Value-Based Technical Debt Model and Its Application

Marek G. Stochel, Mariusz R. Wawrowski, Magdalena Rabiej
Motorola Solutions
Kraków, Poland
{marek.stochel, mariusz.wawrowski, magdalena.rabiej}@motorolasolutions.com

***Abstract*—The majority of software development today is being conducted in a value-neutral setting, where each functionality once being locked down as a part of software release is treated as equally important. This limited visibility of the real value perceived by customer inside software engineering organizational departments has significant consequences in the way the technical quality of the product is being evaluated and maintained. The relentless pursuit of efficiency in the software engineering domain requires a broader view of long-term economical consequences of any product-related decision. Technical debt typically is an internalized (engineering-based) assessment. We propose to expand the understanding and visibility of the technical debt by introducing a model driven approach to provide the means to assess the technical debt impact on perceived product quality parameters, such as codebase/design and architecture, engineering productivity, and finally the company's business return on the engineering investment. Furthermore, the case studies presented in this paper are focused on the application of the technical debt concept—how it could be identified, measured and what are the consequences of not managing it. The key principles of this concept were proved to be valid while evaluating the development of a major software system release. Finally, the need for balanced view for the technical debt management strategy is discussed, to ensure pay-off benefits are aligned with time-to-market expectations.**

Keywords*—*Technical Debt; Software Life Cycle; Software Economics; Software Development and Maintenance; Wisdom of Crowds; Value Based Software Engineering.

I. INTRODUCTION

The technical debt metaphor refers to the product's deficiencies, caused by shortcuts or incomplete engineering knowledge, which may speed up software development and delivery, but inevitably have their drawbacks and incur additional, delayed costs [7][8]. Unfortunately, due to its mostly internalized nature (engineering-based), technical debt is sometimes hardly recognized even by the very business unit where it was created and which is responsible for business return on the engineering investment. We claim that focus should be not only on the product/solution deficiencies, but also on how the value perceived by customers and profitability of a business unit are directly or indirectly impacted. The significant problem with managing technical debt and establishing pay-off strategy lies in the definition of the value to the customer/product/business return. Therefore, we believe that binding technical debt to the real value for the customer and hence the products return on investment (ROI) for the business (profitability) provides a better understanding of the potential consequences of not managing it. Additionally, awareness of the current and predicted product/solution condition is raised as a leading indicator.

The realization of value during software development was also the rationale for Boehm's Value Based Software Engineering concept [1][2], for which we propose a slight adjustment (marked by the dashed line) of the value realization feedback process to explicitly measure and manage technical debt (Fig. 1).

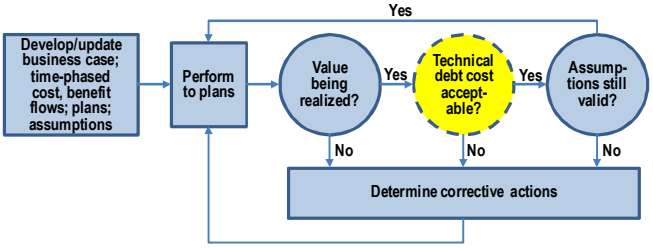

Figure 1. Value realization feedback process enhanced with technical debt cost assessment

As a consequence, technical debt is analyzed and discussed in this paper as a three-layered model, consisting of codebase/design debt, architecture debt, and portfolio debt. From this perspective long-term consequences for engineering and business are visible and understandable, leading to an established communication strategy across these functional areas. This aligns the effort spent in the oftentimes numerous departments of an organization during the planning and execution phases. Missing any of these critical layers of value-added granularity will provide a considerably less robust prioritized definition, and as a result, a distorted view of potential consequences of the technical debt the organization has already experienced and accumulated and may well continue to do so.

While introducing the model-based approach and the discussing underlying rationale, this paper provides also specific examples of how to calculate the technical debt value for subsequent layers, and how the total technical debt value may be coherently assessed. This discussion is supported by the results of two experiments. The first one was conducted to compare the subjective and objective ways of assessing the technical debt, the second—to understand the consequences of the missing knowledge about technical debt and related management strategy. Finally, we discuss

 ISBN: 978-1-61208-230-1

the consequences the proposed model incurs and suggest further areas of the research and investigation.

II. REVIEW OF LITERATURE

Although Cunningham refers to technical debt as a result of gaining knowledge during development [7][8], rather than deliberate choice, we should not neglect the fact that sometimes the latter is justifiable, as suggested by Brazier [3]. Some may suggest that technical debt should be tightly coupled only with the coding aspects that were caused deliberately, as it was referred by Fowler [12]. Furthermore he states that the question should be whether the technical debt metaphor is useful to discuss the design problems. This was supported by the technical debt quadrant concept referring to whether the debt is prudent or reckless, and running into it was deliberate or inadvertent [12]. Other authors expand the technical debt metaphor and propose the Modern Portfolio Theory [15] to calculate technical debt and the resulting associated risk. The risk based approach for software architectural decisions was also discussed by Fairbanks [10]. Following that path (usefulness of the metaphor), we propose to integrate these aspects tying technical debt to real business value in order to enhance communication and visibility.

III. THE MODEL

Understanding technical debt and how it changes over time should be performed consistently across the organization for both the product and the portfolio. In order to achieve this goal we propose a three-layered model aligned with a typical software product development lifecycle (Fig. 2). Note that codebase/design debt is aggregated by architectural debt, and this in turn is aggregated by portfolio debt.

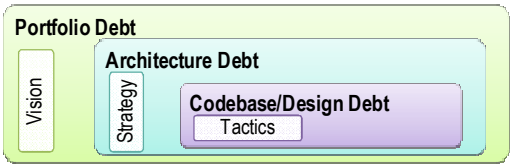

Figure 2. The three-layered model of technical debt

The rationale for this approach stems from a software development process analysis. The P-Diagram approach [17], usually employed for process risk assessment and impact analysis, was used for this purpose (Fig. 3). From this perspective technical debt can be thought of as an unintended byproduct of the software product development process as well as a noise factor of current development practices. Thus, we define technical debt as:

- Caused by the very nature of the software product development practices,
- Creating negative feedback loops, which impact all the intended outputs: product quality, engineering productivity, and organizational profitability,
- Being impacted by control factors, including—but not limited to—technical debt management strategy.

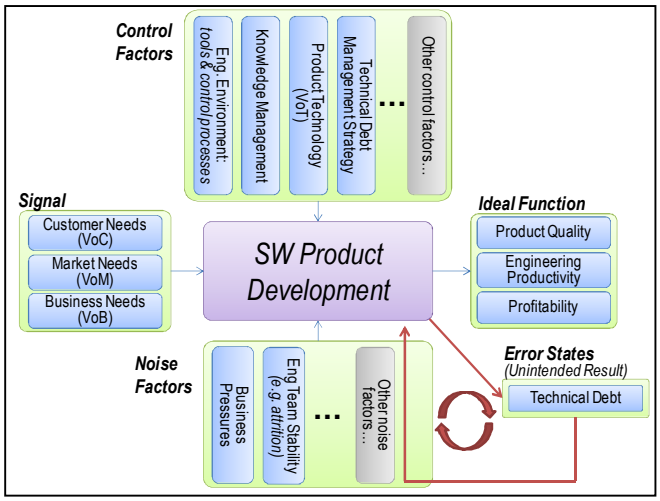

Figure 3. Technical debt in the context of software product development process

A. *Technical debt—"the process view"*

The software product development process constitutes the framework for technical debt assessment. Any omission of these aspects inevitably leads to an incomplete view of the technical aspects concerned and may result in a technical debt management strategy counterproductive from the organizational perspective.

The P-Diagram components (Fig. 3) are defined as follows:

Inputs (Signal): requests for a change in the product, driven by the customer (Voice of Customer—VoC), the market (VoM), and/or the business (VoB).

Outputs—Intended results (Ideal Function)

- Product Quality: product characteristics, as perceived by both customers and producers,
- Engineering productivity (Voice of Process, organizational capability): functionality delivered considering effort spent, may be presented as the engineering organization throughput (optimized cost, quality, schedule, scope according to baseline capability index),
- Profitability (marketing/sales): the business return on the engineering investment between and within projects. This can be measured as the percentage gross margin.

Outputs—Unintended results (Error States): the technical debt itself, which may be referred to as product and process characteristics, internally perceived by the organization (business unit).

Control factors: They comprise all factors, which influence and control the software development process, decreasing its variability thus improving predictability. For example:

- **Engineering development environment**: Tools, quality control processes, continuous software integration

 ISBN: 978-1-61208-230-1

- **Knowledge management/sharing**: Activities pertaining to information interchange, supporting seamless communication, and ensuring optimal work assignments
- **Technology/product management**: Activities controlling *technical inflation* [11] with focus on technology alternatives, ensuring robust testability
- **Technical debt management strategy**

Noise Factors: They are out of the direct control of the engineering organization, for example: business pressures, stability of engineering organization (attrition), and the technical debt itself.

B. Technical debt trend—assessment and prediction

Despite the technical debt value being calculated, it will naturally evolve. The question how to address these changes should be reflected in the technical debt management strategy. As technical debt from an unintended byproduct becomes the noise factor later (Fig. 3), the trend impacting predictability of intended results needs to be constantly monitored.

Understanding the measured impact and process capability we are able to predict whether a technical debt level is acceptable. The metric here should provide information how the product's current owner cares about maintenance and quality of the product, and how the value of technical debt changes over time according to the information provided by the portfolio and architecture teams. We also need to understand how value is being realized, and this information may be provided by a Value Based Software Engineering approach (presented earlier in Fig. 1). As a result, the current value of the technical debt may be determined in the broader context of the three-layered technical debt model. Additionally we are able to establish a threshold of how high the tech debt value might be when immediate action or a change in strategy is required and whether a technical debt pay-off is justifiable from the cost perspective.

IV. CODEBASE/DESIGN DEBT

The codebase/design debt is the most objective aspect of technical debt. It is a concept claiming that various source code quality indicators may be combined into one meaningful value easier to manage. This metric may be expressed as the effort required to change the existing codebase into an easy to maintain, well structured and testable code.

There are various aspects of both static and dynamic code characteristics that can be taken into consideration to create one meaningful technical debt metric. One can use code complexity measures provided by standard tools, such as Klocwork [16] or FXCop [13], to ensure that good coding practices are followed. Other tools may provide information that robust unit test coverage was assured. Enforcing well defined coding standards leads to a lesser number of defects introduced during integration to the version control engine, thus reducing the debt. Delaying integration of local software changes leads to a debt increase: the greater the delay, the less probability of an effortless integration. In principle, the set of measures and tools should explicitly reflect the software product's true characteristics in its intended environment.

A. Measurement Approaches

One approach, proposed for Java-based projects, is the Sonar approach [14]. Sonar is an Open Source Software quality management tool, which leverages the existing ecosystem of quality open source tools (for example: Checkstyle [4], PMD [20], Maven [18], and Cobertura [6]), to offer a fully integrated solution for development environments and continuous integration tools. Being accompanied by technical debt plug-in, Sonar is able to monitor static and dynamic metrics on the project and enforce coding best-practice rules, supporting defect prevention effort.

Another way to assess technical debt in codebase/design may be Wisdom of Crowds technique, which is based on the approach proposed by Surowiecki [22]. This technique has already proved successful in the prediction of defect distribution among system areas [23] and, although not mentioned explicitly, it was a major component of a proposed test case prioritization approach [24]. The software development team, if mature, can readily assess the code quality with precision and predictable results. However, in order to properly assess technical debt using the Wisdom of Crowds method, the following conditions must be met:

- Diversity of opinion—each person should have an opportunity to voice private information (even if it is his/her view of known facts),
- Independence—we have to assure that people can voice their concerns/opinions, and not repeat those of more senior, influential ones,
- Decentralization—we have to ensure the opportunity to present different perspectives, as people are able to specialize and provide conclusions based on local knowledge,
- Aggregation—the mechanism to turn private judgments into a collective opinion.

A more detailed overview of importance of these factors and rationale behind them in the context of experimental setup may be found in [23].

B. Context of the three-layered model

Technical debt introduced in the codebase/design phase is tightly coupled with the code being implemented. It has direct impact on the codebase cohesion, coupling, process flow, etc. Additionally, in the Agile approach, design phase is reduced to minimum. So the overall code and design quality is the responsibility of a software development team and can be assessed together using the codebase debt in the broader context of the three-layered model. Importantly, the technical debt ratio in a product is more of a metric how the historical decisions were made, so it should not be used for comparison of the organizations' maturity. Technical debt trend shows the efforts of the current software development team.

Monitoring technical debt trend in the product can give a software development team an early problem indicator,

 ISBN: 978-1-61208-230-1

before these software problems are actually released (where technical debt ratio has significantly grown during the project). Another way to use technical debt ratio is to compare particular areas of the same code, to identify where investment is required before any new feature is implemented. In order to establish consistent technical debt management strategy the team may define a baseline ratio for codebase/design debt that reflects the acceptable debt level for the product and optimize the particular code areas to limit this debt ratio below the desired threshold. However, some questions have to be answered: *When to stop optimizing? Should the code be optimized until it is "clean" and 100% testable?* Software development team may not possess this knowledge. It is the business/architecture team that knows what the software product's roadmap and strategy is: whether it is a onetime development with no maintenance planned or mission critical software that will be maintained for years. A decision on acceptable level of codebase/design technical debt should be taken by the software development team, on the basis of a feedback loop from business/architecture to avoid sub-optimization.

C. Experiment 1: Sonar vs. Wisdom of Crowds

As it was mentioned earlier in this section, some tools exist, which provide the technical debt calculation, for example Sonar for Java-based products. However, more complex products or development environments may pose a serious challenge to find a consistent approach to evaluate the status of the product. This was the rationale behind comparing objective assessment provided by Sonar to the subjective measurement strategy based on the Wisdom of Crowds approach.

This experiment was conducted in one of the main components of a major system release. The developers prioritized modules according to the following criteria:

1. Where introducing changes is the most difficult
2. Which are poorly written (refactoring required)
3. Where there is high amount of latent defects

Survey results were compared to technical debt ratio calculated by Sonar. The results were analyzed for any correlation between objective measurements and subjective assessment of the technical debt value. The Pearson Correlation coefficient was calculated between each of the criteria and values provided by Sonar. In each case, correlation for the objectively measured technical debt (by Sonar) and the engineering assessment was significant. The first criterion is the closest to how technical debt is usually perceived—the consequences of earlier short-cuts of incomplete knowledge for current work (state of the product being maintained). Not surprisingly, the highest correlation with objective Sonar assessment was observed—the correlation coefficient reached 0.84 (Fig. 4). For criteria 2 and 3, the correlation was also significant, reaching 0.75 and 0.67, respectively.

There is one more aspect, which was considered in this assessment. An additional question was asked: Do you think it is necessary to pay-off technical debt in an area before any changes are introduced? The correlation with technical debt value reported by Sonar is significant (0.61); however we should also focus on the specific measurement points, where although debt is low, engineers insist on paying off the debt. This provides further insight into architectural or portfolio debt concerns.

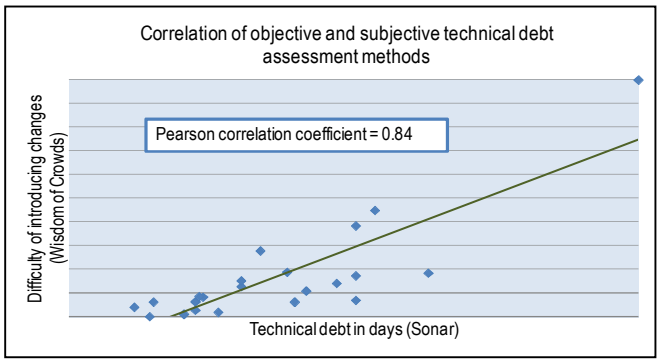

Figure 4. Correlation of objective and subjective assessment methods (difficulty of introducing changes in software modules).

These three criteria mentioned earlier by no means exhaust technical debt concept, but may provide a representative guideline for the assessment. Modules, which accumulated higher technical debt, are perceived as:

- Most difficult to have changes introduced,
- Badly written (require refactoring),
- Highly riddled by defects.

Statistically significant correlation was observed. This assessment provides us with a good overview of the quality perceived by engineering (e.g. maintainability) as well as overall quality visible to the customer (e.g. error proneness of certain system areas).

If we use all the answers as a basis for aggregation, instead of aggregating them question by question, the Pearson correlation coefficient still remains high (0.77). Therefore, we claim that having common understanding of technical debt and its consequences, the subjective measurement based on the Wisdom of Crowds method can provide reliable and consistent results in comparison with objectively defined measurements (in this case Sonar output). This fact opens a possibility of measuring the technical debt in more complex software products, even of a heterogeneous nature.

D. Experiment 2: Naturalistic observation—technical debt management

The following study was conducted to answer the question whether it is worthwhile to consider technical debt—not only as quality indicator for already existing code, but perhaps as a quality gate for software development activities. Unfortunately, delayed payment for what cannot be easily measured, or not knowing the potential consequences and value (real costs) usually causes technical debt to be neglected and accumulate over time, until it is very difficult and costly to address. Without a value associated to this, it is usually omitted during task prioritization. In this experiment, we wanted to understand how two similar modules behave if one is optimized against technical debt concerns and the second is not.

 ISBN: 978-1-61208-230-1

Typically, preparation of an experimental set-up focuses on establishing a controllable environment. Unfortunately the very activity of preparing for the experiment may impact its results. This is manifested in the social proof phenomenon described by Cialdini [5]; in our case, engineers may try to figure out what is the expected "good" result and therefore inadvertently optimize their work against it. Providing some framework before the experiment, even without any explanation about the quality of the value to be compared against, may also cause additional obstacles—such as cognitive bias, known as anchoring [25], impacting the results. In this particular case, we were in a very good position as the study was actually a *naturalistic observation* [21], i.e., no direct researcher influence was disturbing the engineering activities. The retrospective analysis of the results was performed.

We were assessing two similar database replication handler modules, which were developed for two similar database engines. The first one was taking into consideration technical debt and adhering to the rules defined by Sonar plug-in (module_1), and the second one was developed without the knowledge and proper application of that concept (module_2). Both modules were handling data synchronization and monitoring mechanisms, and the same development approach was used: pair programming during similar timeframe. Looking into technical debt as reported by Sonar, the value for module_1 was 2.5 times smaller than for module_2.

The chart presented below shows the defects found and their distribution among testing phases (Fig. 5). What can be observed is that number of defects found in module_1 was 2.6 times smaller than number of defects found in module_2. Moreover for module_1 test screening effectiveness was much better at earlier phases (80%), as only 20% escaped this activity and was found during system tests. No defects were found post release. For the second module, similar box test effort was capable of finding 8% of defects, 69% were found by system tests, and finally 23% were found post release.

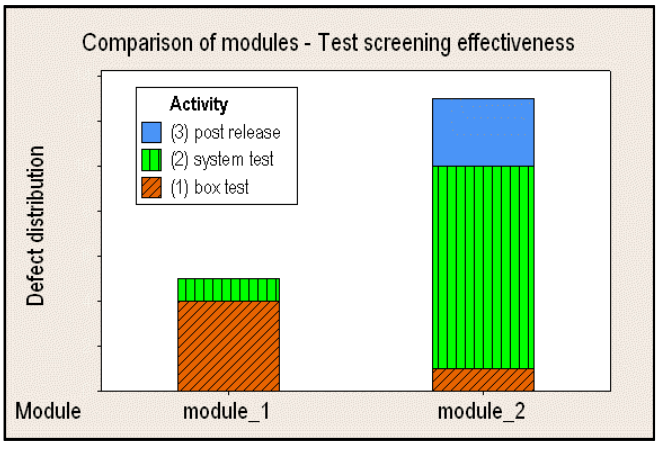

Figure 5. Comparison of defect arrival distribution between modules.

The development cost of the module written neglecting technical debt concerns was 10% smaller. But, as a result, the cost of fixing the problems in this module was almost two times higher than the initial creation effort. The results confirmed that neglecting technical debt incurs not only problems with maintenance later, but also causes higher amount of defects found both during development process and post release ones. We claim that technical debt value is a reliable prediction of future software product quality.

V. ARCHITECTURE DEBT

Numerous publications emphasize the fact that architecture is not only the macroscopic recognition of design, but constitutes the backbone of the system, orthogonal to its functionality, for example [10]. More importantly, the selection of architecture is a choice between multiple ways of implementing the system that accomplish an established set of functionalities. The choice of technology solutions—a decision made in the early phases of the project—implies a more or less explicit choice of the relevant architecture and may prevent the efficient refactoring later (inherent activity in projects using Agile approach). Therefore, reconstruction of the architecture, especially based on a specific technology, becomes much more costly. In this respect, the architecture related decisions should be the result of continuous, long-term analysis of customer needs, leading to an optimal solution selection. Moreover, changing the architecture or technology of software development needs to be a result of a complex analysis of the business process, and qualitative assessment against technical debt. So, the question arises: what metrics should be used to evaluate the architecture? One approach may be the Architecture Tradeoff Analysis Method (ATAM), which offers "utility tree" analysis [19]. Architecture technical debt assessment can be done using quality factors like: modifiability, scalability, and latency. For example, modifiability directly affects the characteristics of the cost of change, which can be treated as expression of technical debt at an architecture level. It is essential that the assessment of architecture should be done in reference to the Voice of Business. Following the conceptual model of "Portfolio Management" [15], it is necessary to assess technical debt for each artifact created in the software production process. Nevertheless, information on technical debt from the architecture level is critical because it accumulates technical feedback from the engineering teams (involved in software development and test activities), and directly takes into consideration information from the business (portfolio). Fig. 6 shows a model of decision making at the architecture level, which provides a roadmap for software product, technology, and development process based on all factors mentioned earlier. These roadmaps define engineering strategy, which is driven by value and considers organizational capability.

Technical debt in this model is presented as a trend in the cost of software changes, and its calculation should take into account the estimated cost of changes expected by the customers in time. This prediction should also include the estimated cost of change as if it were to be done in alternative architecture or technology. As a result, the architecture technical debt can be expressed as a set of

 ISBN: 978-1-61208-230-1

characteristics representing the cost of changes for the current architectural solution and alternative ones. This assessment should constitute a feedback to the process of business analysis on portfolio level, where profitability trend of the investment should be determined.

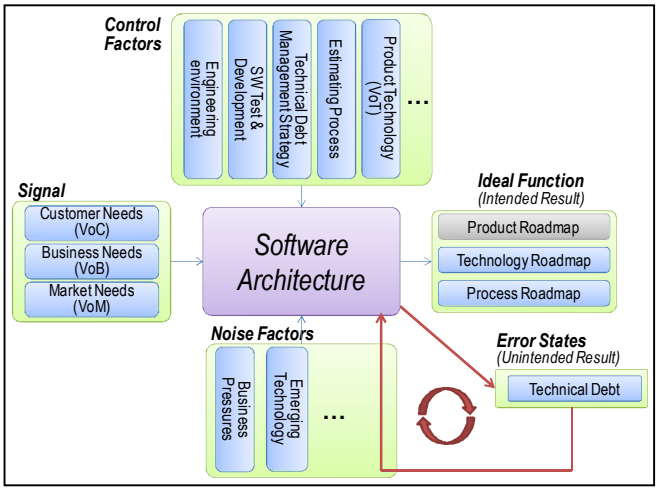

Figure 6. Technical debt limited to the context of architecture deliverables

VI. PORTFOLIO DEBT

The rationale behind portfolio debt layer (highly dependent on external factors) lies in a strategic alignment of effort spent by different organizational departments and common understanding of the goal to be pursued. Referring to the technical debt model shown in Fig. 2, a portfolio level is accompanied by *Vision* for a given product in terms of sales, market needs, technology, etc. Vision guides the strategy definition (architecture layer) and helps to align the software development effort (codebase/design layer). As a result, the profitability aspect needs to be taken into account, what can be expressed as:

$$Max\big(ROI(t)\big) \sim \frac{Sales(t) - Cost(t)}{Cost(t)} \quad (1)$$

where *Cost(t)* is defined as:

$$Cost(t) = C_{tools,IDE,COTS} + C_{migration} + C_{SW\ changes} + C_{other} \quad (2)$$

The selected technology should enable the organization to make an optimal decision, which aims at the highest ROI considering present and future customer needs and sufficient time to market. The result of optimizing the objective function (1) is a certain amount of observed technical debt. For example, lack of investment in new technology and refactoring at some point may just not yield the targeted ROI based upon VoB. Continued analysis of this trend can lead to a decision change, however still vulnerable to the risk of established customer goals volatility, emerging alternative technologies, or significant changes in market conditions.

For example, technical debt may show the estimate variation of the cost of change in time for the three possible technologies. Customers waiting for new and more complex features push the existing solution to the technological frontier. This prevents the effective implementation of new products or improvement of the existing ones.

Let us assume that a new technology emerges, which is very promising and the software team might be prone to immediate migration. However, this incurs a risk that expensive migration might not be justifiable from ROI perspective, as the benefits will not be realized in the expected timeframe. In such a case, a better strategy might be an evolutionary migration, where the current software is gradually modified (code refactoring) to compensate technical debt in relation to the new technology. The decision about final migration can be made when technical debt is reduced accordingly (Fig. 7).

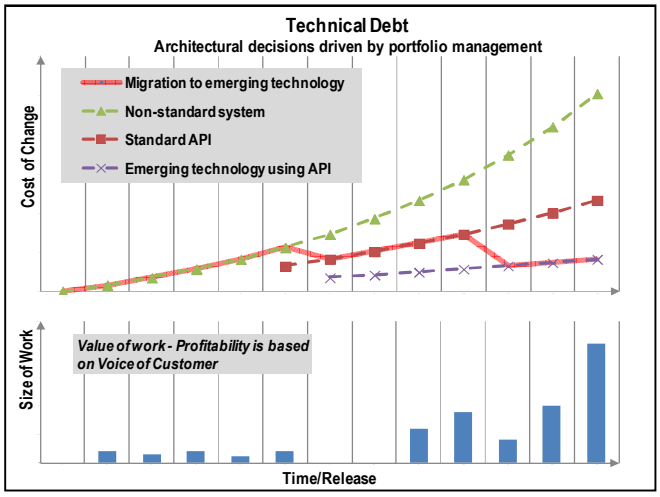

Figure 7. Technical debt driven by Voice of Customer optimized to control the refactoring of architecture

The decision about migration to a new technology is dictated by potential profitability of the estimated cost of change reflected by the Voice of Customer (VoC). The new technology may require additional investment but it will be balanced by potential financial benefits that can be calculated on the basis of VoC and VoB information. *Modern Portfolio Management* offers a statistical method to calculate this risk, making decisions about technical debt reduction economically justifiable [15].

The need for accurate assessments and quantification of future customer/market needs and its associated ROI is evident. The approach of sales prediction (proposed by Eades [9]) may be used as a reference for measuring impact of technical debt on the value perceived by customers and company profitability. Having the sales pipeline properly filled in, the potential value (yield) for a particular functionality can be estimated at any point of time (Fig. 8), as well as technical debt value associated with it. However, when assessing market opportunity, not only value (interpreted as sales prediction) should be taken into account but also how it is aligned with product long-term portfolio planning and potential market needs. In summary, the nature of the portfolio debt trend may not be linear, as the debt reveals itself as a result of certain external factors. Therefore

 ISBN: 978-1-61208-230-1

codebase/design and architecture layers have to take some degree of assumptions and risks (even if unconsciously). This further emphasizes the importance of communication links presented in the next section.

Milestone	Revenue	Win Odds	Milestone description	Yield
T	*sales-at-T* $		Territory	
S	*sales-at-S* $	10%	Qualified Suspect	10%**sales-at-S* $
D	*sales-at-D* $	25%	Qualified Sponsor	25%**sales-at-D* $
C	*sales-at-C* $	50%	Qualified Power Sponsor	50%**sales-at-C* $
B	*sales-at-B* $	75%	Decision Due	75%**sales-at-B* $
A	*sales-at-A* $	90%	Pending Sale	90%**sales-at-A* $
W	*sales-at-W* $	100%	Win	100%**sales-at-W* $

Figure 8. Solution Selling Pipeline Milestone Chart

VII. COMMUNICATION NEEDS

Assessing technical debt as an inherent aspect of the software development process reveals the critical need for seamless communication and organizational alignment to understand and manage all three defined layers of the technical debt model. To remove obstacles with information interchange the communication links as presented in Fig. 9 should be established.

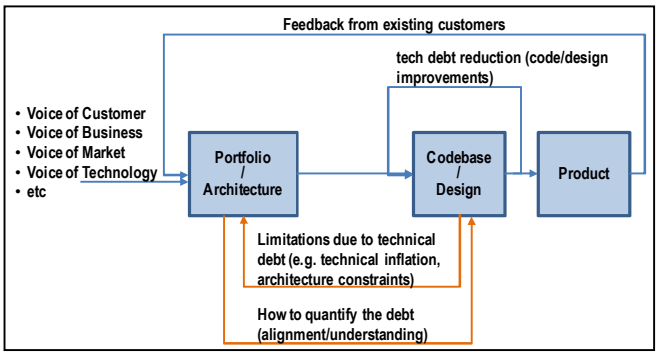

Figure 9. Addressing communication needs related to technical debt

The discussed communication needs may be addressed using a matrix matching feature development schedule with potential benefits to be gained from technical debt pay-off. Such a matrix (table) complements the practical use of the three-layered model discussed in this paper. The sample matrix implementation is presented in Fig. 10. *SW Change Request* rows represent **Technical Debt Items (TDI)** assessed against quality attributes. For each TDI pay-off cost (column *Cost*) and savings per each roadmap feature (columns *Feature A, B, etc.*) are estimated. Each feature is assigned to a planned software release (top row: releases *R1, R2, etc.*) and has a total development cost associated with (row *Cost*). As a result different scenarios may be evaluated, for example: what is the benefit of having a particular TDI paid off in releases R1 or R2, how does it change when scope of releases R1 and R2 will be released together? Such a matrix supports also discussions on the budgeting scenarios. Let us assume that a fixed budget (450k$) was planned for a given project. The total cost of the functionality planned for releases R1 and R2 reaches 500k$. An investment in paying off technical debt items (170k$) provides us with the savings of 275k$ only when features A, B, C, D are implemented together. Summarizing, having all technical debt items paid off, the gained benefits enable the company to develop all features within the fixed budget (3):

$$450k\$ > 500k\$ + 170k\$ - 275k\$ \quad (3)$$

Product roadmap - features vs. releases R1, R2,...

			R1	R1	R2	R1			
			Feature A	Feature B	Feature C	Feature D			
			100	80	120	200			
SW Change Request (Technical Debt Item)	Quality attribute	Cost [k$]	savings per feature				Savings in		
							R1	R2	R1+R2
Unit test environment	Testability	20	20	10	0	10	20	-20	20
Design granularity for subsystem *S*	Modifiability	15	0	0	30	10	-5	15	25
Component *C* response time	Performance	35	50	5	0	40	60	-35	60
New SW Integrated Development Environment *IDE* (CoTS)	Maintenance, Modifiability	100	10	0	80	10	-80	-20	0

Figure 10. Technical debt vs. portfolio assessment matrix

However, the time to market for individual features may be different, what may impact selling value. Such an assessment may lead to the decision of making separate releases, acknowledging the technical debt presence and maximizing the profits. As a result the portfolio analysis makes the roadmap change frequently, reflecting changing market conditions and embracing engineering feedback. Any external change is being reflected in ongoing synchronization of the roadmap, thus optimizing profits and ROI stemming from an engineering investment. Additionally having technical debt properly assessed and being paid off according to the long term needs, engineering organization achieves easier maintainability of the owned code base over a longer period of time.

VIII. CONCLUSIONS

The model proposed in this paper exemplifies the need for a framework, in which the technical debt has to be assessed and reveals clear rationale behind it. Moreover, various approaches to evaluation and quantification of technical debt were presented.

To answer the question *what if the cost prohibits using advanced analytical approaches, or there are no tools to support such an analysis (e.g. heterogenic solutions, auto-generated code)?*, we claim that a properly conducted subjective assessment based on the Wisdom of Crowds is capable of providing sufficient and reliable information to help in understanding the technical debt in such environments and may also support prioritization of refactoring tasks.

Technical debt prioritization may be considered in a three-layered perspective. The consequences of this model show how the defined layers (codebase/design, architecture and portfolio) depend upon and are related to each other. Neglecting any one of these dependencies may result in sub-optimization. Moreover, from a business perspective such an

 ISBN: 978-1-61208-230-1

effort may be treated as a waste (overproduction), which may impede further improvement activities initiated by engineering.

Technical debt management strategy should be the key concern, not relentless pay-off. The understanding and prioritization of the debt may be done on a value-basis, providing a bridge between the business and engineering, and a common strategy for the technical debt management. Furthermore, experiments indirectly stressed the importance of portfolio analysis, to confirm for which software components refactoring effort may be prioritized.

Even a high technical debt value may be discarded if a particular product is close to its retirement. A different approach may be taken when the product is planned to be expanded, to constitute the baseline for other products (according to predicted market needs). Another concern is maintenance time—for long-term projects, where maintenance is scheduled for years, the debt will have a different value comparing to the solutions for products, which have a significantly shorter life. Lastly, the criticality of issues which may occur is also a very important aspect, as mission critical communications should be treated differently than cell phone games.

IX. FUTURE DIRECTIONS

Current work is focused on the validation of the three-layered technical debt model and calibration of measurement approaches among several organizational departments. Additionally there are further concerns which may be addressed by the future research. Several of them are mentioned below:

Technical debt communication:

- How to address the communication needs avoiding known "traps" in the organizations' psychological and sociological composition (as team cooperation and software development are social activities).
- How much proprietary information is to be shared with broader audience? How to ensure it is properly handled?

ROI assessment:

- What factors should be taken into account assessing technical debt impact on engineering productivity? What is their relative impact and importance?
- How to assess and address the nonlinear traits of value-based technical debt trend?

Mathematical formulae:

- How the interdependence among technical debt layers can be approximated by mathematical formula—expanding the model proposed by Guo and Seaman [15]? How it may be deployed to measure technical debt in the company with rich software legacy, with no technical debt evidence consistently tracked?

X. ACKNOWLEDGMENTS

We want to express our gratitude to James Joseph Waskiel and Elżbieta Stochel who supported the work, relentlessly editing and reviewing the paper.

REFERENCES

[1] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, Eds. Value-Based Software Engineering. Springer, 2005.

[2] B. Boehm and L.G. Huang, "Value-Based Software Engineering: A Case Study," IEEE Computer, March 2003, pp. 33-41.

[3] T. Brazier, "Managing Technical Debt," Overload Journal, Vol. 77, Feb. 2007, retrieved on Feb. 13, 2012, from: http://accu.org/index.php/journals/1301

[4] Checkstyle, http://checkstyle.sourceforge.net

[5] R. Cialdini, Influence: Science and Practice, 4th ed. Pearson Education, 2000.

[6] Cobertura, http://cobertura.sourceforge.net

[7] W. Cunningham, "Ward Explains Debt Metaphor," retrieved on Feb. 11, 2012, from: http://c2.com/cgi/wiki?WardExplainsDebtMetaphor

[8] W. Cunningham, "The WyCash Portfolio Management System," OOPSLA'92 Experience Report, Mar. 26, 1992, retrieved on Feb. 11, 2012, from: http://c2.com/doc/oopsla92.html

[9] K. Eades, The New Solution Selling: The Revolutionary Sales Process That is Changing the Way People Sell, 2nd ed. McGraw-Hill, 2003.

[10] G. Fairbanks, Just Enough Software Architecture. A Risk-Driven Approach. Marshall & Brainer, 2010.

[11] M. Fowler, "Technical Debt", Feb. 26, 2009, retrieved on Feb. 13, 2012 from: http://www.martinfowler.com/bliki/TechnicalDebt.html

[12] M. Fowler, "Technical Debt Quadrant", Oct. 14, 2009, retrieved on Feb. 10, 2012 from: http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html

[13] FXCop, http://msdn.microsoft.com/en-us/library/bb429476.aspx

[14] O. Gaudin, "Evaluate your technical debt with Sonar," Jun. 11, 2009, retrieved on Feb. 5, 2012 from: http://www.sonarsource.org/evaluate-your-technical-debt-with-sonar/

[15] Y. Guo, C. Seaman, "Portfolio Approach to Technical Debt Management," Proceeding of the 2nd working on Managing technical debt (MTD '11), May 2011, pp. 31-34

[16] Klocwork, http://www.klocwork.com

[17] E. Maas and P.D. McNair, Applying Design for Six Sigma to Software and Hardware Systems, Prentice Hall, 2009.

[18] Maven, http://maven.apache.org

[19] R. L. Nord, M. R. Barbacci, P. Clements, R. Kazman, M. Klein, L. O'Brien, and J. E. Tomayko, Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM). Carnegie Mellon University, 2004.

[20] PMD, http://pmd.sourceforge.net

[21] J.J. Shaughnessy, E.B. Zechmeister, and J.S. Zechmeister, Research Methods in Psychology, 5th ed. McGraw-Hill, 2000.

[22] J. Surowiecki, Wisdom of Crowds. Why the Many are Smarter Than the Few. Abacus, 2005.

[23] M. G. Stochel, "Reliability and accuracy of the estimation process. Wideband Delphi vs. Wisdom of Crowds," Proceedings of 35th Annual IEEE International Computer Software and Applications Conference, Jul. 18-21, 2011, Munich, Germany, pp. 350-359.

[24] M. G. Stochel and R. Sztando, "Testing optimization for mission-critical, complex, distributed systems," Proceedings of 32nd Annual IEEE International Computer Software and Applications Conference, Jul. 28-Aug. 1, 2008, Turku, Finland, pp. 847-852.

[25] A. Tversky and D. Kahneman, "Judgement under Uncertainity: Heuristics and Biases," Science, New Series, Vol. 185, No. 4157, Sep. 27, 1974, pp. 1124-1131.

 ISBN: 978-1-61208-230-1