dpunkt.verlag im Inter

Ina Schieferdecker ·
Stephan Goericke (eds.)

Setting Quality Standards

Since 1997 the international "Confer
Quality Engineering in Software Tec
(CONQUEST) has gathered together t
ware and quality engineering comm
discuss aspects of software quality
share experiences in industrial and
projects.

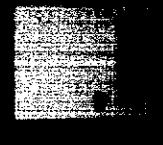
The book constitutes a selection of
sented speeches and lectures. They
first-hand information on the practi
application and ongoing developme
methods and techniques and provid
into real-life case studies along with
quality analysis and evaluation.

The proceedings of the CONQUEST
include

- Processes
- Experience Reports
- Next Generation Testbed for Medi
- Engineering
- Testing
- Secure Software Development
- QA for Finance IT
- Metrics
- Quality and Safety

CONQUEST 2008 is organized by th
national Software Quality Institute
in cooperation with Fraunhofer FO
the German Society for Informatic
kindly supported by the Ministry
mics of Brandenburg, the networ
rity and Safety made in Berlin-Br
(SeSamBB), and the ZukunftsAgen
denburg (ZAB).

ISBN 978-3-89864-567-6



Schieferdecker · Goericke
Setting Quality Standards

Ina Schieferdecker · Stephan Goericke (eds.)

Setting Quality Standards

Proceedings of the CONQUEST 2008

11th International Conference on
Quality Engineering in Software Technology

Potsdam 2008

dpunkt.verlag

Kindly supported by:

LAND BRANDENBURG
Ministerium für Wirtschaft

SeSamBB
Security and Safety made in Berlin-Brandenburg

ZAB ZukunftsAgentur Brandenburg

Ina Schieferdecker · Stephan Goericke (eds.)

Setting Quality Standards

Proceedings of the CONQUEST 2008

11th International Conference on
Quality Engineering in Software Technology
Potsdam 2008

The conference is organised by iSQI – International Software Quality Institute.

dpunkt.verlag

ASQF Arbeitskreis Software-Qualität und -Fortbildung e.V.

iSQI International Software Quality Institute

iSQI – International Software Quality Institute
Am Weichselgraben 19 · 91058 Erlangen · www.isqi.org

Ina Schieferdecker
ina.schieferdecker@fokus.fraunhofer.de

Stephan Goericke
stephan.goericke@isqi.org

Editor: Vanessa Wittmer
Copy-Editor: Julia Neumann, Prince George BC, Canada
Producer: Birgit Bäuerlein
Cover Design: Helmut Kraus, www.exclam.de
Printer: Koninkliijke Wöhrmann B.V., Zutphen, Netherland

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

ISBN 978-3-89864-567-6

1st Edition

Ringstraße 19 B
69115 Heidelberg
Germany

543210

Greetings

Stel Pinhasov Beck
Commercial Attaché of the Embassy of Israel and
Director of the Israel Trade Center:

Shalom,

Dear Participants,

Welcome to this year's CONQUEST in Potsdam with its special focus on Israel.

As the Economic Representative of the State of Israel in Germany, I am especially pleased that Israel has been chosen to be iSQI's official partner country at CONQUEST 2008.

This is a great opportunity to present Israel's outstanding technologies to the international software development community.

At the same time, this year's CONQUEST offers an excellent platform to further intensify the bilateral relations in the software and security sector between Germany and Israel.

Israel's software and security industries' reputation as one of the most innovative in the world and the enormous amount of requests to my office from respective companies looking for business partners in Germany already give an idea about the strong potential of future co-operation.

That is why the Israel Trade Center and the Israel Export and International Co-operation Institute have put great efforts into bringing a delegation of Israeli companies to this year's CONQUEST. In particular, the Brandenburg meets Israel BusinessLink with its pre-scheduled B2B meetings will enable experts to get to know each other and to exchange potential business ideas.

I wish all of us an inspiring and fruitful conference and I'm looking forward to meeting you in Potsdam!

Sincerely,
Stel Pinhasov Beck

Greetings

Prof. Ina Schieferdecker, Conference Chair:

I am pleased to welcome you to CONQUEST 2008 in September 24-26, 2008 in Potsdam, Germany. Since 1997, CONQUEST – the International Conference on Quality Engineering in Software Technology – has been the platform for software professionals to get to know the latest methods and tools for quality engineering from industry and research, to listen to practical experiences and show cases, to see how quality engineering methods are successfully deployed in an industrial environment, to learn about recent tool and service offers, to share experiences and to discuss future directions.

CONQUEST 2008 will feature tutorials and presentations of internationally renowned speakers and high-quality peer-reviewed presentations from members of the quality engineering community. It provides a full picture of software quality in theory and practice.

It is a particular pleasure to celebrate at CONQUEST the 10th anniversary of the Software Testing Working Group at ASQF, which was created back in 1998 and currently runs over 20 meetings a year with more than 500 participants. It also contributed to the Certified Tester Initiative in Germany which, together with more than 30 other national testing boards, can currently draw on more than 90.000 certificates worldwide. All working group members, certified testers and interested parties are cordially invited to the Software Tester Welcome, which is supported by the German Testing Board (GTB). In addition, the GTB-

Forum will take place for the first time in Potsdam as an independent, but affiliated event of CONQUEST.

A variety of international speakers and participants will give you practical and research insights into current and topical aspects of quality engineering. This broad perspective will be presented by attendees from all over the world including Europe, Asia and America. Our exhibition on recent methods, services and tools will include 15 national and international companies from the software quality field and will give you detailed insights into their tools and services.

I wish us all a successful and enjoyable CONQUEST 2008.

Yours truly,
Ina Schieferdecker

Greetings

Stephan Goericke,
Director iSQI:

As the Director of the International Software Quality Institute, I'm very pleased to present this documentation issue for the CONQUEST 2008. Meanwhile, it's now the 11th time that software engineers from all over the world have accepted iSQI's invitation to come to be informed and to exchange information on the topic of Software Quality Improvement. This year's new topic concerns itself with "security and safety" – on the one hand, the area of software development and on the other, secure and safe software-based systems.

The 11th CONQUEST offers another novelty, about which I'm particularly pleased. This year, Israel has become the partner country of the conference, to commemorate the occasion of the 60th anniversary of the founding of the State of Israel and the good business relations which iSQI has made in recent years in the high-tech Mediterranean state. Many representatives of Israeli businesses and institutions are involved in the CONQUEST 2008. It is an honour for us to be able to further enhance business relationships and the transfer of knowledge between the two countries. Only then can we reach our goal to better software quality and institute better standards worldwide.

iSQI now supplies certification in over 40 countries, standards now exist in the areas Software Test, TTCN-3, Software Architecture, Project Management, Innovation Management, Configuration Management, Requirements Engineering, the SPICE-Assessors Certification following iNTACS, IT Security Manage-

ment, Secure Software Engineering, Information Security and soon also the V-Model XT. In addition, the iSQI has developed a standard on the basis of these "certified" programs, which incorporates the recognition of many years of practical experience and which allows for a flexible adaption in an individual job situation: QAMP (Quality Assurance Management Professional). If you're interested in this standard, you'll learn more about it during the CONQUEST.

I'd like to thank all those partners who supported us during this CONQUEST in Potsdam and also all those who helped to make the CONQUEST what it is: one of the most important specialized events in the area of software quality.

I wish you all the best and great success in your work and I would be very pleased to meet you at our future conferences and educational programs.

Sincerely,
Stephan Goericke

Program Committee

Conference Chair:

Ina Schieferdecker, Fraunhofer FOKUS, Germany

Program Committee:

Girts Baltaisbrencis, Exigen Services, Latvia
Manfred Broy, Technical University Munich, Germany
Wim Demey, FRSGlobal, Belgium
Winfried Dulz, University of Nuremberg-Erlangen, Germany
Karol Frühauf, INFOGEM, Switzerland
Anne Mette Jonassen Hass, DELTA, Denmark
Georg Heidenreich, Siemens, Germany
Bernard Homès, TESSCO Group, France
Dehua Ju, ASTI Shanghai, China
Dieter Landes, University of Applied Sciences Coburg, Germany
Peter Liggesmeyer, Fraunhofer IESE, Germany
Alon Linetzki, Best-Testing, Israel
Patricia McQuaid, California Polytechnic State University, USA
Pedro Merino, University of Malaga, Spain
Ludger Meyer, Siemens, Germany
Joanna Nowakowska, Oracle, Poland
Mohammad Nuruzzaman, Daffodil International University, Bangladesh
Barbara Paech, University of Heidelberg, Germany
Sachar Paulus, SAP, Germany
Helmut Pichler, ANECON, Austria
Klaus Pohl, University of Duisburg-Essen, Germany
Ita Richardson, University of Limerick, Ireland
Bj Rollison, Microsoft, USA
Alexander Rudolph, Ecolab, Austria

Sergii Riapolov, Ukrainian Scientific Center for Development of Information Technologies (ITDEV) under Ministry of Education and Science of Ukraine, Ukraine
Achim Schmidt, Beta Systems, Germany
Kurt Schneider, University of Hannover, Germany
Andreas Spillner, University of Applied Sciences Bremen, Germany
Maria Stefanova-Pavlova, CITT Global, Bulgaria
Angela Vozella, CIRA, Italy
Mario Winter, University of Applied Sciences Cologne, Germany
Walter Wintersteiger, Management & Informatik, Austria
Peter Zimmerer, Siemens, Germany

Additional Reviewers:

Stefan Wagner, Technical University Munich, Germany
Eva Geisberger, Technical University Munich, Germany
Marco Kuhrmann, Technical University Munich, Germany
David Cruz, Technical University Munich, Germany
Heiko Stallbaum, University of Duisburg-Essen, Germany
Times Illes-Seifert, University of Heidelberg, Germany
Lars Borner, University of Heidelberg, Germany
Jürgen Rückert, University of Heidelberg, Germany
Shamsuddin Ahammad, Daffodil Institute of IT (DIIT), Bangladesh

Table of Contents

Object Oriented Design Rules – Definition and Automatic Detection

D. Cabrero

Spanish Ministry of Internal Affairs, Spain

J. Garzás Parra

Kybele Consulting S.L. Madrid, Spain

M. Piattini Velthuis

Departamento de Tecnologías y Sistemas de Informacíon (UCLM), Spain

Abstract

This research work targets three main objectives: In the first place, it gathers all the information that is available on the definition and automatic detection of object oriented design rules, including sources of rules and previously developed tools. Secondly, it analyses and classifies the complete set of rules into different categories and sets the basics for a methodology, taking into account the rework needed to fix a given rule violation. In the third place, it introduces the value-based approach applied to rule definition and detection.

The main purpose of this research is to provide mechanisms for managing the huge amount (typically measured by thousands) of violated rules that are detected when auditing a medium-sized software system. Therefore, this work analyses value in terms of the relative importance of each rule violation. This categorization allow engineers to optimise their resources and to focus first on the most important design problems.

1 Introduction

Automatic software analysis tools aim to find improvement possibilities on large software developments by checking compliance with best practices. Commercial and open source tools help developers and QA (Quality Assurance) teams to

obtain low cost and high quality software products. The word "Automatic" suggests that no manual effort is needed to detect opportunity improvements in a given product, hence the great importance of the topic being discussed.

Lots of automatic code analysis tools are emerging nowadays, and static analysis has become a common practice. However, a quick review of the market highlights that those tools are quite "code-based", that is, they help programmers to use the resources of a given language properly. Very few rules based on design heuristics, principles or bad smells can be found. [Moh 06] highlights that current state of the art focuses on "detecting defects such as bugs, unused code, or syntax errors. These tools focus on code-level problems but do not address higher-level defects". In [Moh 07], the author also states that "design defects have not been precisely specified and there are few appropriate tools that allow their detection as well as their correction".

Even though Moha's work and some others [Mar 04] do address this problem, we are still left looking for a better understanding between those implementing the tools and the research community concerning categorization and detection of design defects. On that topic, by means of conducting a literature review, we concluded that rule-based analysis applied at early stages of the project lifecycle is an interesting research gap still to cover.

This article carries out a review of the literature on previous research work concerning definition and detection of design defects. A new approach addressing this research topic is also provided.

The remainder of the paper is organized as follows: Section 2 describes the concept of design rule. Section 3 gives an overview of the available automatic rule detection tools. Section 4 takes as its focal point the sources of rules presently available, both in the existing frameworks and in the research community. Sections 5 go into greater detail about the foundation of value considerations applied to this field. Finally, sections 6 draws some conclusions and identifies future research work.

2 Why design rules?

The concept of "rule" applied to software design is not new. More than 20 years ago, [Gid 84] introduced a set of rules addressing structural design and codification. As with every design (or coding) rule, they also based their proposal on the fact that the maintenance cost of a software system is directly related to how often is it expected to change in the future [Wie 06], and thus to how expensive it is to modify that system. A high quality design must be compliant with rules, because it means that it will be easy to maintain. Thus, when we have a software design and we need to improve it, the fastest way is to automatically detect the deficiencies (violated rules) and fix them.

To talk in more specific terms, design rules are the "what", design patterns are the "how", and refactorings are the "how to apply" design practices. In this case, we are interested in the "what" [Gar 05].

A basic thing to point out is the difference between coding rules and design rules, although the great majority of frameworks misunderstand their meaning and present them as a whole. From our point of view, a design rule should be "language independent". For example, a rule concerning the performance of a given operator in a given language is a codification rule, not a design rule, because it depends on the language. As a matter of fact, we consider a "design" rule to be one which is based only on information available at design level, when the code still does not exist. This kind of rules could therefore be applied following a normal development cycle in the design phases, not during the codification phase. Design rules usually point out defects at a deeper level, which typically need a larger extent of rework to be fixed (redesign of a component) than at code level.

In the context of this research work, we will use the term "design" defect and rule as those that can be detected at design phases of the development, in opposition to "coding" defect and rules, which need to be detected once the coding phase has already started.

3 Categorization of existing rule detection approaches

In the software industry, analysis and design deliverables are often deficient or nonexistent. Because of that, the majority of the tools available in the market nowadays are based on "source code parsing", also called "Abstract Syntax Tree-Based" (AST-Based) techniques. A brief review of this approach is provided in the next subsection.

On the other hand, we can find model-based methods, such as models for interpreting and checking (e.g. UML), explained in subsection 4.2. Other alternative approaches, such as languages capable of describing and detecting design defects are described in subsection 4.3. These last categories of "alternative" methods are mostly provided by the research community, through contributions coming from journals and conferences.

3.1 AST-Based Tools

[Mor 07] comments that "A commonly used approach to build a design rule checker is as a framework with an API which provides a means of inspecting the program. Examples of this approach include PMD or FindBugs". The great majority of the open source and commercial tools are based on this technique (see sub-section 4.2 for further detail).

Those tools are based on code parsing techniques, that is, a compiler generates a tree (Abstract Syntax Tree) representing the code structure. By inspecting the tree using an API provided by the tool, a programmer can specify the set of conditions of compliance of a specific rule. Finally, the tool allows the user to select a given set of rules, which will be automatically applied to a given source code.

This kind of technique also allows the use of this information to obtain OO metrics, which "can be used as indicators for automatically detecting design improvements" [Sah 00], or bad smells [Mun 05].

The main advantages of this approach are the applicability of this technique, even when the design and analysis is not available, and the flexibility provided by the programming language. As a significant drawback, we can highlight that in some cases this method could be a tedious and labour-intensive task [Moh 06]. However, this kind of technique does allow us to assess the system at design stages, when the code is still not developed, by means of automatic code generators that build the code from the model before each evaluation.

3.2 Model-Based Tools

The majority of the research community believes that model-based tools should focus on the UML specification [Fow 03], which is the most widely-used representation when modelling Object Oriented systems.

A good summary of related work concerning design rules implemented on design models can be found in [Egy 07], but both his own work and related pieces of work appear to focus mainly on detecting inconsistencies between different models. For example, that a message in a sequence diagram must match a class method.

Surprisingly, the literature review did not reveal any UML-Based tool that addressed design rules as understood by [Gar 05], but drew attention instead to tools capable of performing a consistency analysis between the different UML diagrams. We consider the development of a UML-based rule checking tool to be an interesting gap in the research which is still to be covered that is not addressed in this paper. On that topic, we believe that a XML Metadata Interchange-XMI-based tool would be the most suitable implementation, due to the fact that almost all the UML tools in the market are beginning to be compliant with the XMI specification.

In contrast with the AST-based tools, model-based tools are recommended at design stages, and inapplicable to the legacy code. Note that no rules implemented under this technology will address coding conventions or best practices. They will instead look at real language-independent design rules, which should be checked and fixed before low level coding rules, as explained above.

3.3 Language-Based Tools

[Mor 07] have recently proposed a new framework, based on a language specified using AOP-Aspect Oriented Programming techniques. To be more specific, it is based on the AspectJ facility [Kic 01]. Its name is PDL-Program Description Logic, and the proposal allows declarative definitions of programmatic structures, in this case structures corresponding to a violated rule.

Focusing on the same line of research, [Moh 06] also define a language based on the BNF notation, and use it to specify and describe design defects, and to generate detection algorithms. A proof of concept has been performed implementing the Blob, Functional Decomposition, Spaghetti Code, and Swiss Army Knife design defects [Bro 98].

This approach has an important advantage. Once the language is defined, new rules can be developed and checked without programming. This is a great point in its favour, and a lot of research work seems to follow this line.

On the other hand, a more in-depth demonstration of the expressive power of the language might be needed. A complete set of rules is still to be defined using those languages, to validate their flexibility. However, whatever the lessons learned from this work, as yet not completed, may turn out to be, the approach will still need to find a good point of compromise between flexibility and simplicity.

3.4 Choosing among different options

As mentioned above, several technologies can be applied to the specific purpose of defining and detecting rules: AST-based, model-based or language-based tools. Advantages and drawbacks of each option are also set out.

Concerning AST-based and model-based tools, and taking into account the context of the research, our preferred option is AST-based technology. There are three main reasons for our choice. Firstly, AST is the only technology that can cover every kind of rule (coding, correct API usage and design). For example, model-based and language-based solutions can not detect coding rules, and language-based solutions need further research to achieve a language that is expressive enough to define all the rules of design and architecture. The second reason for this choice is the fact that there is much work that has already been done which implements many coding rules, as well as some design rules. Finally, AST-based technology is the most widely-used technology. It has been employed by the industry for a while already and lots of open-source frameworks, such as PMD or Checkstyle, already provide the basis for performing the work.

4 Available Sources of Design Rules

4.1 Related Research Work

For many years, authors have highlighted the need for classification of the knowledge available throughout the literature of any given discipline, and for it to be organized into "chunks" [Sha 90] of knowledge. [Gar 05] proposed a rule-based ontology that gathered together the previous research work in Object Oriented Knowledge. This work included heuristics [Rie 96], patterns [Gam 95], principles [Mar 97], refactorings and bad smells [Fow 99], all of which indicate that a problem may exist that you need to address. The following table shows some rules extracted from [Gar 05].

If there are dependencies on concrete classes then these dependencies should be on abstractions.
If a hierarchy of classes has too many levels then reduce the level of inheritance using composition or redesign
If an object has a different behaviour according to its internal state then place each one of these behaviour types in a separate class.
If a super class knows any of its subclasses then eliminate it.
If a class collaborates with too many others then reduce the number of collaborations.
If a change in an interface has an impact on many clients then create specific interfaces for each client.
If a class rejects something that it inherited then avoid it, generally for delegation.
If there is not an abstract class between an interface and its implementation then create an abstract class with an implementation by default between the interface and the class that implements it.

Tab. 1 *Some OODK rules*

[Sku 96] also gives a set of rules that help to avoid design violation when developing programs, and [Tra 99] suggests the use of a set of "reading techniques" to "read" design artifacts manually. These techniques may show the way to implement some automatic detection rules. [Chi 94], in their very well known set of OO metrics, also provide an easy way of automating design and code checks.

Those are some examples of how the research community focused this issue in the past. However, although several rule tools are available on the market, many rules presented above have never been automated using any rule framework. As a result of this literature and market review, we concluded that there is a need for a better understanding between the different parts of research work that is concerned with OO Knowledge and rule tool frameworks.

As a significant example of this weak point, we could point out that some rules specified more than 20 years ago [Gid 84] for structural analysis, such as the convenience of allowing cyclic dependencies, have not been completely implemented (they have been put into operation only at package level, but not at method or class level).

4.2 Open Source and Commercial Frameworks

In the context of this review, we evaluated the most significant open-source frameworks and we found that very few rules could be considered "design" rules, excluding specific API usages or recommendations for the checking of coding:

- PMD [PMD 08] : 6 design rules of over 200.
- Checkstyle [Che 08] : 8 design rules of over 200.
- FindBugs [Fin 08] : 0 design rules.
- Jamit [Jam 08]: 0 design rules.
- JCSC [JCS 08]: 0 design rules.
- JDepend [JDe 08]: 7 design rules of 7, all of them based on metrics).
- DoctorJ [Doc 08]: 0 design rules.

The reviews also revealed that almost all of these were based on metrics [Mar 04], rather than based on structural properties.

As regards commercial rule detection tools, lots of them implement their own rules, but also take advantage of using available open-source rules. As a logical consequence, they provide a bigger set of rules.

For the purpose of this research work, we only analysed tools providing a bigger set of rules. Therefore, the tool "AppPerfect" [App 08], implementing about 800 rules, from which only 8 refer to real design defects, and the tool "CodePro" [Cod 08], implementing over 950 rules, from which about 4 rules focus on design problems as understood in this paper were analysed.

Both in the case of open source and commercial tools, a detailed analysis concerning this topic has been carried out. As a result of this work, the following table shows an example of the extracted information. The whole analysis is not presented for legibility reasons.

	Checkstyle	PMD	AppPerfect	CodePro	...
No abstraction between interface and implementation					
A class rejects inherited properties	X		X		
Fan Out complexity	X				
Superclass is concrete					
Excessive public method count		X	X		
Convert class to interface				X	
A class knows its superclass					
Split interface when complex					
...					

Tab. 2 *Design rules implemented by each tool*

4.3 Interpreting the literature review results

From our point of view, the study points out two main research gaps concerning the state of the art:

In the first place, there is a lack of real design rules detection tools right across the market. The tools presented here focus mainly code-level problems, such as style, syntax or debugging issues. PMD, Checkstyle of AppPerfect tools certainly imply a valuable help for programmers, but in general doesn't provide any help to software designers or architects.
Secondly, the analysis shows that very few research proposals concerning design defect detection have been commonly used by the industrial community. In our opinion this situation reveals that more concrete and usable proposals need to be still presented in order to make the automated design validation a common practice.

However, note that even if it is true that there is the aforementioned lack of connection between industrial and research fields with respect to design rules, it is equally the case that open source and commercial frameworks are unavoidable sources of knowledge and are at the same time a faithful and clear reflection of the industrial practice at this level.

5 A new approach

5.1 The research gap

To throw light on the issue being discussed here, let's focus on a real case of study. At the moment, there is a new quality framework being defined at the Computing Area of the Traffic Division (Spanish Ministry of Internal Affairs). The new proposed framework aims to check every product delivered by software providers (including analysis, detailed design and code), analyzing its compliance according to a given set of quality standards. In this context, a common waterfall lifecycle is used at the company for a great amount of large projects, and any kind of automated checks would be really welcomed.

By means of using the tools discussed here, there is no possibility of checking design models, because as exposed in this paper, only coding bugs and conventions defects are addressed. Thus, a huge amount of resources need to be wasted in order to manually perform design rule checks. Moreover, design checks are not performed in a systematic way, because those depend on the availability of resources.

In addition, for really large systems, those tools provide several thousands of rule violations, and no guidance of which rule violation is more important (and should be fixed in first place) is provided. Those tools are difficult to apply to legacy systems, especially to large ones. From this point of view, we think that there is a still unwritten relative importance added by each rule.

Several classifications of rules are provided by the different tools, such as SUN recommendations rules, JDBC rules, EJB rules, naming convention rules or exception rules. However, those classifications are not based on software engineering concepts, such as rework needed or phases of appliance, but rather on the J2EE specification or coding recommendations.

5.2 A complete set of rules

To address this gap, in the context of this research we identified and classified all the related previous work, both from the research community and with reference to the commercial and open source solutions. A fresh rule classification approach is presented here. It mainly depends on the rework typically needed to fix a violation and hence on the moment of the software development where the check must be performed. Basically, we have classified existing rules into four types:

- Coding Style Rules Widely implemented for the existing tools. Depending on programming language.
- API-Based Rules, such as JDBC or EJB rules – also widely implemented. They are mainly related to the API used or to implementation and language.
- Design Rules. Low level design rules.
- Architectural Rules. High level design rules.

The figure 1 shows the relationship between those types of rules and the rework needed to fix a violated rule.

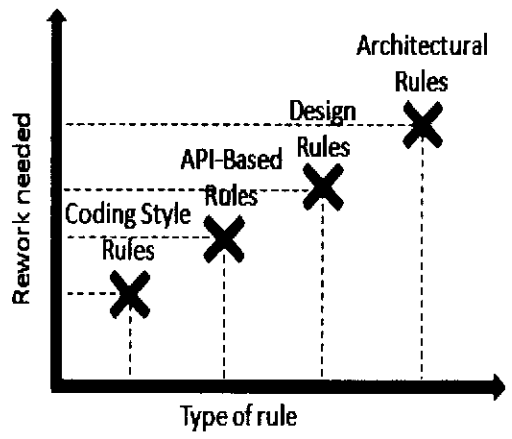


Fig. 1 *Rework needed for each type of rule*

5.3 The Value-Based Approach

The concept of value has been widely used in other fields of knowledge, mainly in the context of economics, but it is really a new concept in the software engineering field. Recently, a value-based software engineering (VBSE) agenda has emerged [Boe 05], whose aim is to integrate value considerations into the full range of existing and emerging software engineering principles and practices. One of the major elements of this agenda is value-based design and development, which involves techniques for ensuring that the system's objectives and value considerations are inherited by the software's design and development practices [Cab 07].

In contrast with the value-based approach we can find the traditional "neutral" approach, where each rule has the same importance. In this context, a system needs to be compliant with every rule to be considered correct.

The defect detection tools which are available were developed in a neutral way. In every tool we examined, a large amount of rules (as commented, mostly coding style and API-based) can be found, but no guidance is provided about which rules are more important than others, or which rules should be applied first. We propose to provide such guidance, checking those rules which avoid greater rework in early stages of development. In other words, we will ensure maintenance by checking the most valuable rules first.

Figure 2 depicts the process of application of the different set of rules. Note that each set of rules should be applied at a different development stage.

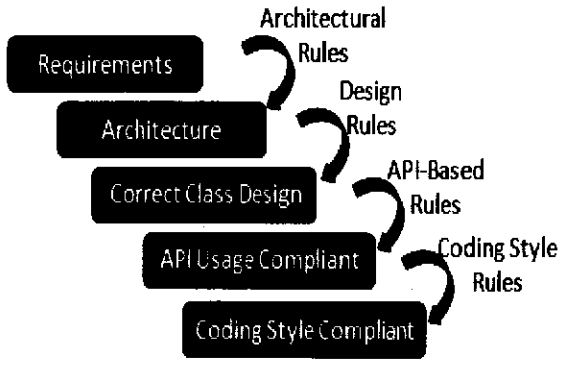


Fig. 2 *Application of rules through design stages*

Our proposal includes this classification of rules and also specifies in which order they should be applied.

5.4 Introducing Change Prone Considerations

The previous sub-section presents how most available rules can be classified and applied at four stages: Architecture, design, API low level design and coding style compliance. In short, it provides a first level of prioritisation.

Those four groups are, however, eventually big enough to propose a second level of value-based prioritisation. Sooner or later, we will encounter the problem that, for example, we do not know which of the available design rules will earn most value in our concrete system.

It is a hard task to propose a general solution (applicable to every project), where one particular rule may be more important than another belonging to the same group. Instead, in [Cab 07] we proposed to create a chart, taking into account the probability of change and the relative importance.

Figure 3 shows the process of prioritisation of each improvement, where each "X" represents a rule violation. We will assign a higher value to the improvements that have a greater probability of change.

Many techniques of estimation of change have been presented in the literature [Cha 05]. The idea is to use these techniques to assign a relative value to each implementation. This will allow us to order and prioritise a large amount of violated rules.

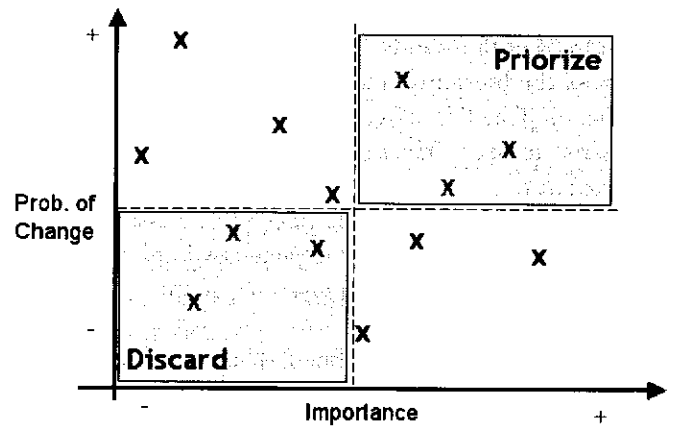


Fig. 3 *Applying Probability of change to Prioritisation, from [Cab 07]*

The main objective is to focus resources and effort on improving first of all those artifacts that will need to be changed in the future. In this way, we ensure better maintainability precisely where the greatest maintenance work needs to be done. This is how the value-based approach can be applied to software design, in order to improve design value (reduce maintenance costs).

Currently, some research work is being carried out in parallel with the one described above. Conducted by this research group, it concerns the application of

the probability of change to rule violation; moreover, a tool supporting the process is being developed.

6 Conclusions and Future Work

This work targets three main objectives:

In the first place, it gathers all the information that is presently available concerning automatic rule tool detection, including sources of rules and tools that have already been developed.

Secondly, it analyses and classifies the complete set of rules into different categories and sets the basis for a methodological application, taking into account the rework needed to fix a given rule violation.

Thirdly, it introduces the value-based approach. This uses the probability of change of the different artifacts to achieve an "earned value" for the fixing of each rule violation. Ordering the relative importance of each rule violation allows engineers to optimise their resources, and to focus on the most important design problems first.

The information collected in this research work, along with the rule detection tool provided, is now being tested out in case studies in the Spanish Ministry of Internal Affairs, at the Traffic Division. We are applying and contrasting the considerations set out above on several high performance J2EE systems. These systems perform hundreds of thousands of transactions every day.

Note that across the literature, most research proposals are tested on open source projects, but as [Che 04] state, open source projects are significantly different from industrial projects. Hence the significance of the results of the case studies being carried out.

During the case studies, the available rules were classified into coding, API, and design rules. The set of rules is also completed by implementing various other design and architectural rules detected in some of the sources we have commented (see section 3 for further details).

The methodology has also been defined. The main idea is to improve rule checking as much as possible at design stages, instead of waiting until coding and release phases. At the moment, we are testing and retrieving data in order to measure the productivity improvement obtained by complying architectural and design rules at earlier phases of the development (design phases), when only UML models are available.

In a second phase of our research, we plan to extend the tool to introduce value considerations; that is, we will provide mechanisms for prioritising the rule violations, based mainly on the probability of change. As explained before, a fixed rule will earn more value for the maintainability of the system if the violation itself concerns a piece of code which is expected to change often. On the other hand, if we can estimate that a piece of code will never change, the improvement of rule violations in this code will probably not be worth the trouble [Cab 07]. This work will be integrated in the same plug-in, obtaining an integral guidance of the order of importance of the refactorization.

We also plan to conduct several case-studies to evaluate empirically the contribution of the new approach proposed in this paper.

7 Acknowledgements

This work has been partially supported by the ESFINGE project (Ministerio de Ciencia y Tecnología (TIN 2006-15175-C05-05), the MELISA project (PAC08-0142-3315), Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, and the IDONEO project (PAC08-0160-6141). Junta de Comunidades de Castilla-La Mancha.

References

[Moh 06] N. Moha, Y.-G. Guéhéneuc and P. Leduc: Automatic Generation of Detection Algorithms for Design Defects. In 21st IEEE International Conference on Automated Software Engineering (ASE'06), IEEE Computer Society, Tokyo, Japan. 2006

[Moh 07] N. Moha: Detection and Correction of Design Defects in Object-Oriented Designs. In Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'07), ACM Press, Montréal, Québec, Canada, pp. 949-950. 2007

[Mar 04] R. Marinescu: Detection strategies: Metrics-based rules for detecting design flaws. In 20th International Conference on Software Maintenance (ICSM'04), IEEE Computer Society, Chicago Illinois, USA, pp. 350–359. 2004

[Gid 84] N. Giddings and T. Colburn: An automated software design evaluator. In ACM'84 Annual Conference, ACM Press, pp. 109-115. 1984

[Wie 06] G. Wiederhold: What is your Software Worth? Communications of the ACM, 49, 65-75. 2006

[Gar 05] J. Garzás and M. Piattini: An ontology for micro-architectural design knowledge. IEEE Software Magazine, 22, 28-33. 2005

[Mor 07] C. Morgan, K. D. Volder and E. Wohlstadter: A Static Aspect Language for Checking Design Rules. In 6th International Conference on Aspect-Oriented Software Development, Vancouver Canada., pp. 63 – 72. 2007

[Sah 00] H. A. Sahraoui, R. Godin and T. Miceli: Can Metrics Help to Bridge the Gap Between the Improvement of OO Design Quality and Its Automation? In International Conference on Software Maintenance (ICSM'00), IEEE Computer Society, San José, CA, USA pp. 154. 2000

[Mun 05] M. J. Munro: Product Metrics for Automatic Identification of "Bad Smell" Design Problems in Java Source-Code. In 11th IEEE International Software Metrics Symposium (METRICS '05), IEEE Computer Society, Como, Italy pp. 15. 2005

[Fow 03] M. Fowler: UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, MA, USA. 2003

[Egy 07] A. Egyed: Fixing Inconsistencies in UML Design Models. In 29th International Conference on Software Engineering (ICSE'07), IEEE Computer Society, Minneapolis, MN, USA, pp. 292-301. 2007

[Kic 01] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W. G. Griswold: An overview of AspectJ. Computer Science, 2072, 327–355. 2001

[Bro 98] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. M. III and T. J. Mowbray: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, 1998

[Sha 90] M. Shaw: Prospects for an Engineering Discipline of Software. IEEE Software Magazine, 7, 15-24. 1990

[Rie 96] A. J. Riel: Object-Oriented Design Heuristics, Addison-Wesley Professional, Boston, MA, USA. 1996

[Gam 95] E. Gamma, R. Helm, R. Johnson and J. Vlissides: Design Patterns, Addison-Wesley Professional, 1995

[Mar 96] R. C. Martin: The Dependency Inversion Principle. C++ Report, 8, 61–66. 1996

[Pes 97] C. Pescio: Principles Versus Patterns. Computer Science, 30, 130-131. 1997

[Fow 99] M. Fowler: Refactoring, Addison Wesley Professional, 1999

[Sku 96] S. Skublics, E. J. Klimas and D. A. Thomas: Smalltalk with style, Prentice Hall, 1996

[Tra 99] G. H. Travassos, F. Shull, M. Fredericks and V. R. Basili: Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality. In OOPSLA, ACM, Denver, CO, USA. 1999

[Chi 94] S. Chidamber and C. Kemerer: A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 476-492. 1994

[PMD 08] PMD: http://pmd.sourceforge.net/. 2008

[Che 08] Checkstyle: http://checkstyle.sourceforge.net/. 2008

[Fin 08] FindBugs: http://findbugs.sourceforge.net/. 2008

[Jam 08] Jamit: http://www.ovmj.org/jamit/index.html. 2008

[JCS 08] JCSC: http://jcsc.sourceforge.net/. 2008

[JDe 08] JDepend: http://clarkware.com/software/JDepend.html. 2008

[Doc 08] DoctorJ: http://doctorj.sourceforge.net/. 2008

[App 08] AppPerfect: www.appperfect.com/products/codeanalyzer.html. 2008

[Cod 08] CodePro: www.instantiations.com/codepro/analytix/about.html. 2008

[Boe 05] B. Boehm: Value-Based Software Engineering: Overview and Agenda. In Value-Based Software Engineering Springer, Heidelberg, Germany, pp. 3-14. 2005

[Cab 07] D. Cabrero, J. Garzás and M. Piattini: Maintenance Cost of a Software Design. A Value-Based Approach. In 9th International Conference on Enterprise Information Systems (ICEIS), Funchal, Madeira. Portugal, pp. 384-389. 2007

[Cha 99] M. A. Chaumun, H. Kabaili, R. K. Keller and F. Lustman: A Change Impact Model for Changeability Assessment in Object-Oriented Software Systems In European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA pp. 130. 1999

[Sha 07] A. R. Sharafat and L. Tahvildari: A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems. In International Conference in Software Maintenance and Reengineering, IEEE Computer Society, Amsterdam, pp. 27-38. 2007

[Tsa 05] N. Tsantalis, A. Chantzigeorgiou and G. Stephanides: Predicting the Probability of Change in Object-Oriented Systems. IEEE Transactions on Software Engineering, 31, 601-614. 2005

[Che 04] K. Chen, S. R. Schach, L. Yu, J. Offutt and G. Z. Heller: Open-Source Change Logs. Empirical Software Engineering, 9, 197-210. 2004