# Better, Faster, and Cheaper: What is Better Software?

Burak Turhan
Department of Information Processing Science
University of Oulu
90014, Oulu, Finland
burak.turhan@oulu.fi

Çetin Meriçli
Department of Computer Engineering
Boğaziçi University
34342, Istanbul, Turkey
cetin.mericli@boun.edu.tr

Tekin Meriçli
Department of Computer Engineering
Boğaziçi University
34342, Istanbul, Turkey
tekin.mericli@boun.edu.tr

## ABSTRACT

**Background**: Defects are related to failures and they do not have much power for indicating a higher quality or a better system *above the baseline* that the end-users expect. Nevertheless, defect counts are commonly used as measures to capture the quality of a system. Further, the statistical association between internal design metrics and quality in terms of defects has been shown in previous work.

**Aims**: Our goal is to conduct an initial data analysis for our longer-term goal of investigating whether there exist a similar relationship between the internal design characteristics of a system and quality perception of end-users.

**Method**: We carry out an exploratory case study in robotic soccer domain. We propose a quality measure derived from the performances in the robotic soccer competitions. Then, we compare the design characteristics and quality levels of two cases. In particular, we compare the two different implementations of a system in terms of their design metrics and their overall quality as measured by game scores.

**Results**: There are statistically significant differences between the two implementations in six out of seven design metrics. The implementation that has a much larger code base shows significantly better design characteristics in terms of complexity. We see significant differences in the quality levels as well.

**Conclusions**: We observe that the implementation that has achieved a better performance in the tournament also has better design characteristics for almost all attributes. However, our analysis does not include enough data points to investigate any association between internal design metrics and the proposed quality metric. Another restriction is that the proposed metric is domain specific. In future work, we plan to extend our analysis using additional implementations of the same system to test the consistency of the results.

## Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

## General Terms

Theory

## Keywords

Software quality, design metrics, case study, robotic soccer

## 1. INTRODUCTION

Software research and practice share a common goal, which is to develop ways of achieving better, faster, and cheaper software. However, "better" is a fuzzy term usually referring to higher quality in software context, where quality is yet another ambiguous term. Shepperd defines software quality as a multi-faceted characteristic not only consisting of many factors, but also perceived differently by distinct stakeholders [22]. There are also different views on software quality such as the quality of the processes and the traditional view of product quality, which further branches to subclasses like internal and external quality. Regardless of these issues, software quality is often defined in terms of defects in the system; a measure whose objectivity is also questionable. Furthermore, defects are surrogate measures of quality [10], whose absence indicate lack of problems rather than superiority of one system over the other. Unfortunately, using surrogate measures for software quality is somewhat unavoidable. Hamlet states [8]:

> *"...surrogate quality measures are needed, and they must be validated against the real properties we care about - actual quality - before they can be used..."*

In this sense, defects may only capture quality from the perspective of the stakeholder that develops the system, (i.e. releasing a non-problematic product that is easy to maintain), but they do not indicate much regarding the quality perception of end-users. Although defects certainly decrease the quality of a product, their absence does not raise the quality bar above the baseline (e.g., defects can be an indicator of failure at best).

So, what are the real properties that end-users care about? A possible answer, which we use in this study, is the properties that capture a system's ability to fulfill the purpose that it is developed for. It is the basic right and assumption of end-users not to have defects in the products they use. Even in the existence of defects, most users learn to live with them until they get fixed, given that the rest of the system is still available and valuable for the end-users. Considering the current wave of agile and particularly lean philosophies for software development, the concept of *value* creation is of highest importance and there is no better way of creating value than providing exactly what the customer needs. Therefore, actual quality hides in the way a system does what it is intended to do. This perspective on quality is of particular importance in cases where there exist alternatives of readily-available software for the customer to choose from. It also becomes important for business stakeholders to reach their business goals in a competitive environment.

From the research perspective, there is an extensive body of literature devoted to predicting quality (i.e. defects) using product (i.e. complexity and OO metrics) and/or process metrics (i.e. code churn). In other words, the validity of such product and process metrics as indicators and predictors of software quality as perceived by the stakeholder that developed the system (i.e. in terms of defects) have been shown. In this paper, we conduct a case study, where we propose a domain specific metric for overall software quality as perceived by the end-users (i.e. the extent to which a system fulfills its purpose) and compare the designs of two cases.

We believe that existing approaches to the quality puzzle employing defect measures only show how *not bad* a system is. We compare the designs of two software systems and assess how *good* they are in terms of fulfilling the end-users' expectations.

## 2. BACKGROUND

### 2.1 Metrics and Defects

Product metrics have been extensively used in software quality research regarding their power for indicating and/or predicting quality, though the basic assumption is that quality is associated with defects.

Fenton and Ohlson made an extensive analysis of defects in two releases of a commercial product using product metrics [6] and report that most of the defects reside in a small number of modules (i.e. Pareto principle [7]). They also report that they have found no evidence regarding the relation between defects and the size/complexity of the modules and they observe similar defect densities in the two versions. Their study is replicated by Andersson and Runeson in order to verify/refute Fenton and Ohlson's findings in three products of a different domain [1]. Their replication confirmed the results of Fenton and Ohlson. The same fault distribution has been observed by Ostrand, Weyuker, and Bell in very large scale telecommunication projects from AT&T [3, 17, 16, 18]. Koru and Tian, in their analysis of two IBM and four Nortel Networks products, also report that most defect prone modules are not necessarily the ones with the highest complexity measures [11]. On the other hand, Koru and Liu's further research strongly argues that smaller modules are proportionally more defect-prone than larger modules [12]. Demonstrating the usefulness of product metrics for predicting defects, Menzies *et al.* constructed high performance models on NASA MDP projects [14]. Further, Turhan *et al.* showed the applicability of their approach across different development sites [25].

Considering object oriented design metrics introduced by Chidamber and Kemerer [4], different studies performed experiments to validate their use [2, 24, 5, 21]. Subramanyam and Krishnan found them useful cautioning that the predictive power of these metrics vary in different programming languages [24]. However, El Emam *et al.* argued that when the effect of class size is also considered, only 4 of these metrics are related with defects and just two of them are useful for building predictors [5].

Further, Jiang *et al.* compared the predictor performances that are learned from design metrics, static code features, and both on 13 NASA projects, concluding that combination of these metrics are able to predict more accurately than their individual use [9]. Same conclusion is also achieved by Zhao *et al.* in their analysis of a real time telecommunication system [26].

In summary, previous research investigates the association between product metrics and defects with a focus on quality from development point of view. On the other hand, we investigate the end-users perspective of quality. In order to avoid the variations in the perception of end-user quality, we perform our case study in robotic soccer domain, where the overall success can be quantitatively and objectively assessed via the results of the soccer games.

### 2.2 Robotic Soccer

The idea of robot soccer was first mentioned by Alan Mackworth in his paper "On Seeing Robots" [13] in 1992. Around the same time in United States, Manuela Veloso and her student Peter Stone from Carnegie Mellon University had been working on soccer playing robots and a separate group of researchers from Japan were discussing on long term grand challenge problems for robotics.

These discussions eventually led to further investigation of soccer as a platform to foster science and technology, and a robotic competition named the Robot J-League was launched by Japanese researchers including Minoru Asada, Yasuo Kuniyoshi, and Hiroaki Kitano in 1993. The competition received overly positive reactions from the researchers around the world just within a month after its announcement and it has been renamed to Robot World Cup Initiative, or shortly "RoboCup". Several demonstration games were played in 1994, 1995, and 1996 at the leading conferences of AI like AAAI, JSAI, IROS, and IJCAI.

First RoboCup event was held in 1997 in Japan and since then RoboCup has been the largest robotics event in the world [19]. The ultimate goal of RoboCup was set as follows:

> *"By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup."*

## 3. CASE STUDY

### 3.1 Context

Currently, RoboCup employs different robot soccer categories addressing different research challenges. The RoboCup Standard Platform League is one of them, in which rather than building custom robots for playing soccer, each participant has to use a standard, commercially available robotic platform determined by the RoboCup Federation [19]:

*"...In the league, all teams use identical (i.e. standard) robots. Therefore the teams concentrate on software development only, while still using state-of-the-art robots. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers..."*

This constraint forces researchers to focus on *software development* that can run real time on an embedded platform. Therefore, it is a software challenge rather than a hardware challenge.

From its establishment in 1998, until 2008, Sony Aibo robotic dog was the standard platform. Since 2008, it's been replaced by Nao humanoid robot manufactured by a French company named Aldebaran Robotics [23]. Our case study is performed on the robotic dogs. Sony AIBO ERS-7, a successor model for ERS-210 used in previous RoboCup competitions, is a robotic dog with 18 degrees of freedom, a color camera, two infrared distance sensors, and a 3-axis accelerometer. It has a MIPS 6000 processor running at 576 MHz. It runs a custom real-time operating system named APERIOS, also developed by Sony. Figure 3.1 shows both ERS series robotic dogs.

**Figure 1: Standard platforms for 2005 competitions: a) Sony AIBO ERS-7 b) AIBO ERS-210.**

### 3.1.1 Main Software Components

Although the software systems of the competing teams differ in terms of the implemented algorithms and the architectural design, most of them consists of five major components that are developed in C++ using provided API's:

- **Vision:** Vision component is responsible for perceiving the environment and recognizing the important objects (e.g., goals, robots, landmarks, the ball, the field, etc.) and their relative positions with respect to the robot using the color camera mounted on the head of the AIBO. The objects on the field are color coded (i.e., the goals are yellow and blue, the landmarks are yellow-blue and blue-yellow, the field is green with white lines, the ball is orange, and the robots on the teams wear red and blue jerseys). The camera of the AIBO provides 30 images per second and it is of crucial importance to develop a vision system that can recognize the objects using their color and shape, and compute their relative bearings and distances to the robot at the full frame rate.

- **Self Localization:** In order to decide on which action to perform, the robot should have the information of its whereabouts on the field. The self localization component is responsible for inferring the robot's position on the field using the visual information (i.e., bearings and distances of the perceived landmarks on the field) and the proprioceptive sensing of the robot's displacement reported by the locomotion component. There are various methods for localization including geometry-based methods (e.g., triangulation) and probabilistic modeling (Monte Carlo localization, Markov localization, and Kalman filter based approaches).

- **Locomotion:** The locomotion module is responsible for generating walk and kick motions which are essential to play soccer. It converts the high level command received from the planning and behaviors component into a series of joint angles to be set at each execution cycle. Most teams use different implementations of inverse kinematics based omnidirectional walking methods.

- **Communication:** AIBOs are equipped with wireless ethernet cards which allow them to connect to a standard IEEE 802.11a/b wireless network. The robots of the same team utilize this wireless connection ability to exchange both the perceived world information (i.e., the positions of the ball, the robot itself, and the other perceived objects), and the planning and behavior related messages, especially for assigning different behaviors to different robots to prevent more than one robot from rushing to the ball at the same time.

- **Planning and Behaviors:** Planning and Behavior component is where the "intelligence" of the whole system lies in. It is responsible for deciding which action to take based on the perceived environment, the internal state of the robot, and the exchanged information with the teammates through the communication component. The decided action (e.g., walking towards a direction at a certain speed, performing a kick, or looking at a point) is then passed to the locomotion module.

### 3.1.2 Team Descriptions

The first case team, which we refer to as Team A in the rest of this paper, is the GermanTeam. GermanTeam is a joint effort of four German universities, namely Humboldt-Universität zu Berlin, Universität Bremen, Technische Universität Darmstadt, and Universität Dortmund. A total of four faculty members, 8 PhD students, and 43 undergrad students have contributed to the team in 2005 competitions [20]. The GermanTeam has been able to achieve a consistently high performance since their first competition in 2001 and their code base has been adopted by nearly half of the teams in the league since then.

The second case team, which we refer to as Team B, is TWaves. Twaves is the RoboCup SPL team of Tokai University, Japan. The team consists of two faculty members, two technicians, and seven students. TWaves uses their own code base written from scratch. Started their development in mid-2004, TWaves is a relatively inexperienced team compared to the GermanTeam.

**Table 1: Hypotheses. (TA: Team A, TB: Team B, *lcom: lack of cohesion in methods,* rfc: response for a class, *wmc: weighted methods per class,* cbo: coupling between objects, *noc: number of children,* dit: depth of inheritance tree).**

| metric | $H_0$ | $H_A$ |
|---|---|---|
| cc | $CC_{TA} = CC_{TB}$ | $CC_{TA} \neq CC_{TB}$ |
| lcom | $LCOM_{TA} = LCOM_{TB}$ | $LCOM_{TA} \neq LCOM_{TB}$ |
| rfc | $RFC_{TA} = RFC_{TB}$ | $RFC_{TA} \neq RFC_{TB}$ |
| wmc | $WMC_{TA} = WMC_{TB}$ | $WMC_{TA} \neq WMC_{TB}$ |
| cbo | $CBO_{TA} = CBO_{TB}$ | $CBO_{TA} \neq CBO_{TB}$ |
| noc | $NOC_{TA} = NOC_{TB}$ | $NOC_{TA} \neq NOC_{TB}$ |
| dit | $DIT_{TA} = DIT_{TB}$ | $DIT_{TA} \neq DIT_{TB}$ |
| perf | $PERF_{TA} = PERF_{TB}$ | $PERF_{TA} \neq PERF_{TB}$ |

### *3.1.3 Tournament Structure*

A total of 24 teams compete in the RoboCup SPL. The tournament starts with a round robin of 8 groups with three teams in each group. The teams ranked first directly proceed to the second round robin games, while the second and the third place teams play against the opposite team of the next group (i.e., the second place team of the first group plays against the third place of the second group, the second place of the second group plays against the third place of the first group and so forth). The second round robin consists of four groups with four teams in each group. The first and the second place of the groups proceed to the quarterfinals where the first and the second place teams of groups 1 and 2 play against the second and the first place teams of groups 4 and 3, respectively. In the semifinals, the winners of the first and the fourth, and the second and the third teams play against each other, and the eventual winners of the semifinal games meet at the final game.

## 3.2 Hypotheses and Analysis Methods

In order to compare the two implementations of the same robotic soccer system, we use the cases in our analysis as the independent variable (i.e., Team A vs. Team B), the code artifacts as the dependent variables (specifically cyclomatic complexity and CK metrics as the design measures), and the performances of the teams as their quality indicators. We postulated eight hypotheses corresponding to each dependent variable. Table 1 shows a list of these hypotheses. For each hypothesis, the corresponding null and the alternative hypotheses are:

$\mathbf{H_0}$ : There is no difference between the two systems.

$\mathbf{H_A}$ : There is a difference between the two systems.

Before applying statistical tests for hypothesis testing, we checked for normality in each distribution. We observed that none of the metrics are normally distributed in either groups. Therefore, we used a non-parametric test, i.e., 2-tailed Mann-Whitney U Test, to check for statistical differences. In all tests we use the significance level $\alpha = 0.05$. All analyses are performed in Matlab R2007a.

## 3.3 Data Collection

In order to differentiate the overall quality of the two systems, we used their robotic soccer competition performances at the RoboCup Standard Platform League in 2005. Among the 24 competing software, Team A won the competition after playing a total of eight matches, with an average of more than four goals scored per game and less than one goal received. On the other hand Team B was eliminated at the intermediate round after 3 matches, with an average of more than three goals received per game and less than one goal scored. While Team A won seven out of eight games, Team B lost all three games. Although we focus on only a single tournament for determining the software that is better in terms of the outcomes, a longitudinal view on both teams also reveal the same pattern: Team A has been consistently successful in the tournament over the years, while Team B usually performed below the average.

**Table 2: Performances of the two teams.**

| metric | TA | TB |
|---|---|---|
| Standing (in 24) | 1st | 16th-24th |
| Avg. goals scored | >4 | <1 |
| Avg. goals conceded | <1 | >3 |
| Wins/draws/loses | 7/1/0 | 0/1/2 |
| Overall performance score | 3.25 | -3.00 |

Our proposed metric for capturing quality is an obvious one for the robotic soccer domain: the average difference between the goals scored and goals conceded per game. Clearly, this metric favors wins over losses and it does not penalize the draws. Furthermore, this metric also favors wins with larger deltas. Since the number of games played by the teams may not be equal, we normalize the quality metric by the total number of games played. The quality metric for a team $TX$, indicated by $perf_{TX}$, is formalized in Equation 1:

$$perf_{TX} = \frac{1}{n}\sum_{i=1}^{n} gs_i - gc_i \tag{1}$$

where $n$ is the total number of games played by team $TX$, $gs_i$ is the goals scores in game $i$, and $gc_i$ is the goals conceded in game $i$. Please note that Equation 1 yields a single number for the overall quality of a team, and to obtain the data points we used in hypothesis testing we used the same equation without the summation term to come up with a sample population of match performances. A summary of team performances in the tournament is given in Table 2.

The source codes for some of the competing teams in the tournament are publicly available through their web sites. For our analysis, we use the two teams' codes with observable differences in their performances as described earlier. From the codes of the teams, we excluded the codes that do not run on the robots, i.e., helper applications for debugging, simulation, etc. We used a static analyzer tool (Predictive 3) for automated data collection from their public source codes. The descriptive statistics of the collected data are provided in Table 3.

## 4. RESULTS

In this section, we present the results of the hypothesis tests and discuss their implications regarding the implementations of the two teams. Please note that our hypothesis are 2-tailed and a rejection only indicates a difference between the two implementations. While presenting the results, we

**Table 3: Descriptive statistics.**

| metric | #data | | median | | min | | max | |
|---|---|---|---|---|---|---|---|---|
| | TA | TB | TA | TB | TA | TB | TA | TB |
| cc | 534 | 31 | 8 | 45 | 0 | 0 | 883 | 1228 |
| lcom | 534 | 31 | 3 | 9 | 0 | 1 | 83 | 52 |
| rfc | 534 | 31 | 7 | 20 | 0 | 0 | 133 | 81 |
| wmc | 534 | 31 | 3 | 18 | 0 | 0 | 70 | 56 |
| cbo | 534 | 31 | 1 | 0 | 0 | 0 | 31 | 9 |
| noc | 534 | 31 | 0 | 0 | 0 | 0 | 24 | 0 |
| dit | 534 | 31 | 0 | 0 | 0 | 0 | 3 | 0 |
| perf | 8 | 3 | 4 | -3 | 0 | -6 | 5 | 0 |

**Table 4: Results**

| $H_0$ | Reject | Direction | p-value |
|---|---|---|---|
| cc | **yes** | TA < TB | **<0.001** |
| lcom | **yes** | TA < TB | **< 0.001** |
| rfc | **yes** | TA < TB | **< 0.001** |
| wmc | **yes** | TA < TB | **< 0.001** |
| cbo | **yes** | TA > TB | **0.0115** |
| noc | no | TA = TB | 0.0929 |
| dit | **yes** | TA > TB | **0.0188** |
| perf | **yes** | TA > TB | **0.0189** |

refer to the medians to comment on the direction of the differences. A summary of the results is provided in Table 4.

**CC:** As a general design principle simplicity is preferred over complexity. Cyclomatic complexity is the number of possible program execution pathways and values above 10 are not recommended. Higher values indicate complex designs that are difficult to understand and maintain. The test for the null hypothesis $H_0 : CC_{TA} = CC_{TB}$ using Table 3 data yields a p-value less than 0.001 and we reject the null hypothesis, in favor of Team A. Therefore, Team A's implementation has better design characteristics than Team B's, in terms of cyclomatic complexity. One interesting observation is that even though Team A has a code base nine times larger than Team B, their class structures are simpler, which indicates that complexity due to more lines of code may be avoided by good designs.

**LCOM:** Cohesiveness is a desirable design attribute that enables encapsulation. Therefore, low LCOM (i.e., lack of cohesion in methods) values indicate better designs whereas higher values suggest the need for splitting the class into subclasses [4]. The test for the null hypothesis $H_0 : LCOM_{TA} = LCOM_{TB}$ using Table 3 data yields a p-value less than 0.001 and we reject the null hypothesis in favor of Team A. Therefore, Team A's implementation has a better design than Team B's, in terms of cohesiveness.

**RFC:** RFC counts the number of possible methods that can be executed in response to a message received by an instance of that class. Lower values are desired, since higher values indicate increased communication complexity between classes [4]. The test for the null hypothesis $H_0 : RFC_{TA} = RFC_{TB}$ using Table 3 data yields a p-value less than 0.001 and we reject the null hypothesis in favor of Team A. Therefore, Team A's implementation has better characteristics than Team B's, in terms of the communication complexity between classes.

**WMC:** WMC captures the total complexity of methods defined in a class. Higher values indicate increased complexity, application specific implementation with low chances of reuse [4]. The test for the null hypothesis $H_0 : WMC_{TA} = WMC_{TB}$ using Table 3 data yields a p-value less than 0.001 and we reject the null hypothesis in favor of Team A. Therefore, Team A's implementation has different characteristics than Team B's, in terms of the inner complexity of classes.

**CBO:** Coupling of two classes occur when one class uses the methods of the other. CBO is the number of classes, to which a class is coupled to. Higher coupling clearly increases inter-object complexity and maintenance effort and is against modular design principles [4]. The test for the null hypothesis $H_0 : CBO_{TA} = CBO_{TB}$ using Table 3 data yields a p-value of 0.0115 and we reject the null hypothesis. In this case, the direction is opposite to the common pattern, i.e., in favor of Team B. However, Team A's coupling scores are not too high (i.e., median value of 1) to conclude a relatively worse design. The lower values observed in Team B's code are probably due to their design that implements most functionality within single large classes that do not communicate with others.

**NOC:** NOC is the count of immediate sub-classes that inherit a class. While high-values suggest reuse through inheritance, it also increases the likelihood of improper abstraction and misuse of inheritance [4]. The test for the null hypothesis $H_0 : NOC_{TA} = NOC_{TB}$ using Table 3 data yields a p-value of 0.0929 and we fail to reject the null hypothesis. Therefore, Team A's implementation is no different than Team B's, in terms of the immediate use of inheritance. On the other hand, it is clear that Team B has not used inheritance at all (i.e., min and max values are 0), and the insignificant difference suggests a limited use of inheritance in Team A.

**DIT:** DIT denotes the level of the class in the inheritance structure. High values indicate a class being deeper in the inheritance tree and are not desirable since they indicate more design complexity due to increased number of inherited methods [4]. The test for the null hypothesis $H_0 : DIT_{TA} = DIT_{TB}$ using Table 3 data yields a p-value of 0.0188 and we reject the null hypothesis. Therefore, Team A's implementation has different characteristics than Team B's, in terms of the inheritance structure. Yet again, it is evident that Team B has not used inheritance at all and Team A has a maximum value of 3. This suggests that Team A has better utilized inheritance.

**PERF:** PERF scores are calculated as described in Section 3.3. Higher values indicate better performances in terms of goals scored and conceded. The test for the null hypothesis $H_0 : PERF_{TA} = PERF_{TB}$ using Table 3 data yields a p-value of 0.0189 and we reject the null hypothesis. Therefore, Team A's implementation is better than Team B's, in terms of the performance scores.

## 4.1 Limitations

One possible factor affecting the outcomes is the individual programming skills, experience and the ability to perform team work. The impact of cultural issues for software development is also applicable. Nevertheless, both implementations are done by a mixed team of graduate and undergraduate students led by one or more senior researchers, and we assume that the overall level of the individuals were equivalent in this sense. In terms of the number of total team members, Team A is five times larger than Team B. While this suggests the availability of more effort and resources for Team A, it also means increased communication and management complexity. Further, Team A carried out multi-site development whereas Team B members were collocated. Therefore, the size of Team A may also be considered as a disadvantage. Indeed, this may be the reason that Team A's code base is nine times larger (i.e., $90,000+$ vs. $10.000+$).

Both teams have passed certain qualification criteria, which are set and evaluated by a technical committee, in order to participate in the tournament like all 24 teams. Each year at least 30+ teams throughout the world apply for participating in the RoboCup Standard Platform League, and only 24 of them are selected by this committee. Each candidate team records a gameplay video showing the team's abilities for achieving qualification criteria and shares it with the technical committee. Therefore, both teams fulfill a baseline quality criteria and are comparable in this sense.

We have used a single tournament's results for differentiating between the quality of two systems. It can be argued that these game results may be due to factors related to chance. Nevertheless, a real-time, dynamic, and non-deterministic environment is an inherent property of the domain that these software are developed for; therefore, these facts should be accounted for in the design. Furthermore, the consistent short term (i.e., within tournament) and long term (i.e., across different tournaments) performances of both teams suggest that the risk of achieving these results by chance is minimum. Similarly, choice or development of certain algorithms for accomplishing the mostly AI-based tasks may directly affect the outcomes; however, we have no control over such critical design decisions.

While it is obvious for the robotic soccer domain, it may be a challenge for other domains to replace the quality measure to quantitatively reflect the end-user perspective. Therefore, our results are valid within the context of the case study that is described in earlier sections.

Furthermore, the number of data points in our analysis is too few to derive general conclusions. For evaluating commercial software, qualitative research methods may be more appropriate to capture end-user opinions or user experiences. The number of downloads or the size of the community may be an indicator of success for the system in fulfilling its purpose. However, these are surrogate measures that can easily be affected by other factors such as marketing strategies, regulations, and false recommendations through social networks.

## 5. CONCLUSIONS

We performed a comparative case study using two different implementations of a real-time, embedded software that competed in a robotic soccer tournament. In order to compare the quality of the two cases from an end-user point of view, we derived a metric from match scores rather than using defect counts. We found that the team that has achieved a significantly higher performance in the tournament also has significantly better design charateristics in terms of within and between class complexities, cohesiveness, and the use of inheritance.

The basic idea behind our analysis is that the perception of quality varies among different stakeholders. Using defects as a quality measure for the stakeholders who undertake the project is reasonable. However, it has no value for capturing end-users perception for quality above the baseline.

Though we evaluated cases with pure research goals and no commercial interests, we believe that the basic idea is applicable and could be customized for other business contexts. Furthermore, predictive models, which are tuned to end-user point of quality (similar to the models proposed by Menzies *et al.* that can be tuned for different business goals [15]), may be constructed if an association is detected between software artifacts and the quality outcome.

Our analysis currently do not have enough data points to investigate an association in the robotic soccer domain. However, most source codes and tournament results over the years are public and as future work, we plan to extend this analysis using those data. We believe that robotic soccer domain is a data rich and special environment for the community to examine.

## 6. ACKNOWLEDGMENTS

This research is partially supported in Finland by Tekes under Cloud-SW project, and in Turkey by Tubitak.

## 7. REFERENCES

[1] C. Andersson and P. Runeson. A replicated quantitative analysis of fault distributions in complex software systems. *Software Engineering, IEEE Transactions on*, 33(5):273–286, 2007.

[2] V. Basili, L. Briand, and W. Melo. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 22(10):751 – 761, Oct 1996.

[3] R. Bell, T. Ostrand, and E. Weyuker. Looking for bugs in all the right places. *ISSTA '06: Proceedings of the 2006 international symposium on Software testing and analysis*, Jul 2006.

[4] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476 – 493, Jun 1994.

[5] K. E. Emam, S. Benlarbi, N. Goel, and S. Rai. A validation of object-oriented metrics. *National Research Council of Canada*, Jan 1999.

[6] N. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *Software Engineering, IEEE Transactions on*, 26(8):797–814, 2000.

[7] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999. Available from http://citeseer.nj.nec.com/fenton99critique.html.

[8] D. Hamlet. An essay on software testing for quality assurance - editor's introduction. *Annals of Software Engineering*, pages 1–9, 1997.

[9] Y. Jiang, B. Cuki, T. Menzies, and N. Bartlow. Comparing design and code metrics for software quality prediction. *PROMISE '08: Proceedings of the 4th international workshop on Predictor models in software engineering*, May 2008.

[10] B. Kitchenham. Performing systematic reviews. *Keele University Technical Report TR/SE0401*, 2004.

[11] A. G. Koru, K. E. Emam, D. Zhang, H. Liu, and

D. Mathew. Theory of relative defect proneness. *Empirical Software Engineering*, October 2008.

[12] A. G. Koru and H. Liu. Identifying and characterizing change-prone classes in two large-scale open-source products. *JSS*, Jan 2007.

[13] A. K. Mackworth. On seeing robots. In *Computer Vision: Systems, Theory, and Applications*, pages 1–13. World Scientific Press, 1992.

[14] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 2007.

[15] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: Current results, limitations, new approaches. *Automated Software Engineering Journal*, 2010.

[16] T. Ostrand and E. Weyuker. The distribution of faults in a large industrial software system. *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, Jul 2002.

[17] T. Ostrand, E. Weyuker, and R. Bell. Where the bugs are. *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, Jul 2004.

[18] T. Ostrand, E. Weyuker, and R. Bell. Automating algorithms for the identification of fault-prone files. *ISSTA '07: Proceedings of the 2007 international symposium on Software testing and analysis*, Jul 2007.

[19] Robocup. www.robocup.org/.

[20] T. Röfer, T. Laue, M. Weber, H.-D. Burkhard, M. Jüngel, D. Göhring, J. Hoffmann, B. Altmeyer, T. Krause, M. Spranger, O. von Stryk, R. Brunn, M. Dassler, M. Kunz, T. Oberlies, M. Risler, U. Schwiegelshohn, M. Hebbel, W. Nisticó, S. Czarnetzki, T. Kerkhof, M. Meyer, C. Rohde, B. Schmitz, M. Wachter, T. Wegner, and C. Zarges. Germanteam 2005 team report. Technical report, University of Bremen, University of Humboldt, Technical University of Darmstadt, and Dortmund University, Germany, 2005. http://www.germanteam.org/GT2005.pdf.

[21] A. Schröter, T. Zimmermann, and A. Zeller. Predicting component failures at design time. *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, Sep 2006.

[22] M. Shepperd. *Foundations of software measurement.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1995.

[23] SPL. Robocup standard platform league www.tzi.de/spl/.

[24] R. Subramanyan and M. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Trans. Software Eng*, pages 297–310, 2003.

[25] B. Turhan, T. Menzies, A. Bener, and J. DiStefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, pages 540–578, 2009.

[26] M. Zhao, C. Wohlin, N. Ohlsson, and M. Xie. A comparison between software design and code metrics for the prediction of software fault content. *Information and Software Technology*, Jan 1998.