PGPEX

BUSINESS REVIEW

October 2014 | First Edition

THE ECONOMICS OF **TECHNICAL DEBT**

What is technical debt? Consider that in a business organization the IT department is rushing to meet the deadline for development of an important software application. The software programmers in the IT department still have major components to develop as the deadline approaches. In such a situation, either of two things can occur:

A) The software programmer makes a conscious decision to compromise with the architecture of the application so as to deliver it by the due date. He develops a long, convoluted logic code which does not follow the design standards laid out for the project. As a result both the code quality and application architecture are compromised.

B) The software programmer was unaware of the design standards or conventions for the project and he unintentionally developed code that degraded the quality of the software code.

> The degradation of the design, quality of code and architecture of the software is known as the **technical debt**

In both cases, although the development of the software application may have met the deadline and it functions correctly, the software code quality and its design standards have been compromised. This degradation of the design, quality of code and architecture of the software is known as the technical debt of that software application. This has important consequences for the future. Another programmer working on the same software application to develop a new business application or to make modifications to the existing one will have to spend more time to understand those parts of the logic of the software application whose quality had been compromised.

The additional effort, resources, time and money spent later to locate and amend the code and design to meet the expected quality standards or to remove any architectural shortcuts made earlier due to shortage of time gives the **measure of the size of technical debt.**

As per a study conducted by CAST, there is an average technical debt of $3.61 per Line of Code (LOC)[3]. According to Gartner, the current global IT debt stands at $500 Billion, which is, expected to double to $1 Trillion dollars by 2015[7].

Technical Debt is often compared to financial debt; it has characteristics of principal and interest like financial debt, except that technical debt does not have any precise unit of measurement unit [9]. However, today with improved IT processes and tools, technical debt can be measured either in terms of Kilo Lines of Codes (KLOC) of the software application or in terms of Total Quality Index (TQI): a composite score of robustness, performance, scalability, transferability and changeability [3] that can be associated with monetary value[13].

Just as in financial debt, technical debt gets compounded over time. If we do not make good the debt by putting in effort to improve the software design and architectural standards, the debt will increase on a compounded basis with each successive software application release cycle, further worsening the quality of the software application [9].

Business management often argues that the purpose of information technology is to support its core business functions. Any investment in improving the quality of the code does not justify opportunity cost in terms of additional time and resources expended. While this may be true in some cases, especially if the developed software has no further application, mostly as technical debt keeps compounding, a stage may arise when the later versions of the application fail to be changeable (ability to make changes quickly) and transferable (ability of other team members or other teams to understand the existing logic) [9], which will lead to a substantial IT budget expenditure on maintenance rather than new development of the software application.

ECONOMICS BEHIND TECHNICAL DEBT

Measuring technical debt helps us to understand when it is best to engage in mitigation of the technical debt and maximize the total economic value from the software application. New IT processes and tools make it possible for us to measure the quantum of technical debt.

A software application can be considered as a production function of a number of function points (unit of measurement to measure the amount of business functionality present in a software application) of an application and the length of the code (measured in terms of KLOC) written to develop the application.The output of this function is the quantitative measurement of quality of the software application represented as total product index.

Total product index = fx (Software code (KLOC), Number of function points) *- Equation 1*

Table below shows the number of function points and the number of lines of code written to develop those function points. The total product index is the production function of the two input variables with the variable function points kept constant.

MEASURING **TECHNICAL DEBT**

Function Points	KLOC	Total Product Index	Average Product Index	Marginal Product
30	0	0	0.00	-
30	15	450	30.00	30.00
30	16	500	31.25	50.00
30	17	545	32.06	45.00
30	18	575	31.94	30.00
30	19	600	31.58	25.00
30	20	620	31.00	20.00
30	21	630	30.00	10.00
30	22	635	28.86	5.00
30	24	635	26.46	0.00
30	25	620	24.80	-15.00
30	26	600	23.08	-20.00
30	27	575	21.30	-25.00

Table 1

TOTAL PRODUCT INDEX

It represents the application's **structural quality aspects** such as: software design standards, robustness of architecture, application scalability, security implementation etc. and the application's functional intricacies such as complexity of functionality, maturity and future usage of the application. **The index gives us a measurement of the quality of the software application.** Both these qualities have some degree of correlation with the number of function points and software code. The structural quality of the application decreases as KLOC of software code written for a given number of function points increases. While, the functional intricacy of the application generally increases with the increase in the number of function points.

The impact of technical debt on a software application can be understood through the economic analysis of the production function *(Equation-1)*. Here, we consider

that the number of function points in the production function is constant for a given software application to be delivered. If no measures are taken to reduce the technical debt, then the software application code in terms of KLOC that has to be written to deliver the next software release will increase with each successive release due to increasing complexity and technical challenges. The *law of diminishing marginal returns* will govern the output of every successive software application.

MARGINAL PRODUCT

It is the marginal increase in the total product index compared to previous release for every 1 unit increase in KLOC written to implement it. As, with each successive release, the number of lines of code required to implement function points increase, the total product index of the application gradually decreases as a result, and the marginal product declines in stage B and stage C, after an initial increase in stage A.

$$\textit{Marginal Product} = \frac{\Delta\ \textit{Total Product Index}}{\Delta\ \textit{Software Application Code}}$$

AVERAGE PRODUCT

It is the total product index of software application compared to the lines of code, KLOC written to implement the same.

$$\textit{Average Product} = \frac{\textit{Total Product Index}}{\textit{Software Application Code}}$$

Figure-1 depicts the law of diminishing marginal returns for a software application that has gradually acquired technical debt with successive release cycles.

The total IT budget for a software release cycle can be represented as: -

$$\textit{Total IT Cost Budget} = \frac{\textit{Fixed Cost (Maintenance Cost)}}{\textit{Variable Cost (Function Points)}}$$

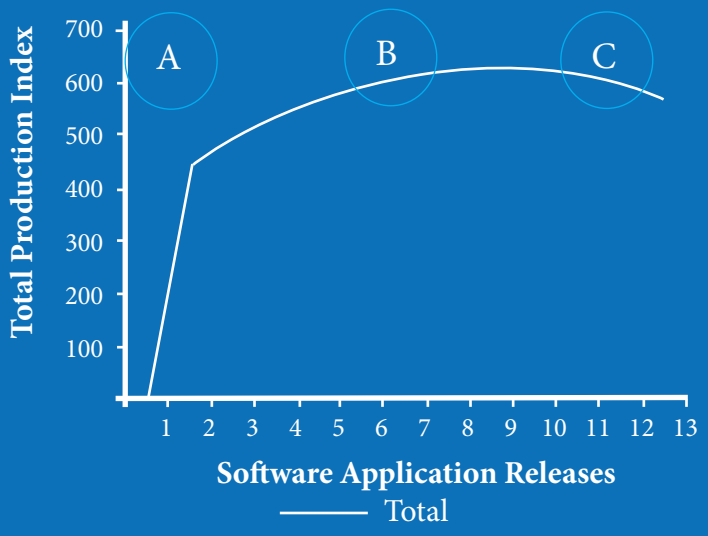


Figure 1

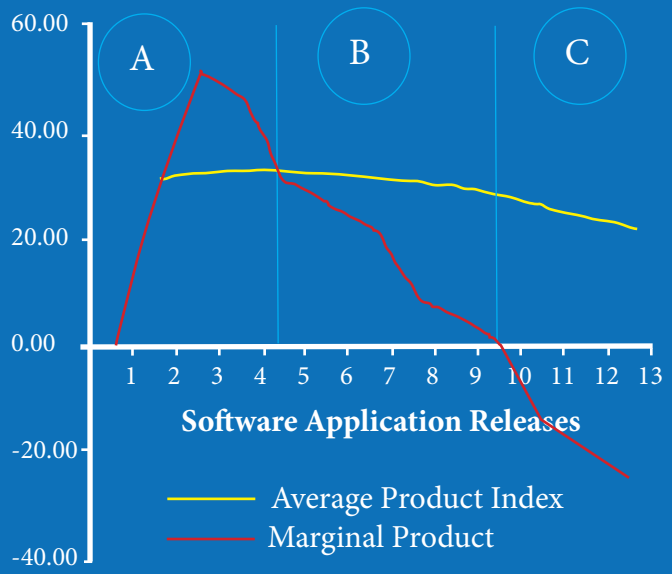


Figure 2

If each function point is assumed to be of the same complexity, the cost of development of each function point will be same, as a result the total variable cost for development of the software release will be constant. However, the fixed maintenance cost will increase with each successive release cycle, due to the increase of issues attributed to the poor design or architectural flaws.

Figure-2 is divided into three stages, each representing a different phase of software application.

STAGE **A**

When an IT investment is made by a business organization; the goal is to implement core business functionalities that would support business processes. During the initial phase, IT budgets are allocated for development of the software application. The application may have a relatively clean design without many architectural decisions to be considered.

New functional features are continuously added to the application without much emphasis on degradation of the design standards or the architectural constraints. Software application is still in scalability mode. As shown in *Figure-1*, the total product index of the application that comprises of the structural quality and the functional intricacy increases. Although the structural quality of the application decreases with each subsequent application release during this stage, overall product index increases because of increase in the functional quality. As shown in *Figure-2*, the marginal product of the software application is greater than the average product index of the application in stage A. Therefore it is more beneficial for the business organization to implement business requirements without allocating significant budget towards initiatives in resolving technical debt.

STAGE **B**

A software application is considered to be in this stage once it has functionally matured. An application in a matured state is characterized by intricate workflows and complex business implementations. New functionality implementation decreases considerably and majority of the application development is for modifications or refinement of the features in the application. During this stage, IT expenditure towards the maintenance of the application grows significantly. If the firm's management does not act towards resolving the technical debt, it will have to incur much higher costs later owing to compounding of the debt. The incremental change in the total product index increases but at slower rate than in the previous stage.

In this stage, as shown in *Figure-2*, the marginal increase in the product index is less than the average product index of the software application. As a result the total product index increases at slower rate than in the previous stage, as shown in figure-1. Unlike stage A, where the management can stay focused on scalability of the application, the management should begin to think of the future of the software application in stage B (please refer to 7 Key Decisions: Measure, Monitor and Mitigate Technical Debt in this article). If the management fails to recognize the importance of strategic decisions towards mitigating the technical debt in this stage, the technical debt can lead to further deterioration of the software application and will have to be separately dealt with in Stage C.

STAGE **C**

This is the terminal stage for the development of the software application. An application is categorized in this stage when it has accumulated a large amount of technical debt. As shown in *Figure-1* the total product index of the application declines because of the marginal decrease in the total product index. Due to a large amount of technical debt, the structural quality of the application declines to such an extent that it outweighs the functional quality of the application. In this stage the average product index of the application decreases due to impact of negative marginal product index. This implies any further changes to the application will cause more issues. A significant IT budget goes towards maintenance of the application rather than development of any new functional features. The effort required for clearing the technical debt of an application is estimated to be greater than the total value that can be achieved from it. Under such circumstances, the firm's management may have to decide to scrap the old application and invest in building a new application instead of fixing the problems with the existing application. This may be because the application has reached such a critical point that any further enhancement or addition of new features to the application requires changes to the existing design implementation or architecture that is no longer cost effective. This situation may also arise because

of obsolescence of the technology platform, or of open source alternatives available in the market, or unavailability of technical resources. Here, a decision to scrap the existing application and build a new application is often considered more feasible. The decision also depends upon the criticality and complexity of the application. For complex business applications that are critical to the core business processes of the organization, it may be more viable to invest in reducing the technical debt of the existing application. It is thus increasingly important for the firm to understand the significance of the technical debt, and how timely action towards planning of mitigation of technical debt can prevent a software application from reaching stage C. The goal here would be to increase the useful life of the software application.

SEVEN KEY DECISIONS MEASURE, MONITOR AND MITIGATE TECHNICAL DEBT

Initiatives such as technical debt mitigation not only involve the IT department, but also require significant attention at the C-suite level; the CIO of the firm should take this responsibility. The CIO should invest time on 7 key decisions before creating a strategy for technical debt mitigation.

1. IDENTIFY THE STAGE
2. ACCESS THE USEFUL LIFE
3. IDENTIFY THE CRITICAL BUSINESS COMPONENTS
4. ESTIMATE TECHNICAL DEBT
5. BUY-IN FROM CEO
6. RE-ESTIMATE DEVELOPMENT EFFORT
7. LONG TERM STRATEGY

REFERENCES

1. ***David K. Williams*** *- http://www.forbes.com/sites/davidkwilliams/2013/01/25/the-hidden- debt-that-could-be-draining-your-company/: accessed June 16, 2014.*

2. ***Joe McKendrick*** *- http://www.zdnet.com/will-software-publishers-ever- shake-off-their-technical-debt-7000010366/: accessed June 16, 2014.*

3. ***CAST*** *- http://www.castsoftware.com/resources/resource/brochures/thank-you/full-crash-report: accessed June 16, 2014.*

4. ***The Software Engineering Institute*** *- https://www.sei.cmu.edu/community/td2011/upload/foser076-brown.pdf: accessed June 16, 2014.*

5. ***James Shore*** *- http://www.jamesshore.com/Blog/An-Approximate-Measure-of-Technical-Debt.html: accessed June 16, 2014.*

6. ***Ward Cunningham*** *- http:// c2.com/doc/oopsla92.html: accessed June 16, 2014.*

7. ***Deloitte*** *-http://www.deloitte.com/assets/Dcom-Luxembourg/Local%20Assets/Documents/Whitepapers/2014/dtt_en_wp_techtrends_10022014.pdf: accessed June 16, 2014.*

8. ***Wikipedia*** *- http://en.wikipedia.org/wiki/Technical_debt: accessed June 16, 2014.*

9. ***Reinertsen& Associates*** *-http://reinertsenassociates.com/technical-debt-adding-math-metaphor/: accessed June 16, 2014.*

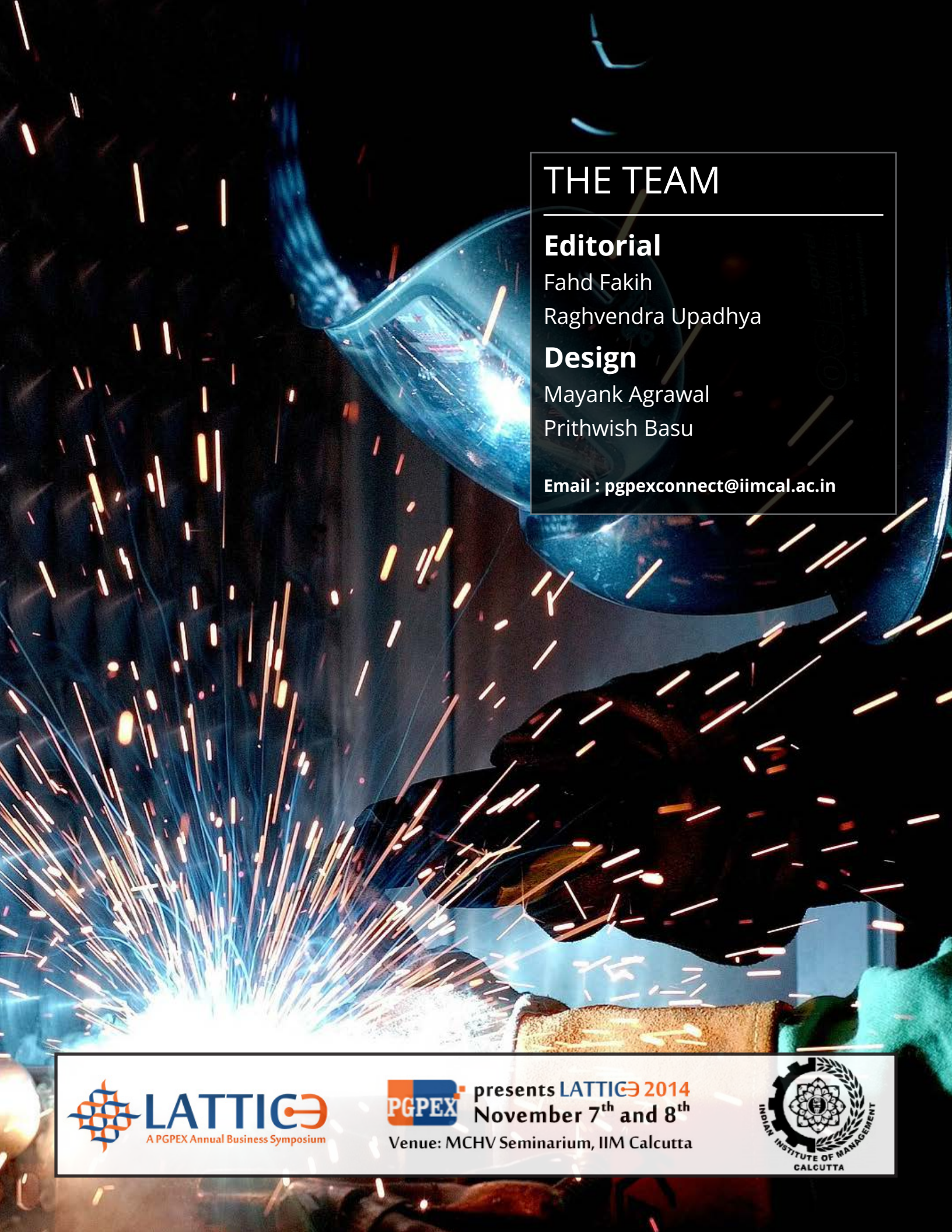
10. ***Shipra Malhotra*** *- http://www.dynamiccio.com/2014/04/technical-debt-why-cios-should-be-bothered.php: accessed June 16, 2014.*

11. ***Robert S. Pindyck , Daniel L. Rubinfled , Prem L. Mehta*** *- Microeconomics Book, Chapters 5,6 and 7.*

12. ***Steve McConnell*** *- http://www.construx.com/10x_Software_Development/Technical_Debt/: accessed June 16, 2014.*

13. ***Jonathan Bloom*** *-http://blog.castsoftware.com/gartner-cast-whitepaper-how-to-monetize-application-technical-debt/: accessed June 16, 2014.*

Written by:
Abhishek Gupta
PGPEX 2015



THE TEAM

Editorial

Fahd Fakih

Raghvendra Upadhya

Design

Mayank Agrawal

Prithwish Basu

Email : pgpexconnect@iimcal.ac.in

LATTICE

A PGPEX Annual Business Symposium

PGPEX presents LATTICE 2014

November 7th and 8th

Venue: MCHV Seminarium, IIM Calcutta

INDIAN INSTITUTE OF MANAGEMENT CALCUTTA