical software quality requirements through software engineering methods. We apply the quality requirements engineering method MOQARE to elicit specific quality requirements for an imaginary drug advisory system and report our lessons learned.

Categories and Subject Descriptors
D2.1 Requirements / Specification

General Terms
Management, Documentation, Design, Economics, Human Factors, Standardization, Languages

Keywords
Quality Requirements, medical software quality challenges, medical business processes, risk analysis

1. INTRODUCTION
Software quality is a difficult issue in general. Very often quality issues are not adequately dealt with during software development, often due to cost and time restrictions but also because it is difficult to derive a coherent view of quality between all stakeholders[6]. The latter is particularly true for the development of medical software, as the business processes involved themselves are subject to strong quality constraints and as there are typically many different stakeholders involved.

In this paper we want to explore the potential of software engineering methods for medical software. We have not researched in detail for other software engineering methods applications to medical software such as e.g. [1], and our intent is not to present a mature approach. Instead we want to explore with an imaginary example the usefulness of one particular software engineering approach which enables the rational treatment of quality requirements for medical software.

In section 2 we describe typical quality challenges for medical software. Then we sketch an application of the quality requirements engineering method MOQARE for the design of an antibiotic drug advisory system. In section 4 we discuss our lessons learned. The paper closes with an outlook on future research.

2. SOFTWARE QUALITY CHALLENGES IN MEDICAL SOFTWARE
In the sequel we will use the term *medical software* for software used in the variety of settings where health care professionals and paramedics treat patients and as part of that administer and document diagnostic and therapeutic procedures, take notes, seek advice, communicate, and look up data from previous treatments. Software for patients rather than for health care professionals poses a whole new set of challenges that will not be included here.

The purpose of this section is to capture typical challenges for the quality of medical software. Medical care as an industry and clinical users as stakeholders set up a peculiar workplace, some of whose characteristics will subsequently be outlined, with primary focus on physician users. Other user groups such as nursing and hospital administration will be mentioned where appropriate and add to the size of the problem.

Physicians' work can be characterized through complexity, autonomy, and innovation. Entailed from these primary characteristics are secondary ones such as high variability and high rate of change

Complexity relates to the complexity of the medical domain. ICD – widely used in the 1997 version ICD-10 – is an International Classification of Diseases and Related Health Problems which lists appr. 12,000 different conditions. A patient is typically characterized through a one or a small two digit number of ICD-codes. Assuming that a patient has only three conditions yields $12{,}000^3 \sim 2*10^{12}$ combinations as a conservative lower estimate.

Complexity of the domain is reflected through complexity of education and care. Depending on the national designs of medical education there may be more than hundred specialties and sub specialties of medical physicians. A patient with a complex condition is typically seen by several sub specialties which need to share data and processes. Furthermore each sub specialty contributes proprietary data and measuring modalities.

For a number of reasons physicians have high **autonomy** in doing their work. The Declaration of Helsinki holds up the basically unchallenged claim of Freedom of Treatment: as long as they regard it appropriate in appreciation of special risks or opportunities of a case physicians may proceed as they prefer and may disregard any existing standard procedures or protocols. This can be justified through both an outstanding level of education and work ethos on the side of the physicians and the lasting incompleteness of the knowledge: it is out of consideration to have scientific evidence about the best approach for all > $2*10^{12}$combinations of conditions in any foreseeable future. As a

result, physicians can and do deviate from established procedures in an unforeseeable way.

Many of the most renowned physician practitioners are also outstanding **innovator**s. In their role as researchers they push the limits of the sub specialties and develop and test new procedures. Their first of a kind approaches require and produce new types of data and impose new kinds of constraints on others working in the same hospital and on software used and data shared.

All arguments so far come down to well justified, high **variability** of processes in medicine. While there may be industries where IT is used as a means and software can be designed to the end to reduce variability, medical software rather has to be designed to comply with the variability.

While all factors above are inherent to the nature of the problem – health and how to treat disease – the following is self made organizational and yet unavoidable for the foreseeable future: the **diversity** in organizing health care across countries and states, sectors and professions. This diversity entails differences concerning which data are required to claim reimbursement for services offered, which resources can be booked right away or have to be applied for etc. Assuming that a valid software engineering approach has been found that masters all other challenges in one organizational setting lots of new work is likely to begin for any other organizational settings [5].

Diversity will not be pursued in depth, because it is worth an article of its own. In the context of this paper it is relevant because of its impact on **change**. The above mentioned inherent challenges entail already a very high rate of change. This is considerably increased through variability simply because the more factors there are the higher is the rate of change coming down on the software [7].

For the rest of the paper we concentrate on complexity, autonomy and change.

3. QUALITY REQUIREMENTS ENGINEERING WITH MOQARE

In [4] we introduced the method MOQARE for eliciting and prioritizing quality requirements. The main idea is to specify quality as countermeasures to quality defects where quality defects are characterized by development or operation misuses which may cause them. This idea is typically applied in the specification of reliability, but works equally well also for other quality attributes. It helps to elicit and rationalize specific requirements for typically vague quality goals.

A major outcome of an MOQARE analysis is the so-called Misuse Tree which shows how different quality goals, misuses and countermeasures are related to each other and to the business goals. This way, dependencies between business and system requirements are made explicit and can be exploited during prioritization. Furthermore the benefit of a quality requirement (countermeasure) can be derived from the risks of the misuse cases it prohibits, mitigates or detects.

In the following we sketch a MOQARE analysis for an imaginary medical software system, the antibiotics advisory system (AntiBiAS) for the paradigmatic example of advising physicians in prescribing antibiotics and ordering them through the hospital pharmacy. It gives a systematic account of typical quality issues (as discussed in section 2) and quality requirements which try to solve them.

Some of the threats to quality processes may be related to partially competing goals which legitimately coexist in hospitals, and which can be served in a balanced way if appropriate feedback loops are present and users are fully aware of the present state of the whole feedback loop. Given the above arguments this is very hard to achieve and partial views are the rule. The following scenario is owed to one such set of partial views of the development of AntiBiAS.

In the rest of this section we explain how quality requirements for this system are derived from general business and quality goals. The corresponding Misuse Tree is depicted in Figure 1.

A hospital has typically among others the following business goals

- Good management of bacterial infections (medical).
- Low cost (financial)

Clearly, these goals overlap: If epidemics of nosocomial (hospital acquired) infections are avoided, mass consumption of antibiotics and the resulting cost is avoided But they also compete: If antibiotics are withheld, especially in a situation where an epidemic is developing and is noticed or reacted to too late, money is saved at first but higher consumption/payment may be due later to fight an epidemic. They also compete in a more subtle and somewhat paradoxical way. If broadband antibiotics are routinely administered without specific indication, resistant bacteria are the only that will survive and will form populations for which no more cure exists. In other words: antibiotics utilization is a continuous trade off between fighting existing or imminent threats and trusting the patient's immune system when there are no threats. So these business goals can be refined into the following three, which are considered in the following:

- BG1: Low antibiotics consumption (to reduce cost, but also to avoid conditions where resistant bacteriae will selectively survive)
- BG2: Low number of nosocomial infections; This is core to the medical goal
- BG3: Low cost of antibiotics

For BG2 other organizational and hygiene measures obviously play a role. For our consideration, however, to achieve the business goals the hospital establishes AntiBiAS combined with an order entry system which initiates drug delivery and records ordering in the electronic patient archive.. The quality goal to be achieved through the software that serves all three business goals is

- QG0: optimized antibiotics ordering

Thus

- QG1: accuracy of the advisory and order function

is critical for business success. That means on the one hand that

- QG11: the information recorded in AntiBiAS has to be accurate and on the other hand that
- QG12: the knowledge base (data and rules) of AntiBiAS need to be accurate.

However, AntiBiAS is not a fully automatic system. Physicians are supposed to consult the data and advice offered and to release an order through the system. Unless they are convinced that they have knowledge and evidence beyond the scope of the system and then their autonomy allows and even encourages them to deviate, they should confirm and authorize

the order suggested by the system. Therefore, a **nontechnical** quality goal is that

- QG2: physicians rationally handle system suggestions.

Only if the advice is normally correct and correct advice is normally adhered to, can optimal performance be achieved.

The main idea of MOQARE is to look at quality risks triggered by misuses in order to better understand what quality means to the stakeholders. In order to come up with the misuses it is important to understand the context of the system that means the business processes and rules and the typical users, As described in section 2 this is characterized by high complexity, autonomy and change.

Epidemics are variations over time of the spread and threat through bacteria residing in the hospital. An epidemic may be detected

- by chance, through individuals with insight and spontaneously granted access to data
- indirectly through increasing ordering of antibiotics
- directly though increased number of positive microbiology lab samples

Obviously, quality assurance cannot rely on chance or insight. For the other two indicators, however, appropriate measures can be taken. For this purpose the system needs to record and analyze the orders in both quantity and type as well as lab information. Therefore, QG11 is at risk, if

- QD1: the current orders are not recorded completely or
- QD2: the recorded evidence on epidemic developments is not accurate. E.g. Nosocomial infections/resistant agents are not noticed or not incorporated in the system.

While the former two deficits relate to data required to monitor epidemics, the next two quality risks relate to knowledge required to generate the appropriate advice, irrespective of whether there is or isn't an epidemic. QG12 is at risk, if

- QD3: AntiBiAS advises unnecessarily expensive orders or
- QD4: AntiBiAS advises medically inappropriate orders.

The main defect compromising QG2 is just its opposite

- QD5: physician does not rationally handle advice.

The next step in MOQARE is to analyze which actions (misuses) can trigger the defects. Here one also hast to consider vulnerabilities, that means properties of the system or the environment which make the misuse more likely. If the vulnerability can be avoided, the countermeasure should also address the vulnerability (e.g. insufficient education), but often the vulnerability is just inherent in the system or the environment. Then the countermeasure has to address the misuse in the presence of the vulnerability (e.g. autonomy) or prevent side effects of the misuse. In Figure 1 we highlight vulnerabilities. In particular, we concentrate on vulnerabilities which correspond to the challenges identified in section 2.

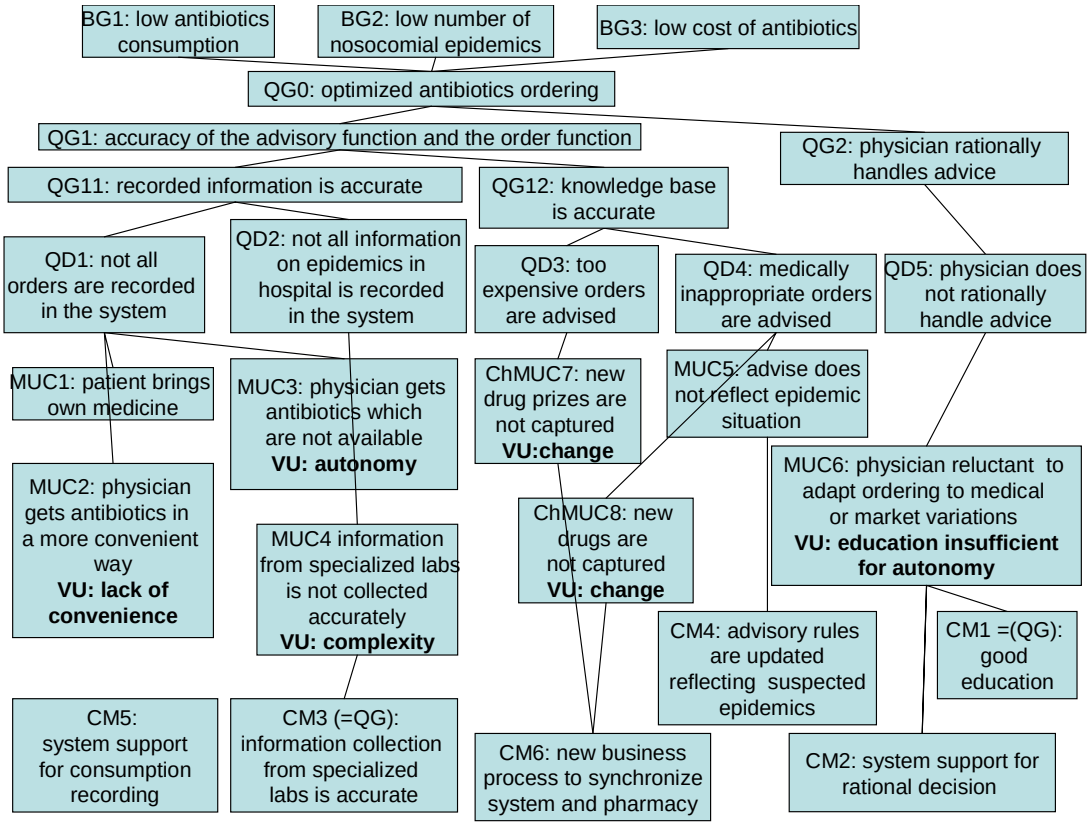

Figure 1: Misuse Tree AntiBiAS

For the analysis of the misuse cases, first we concentrate on QD1: There are several actions which prevent the system from recording orders:

- MUC1: antibiotics are consumed which have never been ordered through the system, e.g. patients bring their own medicine.
- MUC2 comprises all physician behaviors to get in a more convenient way an antibiotic which they could principally order through the system from the pharmacy. Thus, lack of convenience is a vulnerability. It can be a non-intuitive interface of the software, the distant or rarely available access points to computers, authentication and signature requirements. To work around the inconvenience physicians may make a phone call to the hospital pharmacy, order from a local pharmacy or have some antibiotics on stock on the ward.
- MUC3 comprises all physician behaviors to get an antibiotic which they cannot order from the hospital pharmacy but regard necessary for the patient they are treating. The mechanisms can be the same as for MUC2 and hence the effect on quality is the same. The motivation (and thus the vulnerability), however, is lack of convenience in MUC2 and autonomy in MUC3.

At least as important an indicator for epidemics as antibiotics ordering is the detection of bacteria in samples sent to microbiology labs. Therefore, QD2 may materialize through

- MUC4: typically there is not only one central lab, but many small labs for specialized units co-exist. Often information from the specialized labs cannot be collected accurately because the respective Laboratory Information Systems communicate results in proprietary formats rather than adhering to existing standards. This is a typical example of the complexity vulnerability where defects in one system (collecting lab information) jeopardize other systems.

QD3 means that medically appropriate antibiotics are suggested but that a cheaper choice exists in the hospital pharmacy. QD4 means that the expected patient outcome is better with a medically different antibiotic from the hospital pharmacy. Both deficits point to the fact that, in order to build AntBiAS, the knowledge required to advise the most economic of the medically appropriate antibiotics must already be present. It is assumed here that this knowledge is available. Then the software engineering challenges remain to capture and present this knowledge appropriately. The latter includes that, when changes become necessary, they can be incorporated smoothly.

Three types of changes are most prominent: prices for antibiotics change, new antibiotics are developed that better serve some patients' conditions; these both happen in the uncontrollable environment. Furthermore epidemics develop and disappear again inside the hospital.

Changes in the environment constitute risks for the system. This is covered in MOQARE by collecting so called Change Misuse Cases (abbreviated ChMUC) which characterize the expected changes. QD3 and QD4 then materialize due to related change misuses that are, however, different in one important aspect:

- ChMUC7: Ignoring changed prices means that hospital profitability goes down.
- ChMUC8: Ignoring new superior antibiotics means that opportunities for patient cures are not taken.

Besides the change misuses the internally caused main problem for QD4 is

- MUC5: the advice given does not reflect the current epidemic situation in the hospital. AntiBiAS' advice is too "aggressive" (suggesting antibiotics in too many cases) with no epidemic present or too conservative or unspecific (not suggesting antibiotics or suggesting general ones) while an epidemic is present or even a specific agent is known as cause.

While all aspects so far originate from system properties and behaviors - with physicians contributing logistics misuses such as MUC2 and MUC3 - QD5 originates from physicians taking their autonomy too far: They bluntly disregard system advice rather than at least considering whether it may add value to a decision.

Because of the autonomy, physicians will always be allowed to order antibiotics which are not recommended by the advisory system or to not order recommended antibiotics. Thus, QD5 materializes when both habits and irrational and conservative attitude guide behavior towards the following misuse:

- MUC6: Physicians tend to prescribe what they know and are used to and try to ignore that better opportunities exist or emerge that are worth taking into consideration. The vulnerability here is insufficient education in the light of the autonomy.

The main goal of this misuse analysis is to identify adequate countermeasures. These can help to detect, mitigate or inhibit the misuse cases. For the latter it is important to know the vulnerabilities which enable the misuse case. One important vulnerability for medical software is the autonomy of the physicians (see MUC3). But as argued in section 2 this is inherent in the medical domain. For requirements engineering the challenge is here to find a balance between the autonomy requirement and the other quality requirements. Another vulnerability, e.g. important for MUC6 is the education of the users, in this case the physicians. This gives rise to non technical and technical countermeasures. So for MUC6 we have

- CM1: good education of the user. Clearly this has to be refined, but is out of the scope of this paper.

A further countermeasure which supports the education is system support for rationale decision making, when unsolicited antibiotics are about to be ordered. This includes

CM2.1: passive help function ("Infobutton" [2]) that informs upon user's request about the system's logic to recommend or deny an antibiotic, and

CM2.2: active antibiotics assistant logics and interface design which discourage antibiotics unless clearly indicated (low sensitivity, high specificity)

Also the countermeasure to MUC4 is a new quality requirement which needs to be refined. To mitigate the inaccurate information collection one has to analyze in detail the information collection including e.g.

CM3.1: An internal technical countermeasure is to analyze the data delivered through the laboratory information system and to translate into the conventions required for epidemics detection.

CM3.2: An external managerial countermeasure is to force vendors of so far noncompliant laboratory information systems to change the data formats they deliver.

The decision which of the two countermeasures to prefer depends on factors such as power of hospital and vendor, skills available in the hospital, time, dependency on the specific laboratory information system.

The epidemical advise (MUC5) depends strongly on the data and criteria used to detect the epidemic. As mentioned before we assume that this knowledge is available in the hospital. So it remains to ensure that the advisory rules are updated when new evidence on epidemics is present

- CM4: the advisory rules are updated according to epidemic evidence.

To avoid incomplete data on antibiotics consumption through MUC1 and MUC3 (and possibly MUC2) the following countermeasure suggests itself:

CM5: Add function to AntiBiAS which records antibiotics that are indeed given to patients although they were not ordered through the system.Its relationship to the three misuses is not shown in Figure 1 for sake of readability.

The final countermeasure mitigates ChMUC7 and ChMUC8.

- CM6: A business process is established initiated by pharmacy and administration: They communicate the plan to introduce new drugs or changing drugs through apparently equivalent cheaper ones to software maintenance. Software maintenance accommodates the advice function to these changes and approves the new purchases to pharmacy when the respective software release of the advisor is ready for launch. In section 4 we will briefly discuss how this and other countermeasures can be regarded from a broader perspective of professional and enterprise culture.

4. LESSONS LEARNED

When applying MOQARE to the example system we wanted to evaluate how feasible the application of a rigorous quality analysis is for medical software given the challenges.

During our analysis we made the following experiences:

- It was possible to rationally derive the countermeasures from the business goals and the main software quality goal. We needed several iterations to arrive at the final picture, but this was felt as an advantage. *The misuse tree forced us to very carefully think about what to make explicit or not.* For example, at first we tried to deal with QG2 implicitly as we wanted to concentrate on the system quality. But when discussing MUC6 (which first came up when thinking about how the system can achieve BG3) we realized that it is important to distinguish good uses of autonomy (as in MUC3) of bad ones. As another example, for QD1 we first had collected all the different ways how physicians can bypass the order recording. Given the logistics in a hospital there are many ways to do this. But by looking at the misuses it became clear that there are different reasons for the bypassing and that actually the reasons are more important to identify adequate countermeasures.
- It was possible to adequately highlight the challenges discussed in section 2 in our analysis. The misuses helped to identify how these challenges in particular affect the software under consideration. This is made explicit through the vulnerabilities.
- Quality considerations impact the system scope: An interesting combination arose wrt. MUC1 and MUC3: neither can we prevent that patients bring their own medicine nor can we preclude physician autonomy. Therefore, a countermeasure cannot be established that prevents the misuses but only one which avoids its harmful side effects (CM5). Interestingly, this compromises the basic intention of the system to be a closed antibiotics advisory and ordering system with documentation of consumption being a side effect of ordering. CM5 modifies this closed system perspective for the better good of at least capturing the data from processes that don't fit in the closed metaphor.
- Countermeasures often have implications on professional and enterprise culture. To keep the Misuse Tree simple for the paper we have not discussed the implications in detail. For example, CM1 und CM6: CM1 does have medical contents but the major education goal is the change of attitude from experience based and authority supported towards rational decision making. CM6 comprises the synchronization between purchasing and software maintenance. This would entail to establish in addition managerial processes that assess every new drug and every new price as to its financial and medical effects in the hospital versus effort to have it incorporated in the AntiBiAS. A consensus is then achieved as to either coordinated implementation or to agreement to ignore a change. Hence an enterprise culture of local optimizers that disregard side effects imposed on other enterprise functions gives ways to a culture of communication and seeking for a global optimum.
- Combining Misuse analysis and architecture may help to suggest countermeasure that would not come to mind easily with misuse analysis alone. Through his familiarity with an architectural concept specifically developed for medical information systems one of the authors came up with the following farther ranging countermeasure for MUC2, i.e. convenience driven phone calls instead of using AntiBiAS. The corresponding architectural notion is uncontrolled redundancy: two ways to execute a function for which only one way is necessary. This redundancy can be modified. Either can the convenience factor be reduced through a voice recognition enabled automated ordering system which just records orders but takes the physician through an endless dialogue. Or AntiBiAS could itself be voice enbabled such that the phone call leads to the same advice and to an equally recorded AntiBiAS order. This would coincide with a countermeasure to unavailability of access points.
- A MOQARE analysis is time consuming and for a complex system the Misuse Tree gets too big. From the example it is difficult to estimate whether the time needed is worth the effort. There are several ways how the time can be made more worthwhile: Clearly, for a real system requirements management tools are needed to capture the results of the MOQARE analysis. Wrt. the time constraints it seems important to restrict a detailed analysis to certain difficult quality trade-offs. Furthermore often parts of the analysis

can be re-used for similar systems. This re-use decreases the time needed for the specific system.

The main advantages of rationale management in general are support for communication, transparent decision making and software evolution [3]. Given the variability and the complexity of the domain as well as the constant innovation and change such support seems urgently needed. CM6 in its different levels of sophistication is an example for initializing a communication about direct and indirect effects of replacing certain antibiotics through cheaper ones. Locally optimal patterns of behavior may give way for a hospital wide strategy that leads towards a global optimum.

Altogether, the effort needed is a general argument against software engineering methods. As sketched in section 2 so far the medical domain has put more emphasis on managerial actions than a system quality culture. As there is evidence from other quality critical domains like aerospace or automotive, it seems worthwhile to explore the benefits of a system quality culture for medical software.

5. FUTURE WORK

In the above we have discussed the application of the quality requirements engineering method MOQARE to medical software. Clearly, this only treats a small part of the general quality problems. We see the following issues for future work.

- The first step, of course, has to be to apply this in a real life scenario in order to check whether it is feasible. Naturally, this would lead to an improvement of MOQARE to fit it better to medical software. E.g. the categories of business and technical risks might not be sufficient? QD3 and QD4 are presented here as equivalent. QD4, however, may risk lives while QD3 only risks profitability.
- Based on this one could investigate different system types, e.g. proactive software rather than just reactive to user initiated transactions. Or multi-vendor situations as they are typical in hospitals require specific attention MUC4 e.g. is not covered through CM3 unless software vendors deliver data through precisely specified interfaces. Hospitals with renowned software development departments have been hit by surprise and failed to get mission critical lab data integrated with decision support systems (partially) because their technical skills were not sufficient to make and follow up on contracts that were supposed to force vendors to comply. Their existing skill set and enterprise culture turned out as vulnerabilities for the continuum of requirements when interfacing a highly variant set of pieces software from various sources with self-developed software [8]. For specific contexts it seems feasible to come up with checklists for typical quality defects, misuses, vulnerabilities and countermeasure.
- Given the high impact of change, it is also necessary to investigate the influence of change on quality. This can not only be captured through the analysis of change misuse cases (clarifying how the system can adapt to change which might threaten the quality goals). In addition, the evolution of the quality goals itself and its treatment in MOQARE is an open question. CM5 is an example of such evolution of quality goals. It violates a core assumption about the role of AntiBiAS that remained implicit: The original aim of AntiBiAS entailed that data on antibiotics **given** is complete. CM5 jeopardizes this concept by not only tolerating but actively supporting that the advisory function be bypassed through a mere documentation of antibiotics selected or present in ad-hoc processes.
- One way to amortize the effort in building models is typically to use the models during run-time. For MOQARE one can imagine incorporating a monitoring mechanism for quality defects and misuses in the system. It is e.g. interesting to investigate whether monitoring misuses is easier than monitoring quality directly.

6. ACKNOWLEDGMENTS

The first author was on sabbatical at the University of New South Wales in Sydney during the preparation of this paper. This provided a very positive and stimulating environment.

7. REFERENCES

1. Chen, B. Avrunin, G.B..Clarke, L.A. Osterweil, L.J. Automatic Fault Tree Derivation from Little-JIL Process Definitions**,** *Software Process Workshop (SPW 2006) and 2006 Process Simulation Workshop (PROSIM 2006),* Springer-Verlag LNCS, Vol. 3966, pp. 150-158, 2006.
2. Cimino; J.J., Li, J. Bakken, S.; Patel, V.S. Theoretical, empirical and practical approaches to resolving the unmet information needs of clinical information system users; *Proc. AMIA Symp.*, pp 170-174 ,2002
3. Dutoit, A.H.; McCall, R.; Mistrik, I.; Paech, B. (Eds.) *Rationale Management in Software Engineering*, Springer Verlag, 2006
4. Herrmann, A. Paech, B. "MOQARE: Misuse-oriented Quality Requirements Engineering", *Requirements Engineering Journa*l, Springer-Verlag, accepted for publication, 2007
5. Jones MT; Computers can land people on Mars, why can't they get them to work in hospitals? *Methods Inf Med* 42 410-5, 2003
6. Paech B, Kerkow D. Non-Functional Requirements Engineering - Quality is Essential. In: Regnell B, Kamsties E, Gervasi V (eds) *Proceedings of the 10th Intl. Workshop on Requirements Engineering REFSQ04*, Essener Informatik Beiträge Bd 9, pp 237-250, 2004
7. Wetter, Th; To decay is system: The challenges of keeping a health information system alive*; Int. J. Med. Inform.* 76 (S1), pp. 252–260, 2007
8. Wetter, Th. Safeguarding clinical software – A managerial case study about project management and oversight; *Proc. APAMI conference*, pp. 27-31, 2006