

# Modellierung von Software-Security mit aktivitätenbasierten Qualitätsmodellen

Stefan Wagner, Shareeful Islam

Fakultät für Informatik  
Technische Universität München  
{wagnerst,islam}@in.tum.de

**Zusammenfassung** *Security* oder *Informationssicherheit* stellt für Software immer noch eine große Herausforderung dar. Trotz breiter Anstrengungen Software sicher zu machen, ist die Zahl der berichteten Schwachstellen unvermindert hoch. Um dem entgegenzuwirken ist es wichtig, Security-Anforderungen klar zu formulieren und den Entwicklern und der Qualitätssicherung detaillierte Richtlinien an die Hand zu geben. Dazu wird die Modellierung von Software-Security mit Hilfe von aktivitätenbasierten Qualitätsmodellen vorgestellt.

## 1 Einleitung

Trotz der vielfältigen und kostspieligen Anstrengungen, die Sicherheit von Software sicherzustellen, ist bei den öffentlich-gemachten Sicherheitslücken kein signifikanter Rückgang zu verzeichnen [1]. Darüberhinaus sind aber nicht nur solche einzelne Lücken, sondern auch das Zusammenspiel aller Security-Mechanismen in einem Software-System entscheidend, was die Komplexität der Sicherheitsanalyse weiter erschwert. Software-Security ist daher immer noch eine große Herausforderung in heutigen Softwaresystemen.

Qualitätsmodelle beschreiben was mit *Qualität* bedeutet und verfeinern dieses Konzept in einer strukturierten Art und Weise. In der Terminologie von [2] sind also Qualitätsdefinitionsmodelle gemeint. In der Praxis wird dies oft auf Metriken wie *Zahl der gefundenen Fehler* oder sehr abstrakte Beschreibungen wie in der ISO 9126 [3] reduziert. Grundsätzlich werden Qualitätsmodelle mindestens auf zwei Arten in einem Softwareprojekt eingesetzt: (1) als Basis zur Definition von Qualitätsanforderungen und (2) zur Zuordnung von qualitätssichernden Maßnahmen und Messungen zu den Qualitätsanforderungen. Ersteres wird in der Regel durch die Beschränkung üblicher Qualitätsattribute (Zuverlässigkeit, Wartbarkeit, ...) eines Qualitätsmodells erreicht. In der Praxis findet man beispielsweise verkürzte Formulierungen wie "Das System soll einfach wartbar sein." Die zweite Verwendung von Qualitätsmodellen wird oft nicht explizit durchgeführt, sondern Metriken, wie die Zahl der durch Inspektion und Test gefundenen Fehler, werden direkt verwendet. Der Grund ist die hochkomplexe Zuordnung von Metriken zu abstrakten Qualitätsattributen. Es ist also prinzipiell wünschenswert, wenn Qualitätsmodelle hier mehr Struktur und Detail liefern, so dass sie eng in den Entwicklungsprozess integriert werden können.

*Problemstellung* Security sollte, wie auch andere Qualitätsattribute, früh im Entwicklungsprozess berücksichtigt werden. Jedoch fehlen auch im Bereich der Sicherheit immer noch integrierte und konkrete Qualitätsmodelle, die sowohl die präzise Spezifikation von Sicherheitsanforderungen, als auch deren direkte Umsetzung und Überprüfung im System unterstützen.

*Beitrag* Das erprobte Vorgehen, Qualität mit Hilfe von aktivitätsbasierten Qualitätsmodellen zu beschreiben, wird auf Security übertragen. Dabei werden bekannte Quellen benutzt und integriert. Ziel ist die Integration von Security mit Hilfe der Modellierung von System, System-Elementen, Umgebung und Prozess und deren Einfluss auf Aktivitäten, insbesondere Angriffe. Dadurch wird die Sicherstellung von Security mit in den allgemeinen Qualitätsmanagement-Prozess eingebunden.

## 2 Aktivitätenbasierte Qualitätsmodelle

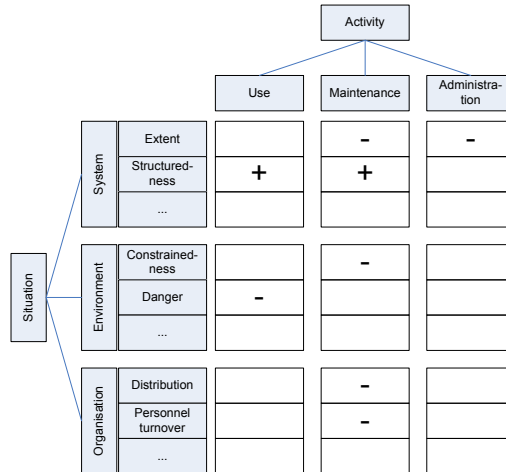
Zur Erreichung dieser Strukturierung und des Detailgrads wurde die Verwendung von aktivitätsbasierten Qualitätsmodellen vorgeschlagen [4]. Die Idee ist, abstrakte “-ilities” für die Definition von Qualität zu vermeiden und stattdessen Qualität in detaillierte Fakten herunter zu brechen und deren Einfluss auf die Aktivitäten, die auf und mit dem System durchgeführt werden, zu beschreiben. Für Wartbarkeit wird dies beispielsweise in [4] gezeigt. Das Modell enthält Informationen über die Charakteristika eines Softwaresystems und anderer interessanter Umgebungseinflüsse und deren Einfluss auf Wartungsaktivitäten, wie beispielsweise *Code lesen*, *Modifizieren* oder *Testen*. Ein konkretes Beispiel dafür sind redundante Methoden im Quelltext, also so genannte Klone. Sie haben verschiedene Einflüsse, besonders wichtig ist aber der negative Einfluss auf Modifikationen des Quelltextes, da Änderungen an Klonen an verschiedenen Stellen im Text durchgeführt werden müssen. Das bedeutet, dass falls eine Systementität *Methode* das Attribut *REDUNDANZ* besitzt wird dies einen negativen Einfluss auf die Aktivität *Modifizieren* (also eine Änderung der Methode) haben.

Das Modell enthält nicht nur diese Einflüsse der Fakten auf Aktivitäten sondern auch die Zusammenhänge untereinander. Sowohl die Fakten als auch die Aktivitäten sind als Hierarchien abgelegt. Die oberste Aktivität *Aktivität* besitzt Unteraktivitäten wie *Benutzen*, *Warten* oder *Administrieren* (siehe auch Abb. 1). In praxistauglichen Modellen werden diese dann weiter verfeinert. Beispielsweise kann die *Wartung* die Unteraktivitäten *Code lesen* und *Modifizieren* haben.

Die Fakten setzen sich zusammen aus *Entitäten* und *Attributen*, also Charakteristika einer Entität. Dies erlaubt die Entitäten einfach in einer Hierarchie zu organisieren. Die oberste Ebene ist hier die *Situation* des Softwareentwicklungsprojekts. Es enthält beispielsweise das *System*, seine *Umgebung* und die *Entwicklungsorganisation*. Wiederum müssen die Entitäten weiter verfeinert werden. Beispielsweise besteht das *System* aus statischen und dynamischen Aspekten. All Entitäten werden durch Attribute beschrieben. Ein Beispiel-Fakt ist die *STRUKTURIERTHEIT* des *Systems*: Ist das System in einer sinnvollen und definierten Art

und Weise strukturiert? Diese Fakten sind entweder automatisch oder manuell überprüfbar. Soweit möglich werden auch entsprechende Metriken angegeben.

Beide Hierarchien, der Faktenbaum und der Aktivitätenbaum, können zusammen mit den Einflüssen der Fakten auf die Aktivitäten als Matrix wie in Abb. 1 dargestellt werden. Die Einflüsse sind dabei als Einträge in die Matrix dargestellt, wobei ein “+” ein positiver und ein “-” ein negativer Einfluss ist.



**Abbildung 1.** Abstrakte Darstellung eines aktivitätenbasierten Qualitätsmodells als Matrix

Es existiert eine abstrakte, textuelle Notation, die die entsprechenden Entitäten, ihre Attribute und deren Einfluss auf Aktivitäten kompakt darstellt. Beispielsweise ist das folgende Tupel ein Eintrag im Qualitätsmodell über konsistente Bezeichner: [Bezeichner | KONSISTENZ]  $\xrightarrow{+}$  [Modifizieren] Dies bedeutet, dass konsistente Bezeichner einen positiven Einfluss auf Modifikationen am System haben. Im Modell selbst werden darüberhinaus aber noch weitere Informationen, wie ausführliche Beschreibungen und Quellen, dokumentiert.

Durch die Verwendung aktivitätenbasierte Qualitätsmodelle können früh konkrete und überprüfbare Anforderungen aufgestellt [5] und später auch konkrete Qualitätssicherungsmaßnahmen abgeleitet werden. Beispielsweise können Checklisten für Reviews automatisch generiert werden [4]. Weitere Informationen über diese Art von Modellen und den bisherigen Erfahrungen in der Praxis sind in [4, 6–8] zu finden.

### 3 Das Security-Modell

Das wichtigste neue Konzept für die Modellierung von Security ist Angriffe als Aktivitäten zu sehen und in den Aktivitätenbaum zu integrieren. Diese Akti-

vitäten müssen also durch Fakten negativ beeinflusst werden, um ein hochqualitatives System zu erreichen.

### 3.1 Der Aktivitätenbaum

Zuvorderst muss zwischen erwarteten und unerwarteten Angriffen unterschieden werden. Eine Hauptschwierigkeit bei Security ist nämlich, dass es unmöglich ist, alle zukünftigen Angriffe, dem ein System ausgesetzt sein wird, zu kennen, da täglich neue Angriffe entwickelt werden. Deshalb ist es wichtig zwei Strategien zu verfolgen: (1) das System gegen erwartete Angriffe absichern und (2) das System prinzipiell zu härten, also unempfindlicher gegenüber Angriffen zu machen. Für die Klassifizierung der Angriffe können verschiedene Quellen verwendet werden. Wir bauen hauptsächlich auf die *Common Attack Pattern Enumeration and Classification* (CAPEC) [9], die vom U.S. Department of Homeland Security vorangetrieben wird. In CAPEC werden existierende Angriffsmuster gesammelt und klassifiziert. Diese Angriffe werden in einer Hierarchie angeordnet, die direkt im Aktivitätenbaum wiederverwendet werden kann (Abb. 2).

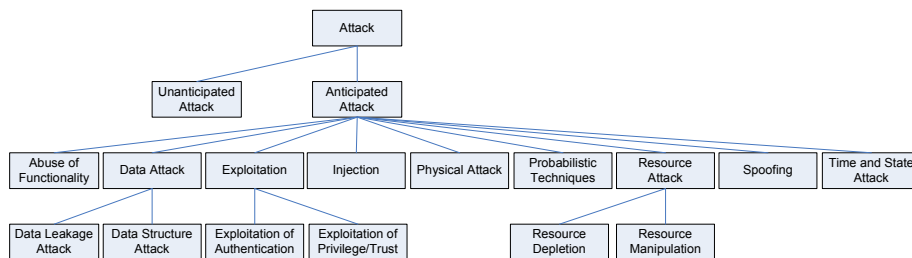


Abbildung 2. Die obersten Ebenen des Angriff-Teilbaums des Aktivitätenbaums

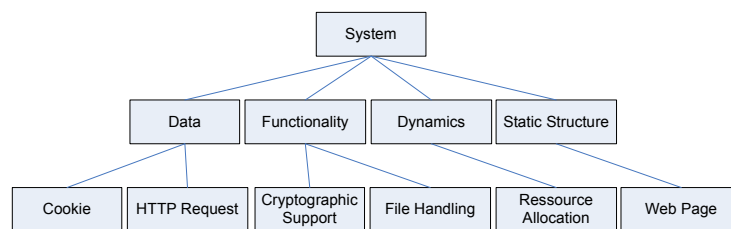
Die erwarteten Angriffsarten (*anticipated attacks*) werden in eine Reihe von Mustern und Untermustern aufgeteilt. Die Einteilung folgt dabei im wesentlichen der CAPEC und ist dabei nicht völlig überscheidungsfrei. Dies ist durch die Komplexität der Angriffsarten nicht möglich, aber auch nicht notwendig. Entscheidend ist, dass bestimmte Angriffsmuster schnell gefunden werden können. Teilweise wird dabei unterschieden, *was* angegriffen wird (*Abuse of Functionality*, *Data Attack*, *Physical Attack*, *Resource Attack*), teilweise auch *wie* der Angriff durchgeführt wird (*Exploitation*, *Injection*, *Probabilistic Technique*, *Spoofing*, *Time and State Attack*).

### 3.2 Der Faktenbaum

Die Erstellung des Faktenbaums ist demgegenüber wesentlich komplizierter. Es muss das vorhandene Wissen über Charakteristiken eines Systems, der Umgebung und Organisation enthalten, die diese Angriffe beeinflussen. Dazu verwenden wir wiederum eine Reihe von Quellen, unter anderem die ISO/IEC 27001 [10]

oder die Sun Secure Coding Guidelines for the Java Programming Language [11]. Zwei Hauptquellen stellen dabei bestimmte Teile der *Common Criteria* (CC) [12] und die *Common Weakness Enumeration* (CWE) [13] dar. Die Common Criteria beschreiben Anforderungen, was ein System leisten soll, damit es sicher ist. Dabei liegt der Fokus auf Funktionalitäten. Die CWE beleuchtet dies stärker von der anderen Seite und beschreibt wiederkehrende Schwachstellen, die von Angriffen ausgenutzt wurden. Zusammengenommen liefern diese beiden Quellen ein starkes Fundament für das Security-Modell.

Wir können hier nicht alle Details des Modells darstellen, geben aber einige Beispiele, wie Wissen aus den Quellen in das Modell überführt wurde. Dafür verwenden wir einen Unterbaum des Faktenbaums (Abb. 3).



**Abbildung 3.** Beispielhafte Einträge des Faktenbaums

Viele der Einträge im Qualitätsmodell, die ihren Ursprung in den CC haben, wurden als Teil der Entität *Functionality* modelliert, da sie überwiegend Verhaltensaspekte beschreiben, die für Security wichtig sind, beschreiben. Ein Beispiel ist die kryptographische Unterstützung des Systems als Teil der Funktionalität. Nach den CC kann sie in *Cryptographic Key Management* und *Cryptographic Operation* unterteilt werden. Ersteres enthält wiederum *Cryptographic Key Generation*, zu dem in den CC definiert wird, dass sie in Übereinstimmung mit einem spezifizierten Algorithmus und spezifizierten Schlüsselgrößen sein soll. Im Modell wird dies durch die Verwendung des Attributes *APPROPRIATENESS* für *Cryptographic Key Generation* ausgedrückt. Die textuelle Beschreibung dieses Fakts ist dann: “The system generates cryptographic keys in accordance with a specified cryptographic key generation algorithm and specified cryptographic key sizes that meet a specified list of standards.” Die CC enthalten leider keine Beschreibungen von Einflüssen, die sie noch nützlicher machen würden, da die Motivation für die angegebenen Anforderungen gleich mitgeliefert würden. Aus diesem Grund ergänzen wir diese Information aus anderen Quellen. In diesem Fall enthält CAPEC die Beschreibung des Angriffs *Kryptoanalyse* und die Vermeidungsstrategie bewährte kryptographische Algorithmen mit empfohlenen Schlüsselgrößen zu verwenden. Also sieht der Einfluss wie folgt aus:  $[\text{Cryptographic Key Generation} \mid \text{APPROPRIATENESS}] \xrightarrow{-} [\text{Cryptanalysis}]$

Die CWE enthält oft Charakteristiken eines Systems und speziell von Quelltext, die vermieden werden sollten. Dies kann so im Modell verwendet werden.

Einige beschriebene Schwachstellen in CWE beziehen sich nicht auf bestimmte Attacken, werden aber als Indikatoren für potenzielle Sicherheitslöcher angegeben. Diese werden als Fakten modelliert, die einen Einfluss auf unerwartete Angriffe haben. Ein Beispiel dafür ist toter Code, beispielsweise ungenutzte Variablen. Im Modell ist dies wie folgt enthalten:  $[Variable | SUPERFLUOUSNESS] \xrightarrow{+} [Unanticipated Attack]$ .

Schließlich muss noch angemerkt werden, dass im Modell nicht nur die Möglichkeit besteht Einflüsse auf Angriffe zu modellieren, sondern auch andere Aktivitäten, wie die Benutzung des Systems können beeinflusst werden. Abhängig von der Definition von Security kann man damit beispielsweise auch die zufällige Veröffentlichung sensibler Daten darstellen.

### 3.3 Beispiel

Wir bearbeiten gerade eine größere Fallstudie zur Anwendung des Security-Modells auf den Servlet-Container Tomcat<sup>1</sup>. Insgesamt liegt der Fokus dort auf der Bestimmung der Security-Anforderungen. Wir beschränken uns hier aber nur auf den Vergleich der veröffentlichten Security-Lücken von Tomcat und korrespondierender Einträge im Qualitätsmodell. Dies erlaubt noch keine quantitative Aussage über den Wert des Einsatzes des Qualitätsmodells, aber es zeigt das Potential des Ansatzes auf.

Insgesamt wurden für die Tomcat-Version 6.0 vom Dezember 2006 bis Juli 2008 19 Schwachstellen veröffentlicht. Die Tomcat-Entwickler klassifizieren diese Schwachstellen in die Typen *Cross-Site Scripting*, *Session Hi-Jacking*, *Directory Traversal*, *Information Disclosure*, *Data Integrity* und *Elevated Privileges*. Diese Typen können also zum Teil bereits auf unseren Aktivitätenbaum zurückgeführt werden. Weiterhin wurde die Wichtigkeit der Schwachstellen bewertet. Es wurden nun alle diese Schwachstellen mit dem Security-Modell verglichen, um zu analysieren, ob entsprechende Qualitätsregeln enthalten sind. Tabelle 1 zeigt die Zahl der Schwachstellen im Verhältnis zur Zahl der entsprechend gefundenen Qualitätsregeln im Modell auf. Es zeigt sich, dass nur für drei von 19 Schwachstellen keine Regel gefunden werden konnte. Es hätte also die Chance bestanden durch Verwendung des Modells diese Schwachstellen zu vermeiden. Ob dies wirklich gelingen kann, wird derzeit in der Fallstudie untersucht.

## 4 Verwandte Arbeiten

Allgemeine Qualitätsmodelle, die einen Dekompositionsmechanismus verwenden wurden bereits von Boehm [14] und McCall [15] vorgeschlagen. Trotz der vielfältig dokumentierten Schwächen [16] folgen aktuelle ISO-Standards immer noch dieser Tradition [3]. Im Gebiet der Security, können die oben erwähnten Richtlinien wie die *Secure Coding Guidelines for the Java Programming Language* oder die *CERT C Secure Coding Standard* verwendet werden. Solche Richt-

<sup>1</sup> <http://tomcat.apache.org/>

**Tabelle 1.** Zusammenfassung des Vermeidungspotentials

Typ	Anteil	Wichtigkeit	Anteil
cross-site scripting	7/7	low	10/10
session hi-jacking	4/4	moderate	2/2
directory traversal	1/2	important	4/7
information disclosure	3/4	total	16/19
data integrity	0/1		
elevated privileges	1/1		

linien sind wichtig und wertvoll, da sie spezifische und konkrete, oft auch überprüfbare Regeln angeben. Jedoch enthalten sie meist nicht den Einfluss, den die Ein- bzw. Nichteinhaltung der Regeln hat. Darüberhinaus liefern Security-Normen, wie die ISO/IEC 27001 [3] und die *Common Criteria* [12] einen breiten Blick auf Security. Es werden nicht nur für Quelltext Richtlinien angegeben, sondern auch Aspekte der Funktionalität oder der Organisation berücksichtigt. Dies führt aber auch dazu, dass diese Normen wesentlich generischer und damit schwieriger zu überprüfen sind. Auch hier werden Auswirkungen und Einflüsse kaum berücksichtigt.

Chung und Nixon [17] haben eine systematische Methode zum Umgang mit Qualitätscharakteristika, das NFR-Rahmenwerk. Dieser Ansatz betrachtet Korrelationsregeln, um existierende Ziele mit neuen Zielen zu verbinden. Eine Einbindung in die Qualitätssicherung wird aber nicht explizit angegeben. Zu einem gewissen Grad, stehen auch Anforderungsansätze für Security-Anforderungen im Bezug zu dieser Arbeit. Bekannte Vertreter sind hier SQUARE [18] und darauf aufbauend SREP [19]. Beide konzentrieren sich stärker auf den Prozess, beziehen sich aber sonst auf klassische Qualitätsmodelle.

## 5 Zusammenfassung

Die hier beschriebene Modellierung von Security mit Hilfe von aktivitätenbasierten Qualitätsmodellen stellt ein strukturiertes Vorgehen zur Erfassung von sicherheitsrelevantem Wissen zur Verfügung. Darüberhinaus kann so auch die Sicherstellung von Sicherheit in den normalen Qualitätsmanagement-Prozess eingebunden und sogar Überlappungen mit anderen Qualitätsattributen identifiziert werden. Wir arbeiten derzeit an einem umfassenden *Security Requirements Engineering*-Ansatz auf Basis des beschriebenen aktivitätenbasierten Qualitätsmodells.

## Danksagung

Diese Arbeit entstand teilweise mit Förderung des Bundesministeriums für Bildung und Forschung (BMBF) im Projekt *QuaMoCo* (Förderkennzeichen: 01 IS 08023B).

## Literatur

1. CERT: Vulnerability remediation statistics. Online verfügbar unter [http://www.cert.org/stats/vulnerability\\_remediation.html](http://www.cert.org/stats/vulnerability_remediation.html)
2. Deissenboeck, F., Juergens, E., Lochmann, K., Wagner, S.: Software quality models: Purposes, usage scenarios and requirements. In: Proc. 7th International Workshop on Software Quality (7-WoSQ), IEEE Computer Society (2009)
3. ISO: ISO 9126: Product Quality – Part 1: Quality Model (2003)
4. Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., Girard, J.F.: An activity-based quality model for maintainability. In: Proc. 23rd International Conference on Software Maintenance (ICSM '07), IEEE Computer Society Press (2007)
5. Wagner, S., Deissenboeck, F., Winter, S.: Managing quality requirements using activity-based quality models. In: Proc. 6th International Workshop on Software Quality (WoSQ '08), ACM Press (2008) 29–34
6. Winter, S., Wagner, S., Deissenboeck, F.: A comprehensive model of usability. In: Proc. Engineering Interactive Systems 2007 (EIS '07). Volume 4940 of LNCS., Springer (2008)
7. Wagner, S., Deissenboeck, F., Feilkas, M., Juergens, E.: Software-Qualitätsmodelle in der Praxis: Erfahrungen mit aktivitätsbasierten Modellen. In: Workshop-Band SQMB 2008, TU München (2008)
8. Mas y Parareda, B., Streit, J.: Software quality put into practice. In: Workshop-Band SQMB 2008, TU München (2008)
9. Homeland Security: Common attack pattern enumeration and classification (CAPEC). Available Online at <http://capec.mitre.org/>. Accessed in October 2008
10. ISO: ISO/IEC 27001: Information technology – Security techniques – Information security management systems – Requirements (2005)
11. Microsystems, S.: Secure coding guidelines for the java programming language, version 2.0. Available Online at <http://java.sun.com/security/seccodeguide.html>
12. CCRA: Common criteria for information technology security evaluation, version 3.1. Available Online at <http://www.commoncriteriaportal.org/>
13. Homeland Security: Common weakness enumeration (CWE). Available Online at <http://cwe.mitre.org/>. Accessed in October 2008
14. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G.J., Merrit, M.J.: Characteristics of Software Quality. North-Holland (1978)
15. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in software quality. Reports NTIS AD/A-049 014, 015, 055, US Rome Air Development Center (1977)
16. Kitchenham, B., Pfleeger, S.L.: Software quality: The elusive target. IEEE Software **13**(1) (1996) 12–21
17. Chung, L., Nixon, B.A.: Dealing with non-functional requirements: Three experimental studies of a process-oriented approach. In: Proc. 17th ICSE. (1995) 25–37
18. Mead, N., Steheny, T.: Security quality requirement engineering methodology. In: Proc. Workshop on Software Engineering for Secure Systems (SESS '05). (2005)
19. Mellado, D., Medina, E., Piattini, M.: Acommon criteria based security requirements engineering process for the development of secure information system. Computer standards & interfaces **29** (June 2007) 244–253