Technical Debt: Going beyond code

APPLICATION TRANSFORMATION MANAGEMENT

Table of Contents

Author information

Arnab is an Enterprise Architect having 17 years of experience in leading large accounts, handling all disciplines ranging from business, technology to security & information architecture. He is an expert on TOGAF & Zachman frameworks especially in areas of process definition, reference & application architecture, integration & services architecture and operational modeling.

He is currently working in Hexaware's EA Consulting practice supporting multiple pre-sales activities as a Solution Architect, doing technical due-diligence & risk assessment along with pure-play consulting engagements.

A quick take on Technical Debt

Technical debt (TD) refers to a product's deficiencies caused due to resource constraints, time-to-market or environmental factors like technology, process or business immaturity. The product deficiencies, other than lack of functional completeness, can range from operational characteristics like maintainability, configurability issues or extensibility challenges to overall stability of the platform engaged.

This can result in incomplete, immature or inadequate delivery artifact i.e. there is nothing as "doing perfect change" in product life cycle. It's important for any enterprise to completely understand the root causes of technical debt, parameters governing it and ways of avoiding it. The biggest problem of technical debt is not the principal (original gap, hack or shortcut) but how long the accumulated interest (continuous development on flawed codebase) is carried forward without being addressed. A non-payment of technical debt in the long run forces software "bankruptcy", when no further value is derived and even trivial changes can result in significant cost overheads.

> "A recent study suggests that an average of $3.61 of technical debt exists per line of code, or an average of more than $1 million per system. Gartner says that 'current global IT debt is estimated to stand at $500 billion, with the potential to rise to $1 trillion in next couple of years'. Best-case testing is capable of unearthing only 35% defects."

Now, is technical debt issue limited to just code quality and focused only on codebase? No, we feel it is not, and would like to further add that in the true sense what started off as code quality metrics has now evolved into a product engineering issue. This is because we tend to be value-neutral when it comes to software development.

So our stance is that technical debt is a factor of business value return of product engineering investments and the value needs to be understood at every phase of development. This whitepaper offers a different view-point to conventional understanding & visibility around technical debt, by introducing model-driven approach of analyzing various parameters governing technical debt and multi-staged remediation framework to overcome this debt.

A deep-dive analysis of Technical Debt – Value Chain Model

Technical debt is never constant. It changes over time across different phases of product and software development cycle, probably starting right from business problem definition. We feel that TD starts from business problem definition/ analysis phase because of certain business constraints (like time to market, cost or complexity) or impact of fitment on existing product. Also, TD can emerge over a period of time as decay of a product due to continuous business change which further results in diminishing product value. So it is important to align the product to "value" as perceived by business user and which can be techno-functional in nature.

TD can be categorized into the 4 broad based areas as indicated below. These can be used as a foundation to baseline, measure, monitor and track TD to closure.

- Business value in terms of delivered process maturity & value, operational characteristics and enhanced usability
- Strategic debt (like time to market) may force only partial delivery of ""expected" value.
- Mechanism to measure "delivered" value

- Gap in structural integrity between architecture models with underlying Meta-model
- Tactical debt (because of NFR needs) in model
- Loss of details from model-to-code transformation
- Architecture deficiencies due to complex problem

- From lock of tests, in adequate coverage, improper tests in codebase
- Business Value Assurance is still in infancy
- Test debt is currently managed as function of lines-of-code execution flow (as code coverage metrics)

- Lack of "right" programming practices, patterns usage or wrong technology choice
- Inadvertent debt because of lack of awareness or skills around development
- Managed to a large extent through static code analysis for violations

Fig 1: Categorization of Technical Debt

From a strategic standpoint, technical debt should be gauged, monitored and tracked at overall portfolio level. Depending on business criticality & maturity of portfolio, the parameters & threshold value governing technical debt may vary. Technical debt is inconsequential when building a product for the first time but the 'principal' that gets created (during initial development), if not addressed, becomes a challenge to tackle while executing any subsequent "change" to the product.

So what is a "change" and how do we determine the business value of a "change"?

A "change" is an orchestrated set of functional, technical, operational & non-functional capabilities that together provide to a business the desired "value" at a defined cost. So even if a product meets all its functional objectives, non-compliance of operational characteristics like scalability, performance or ease-of-maintenance may still result in suboptimal value delivery. Also, the "value" can be a 'function of time' as some value may not have an immediate realization. We can categorize values into 3 major forms as shown below:

"Value" Categorization	
Functional Value	• Enhanced process efficiency, maturity, business needs at right cost • Improved product usability, usage & acceptance/ ease of use • Deliver accurate information & content to intended audience
Operational Value	• Compliance to business KPI as needed • Compliance to SLAs like availability, performance characteristics, scalability, stability, resource utilization, demands etc. • Cost of maintenance & operations like ticket density & frequency, release cycles, support service needs (back-ups, administration, configuration effort etc.)
Organizational Value	• Improved people maturity with better alignment of product goals with job role, structure & KRA. • More matured workplace through clear communication, orderly business environment through STP, precise handovers etc.

Most organizations have certain constraints towards "value" delivery, many of which act as triggers towards building of technical debt when a new product is being conceived. The triggers act universally across an entire application portfolio and can vary based on underlying business complexity, platform and development team maturity.

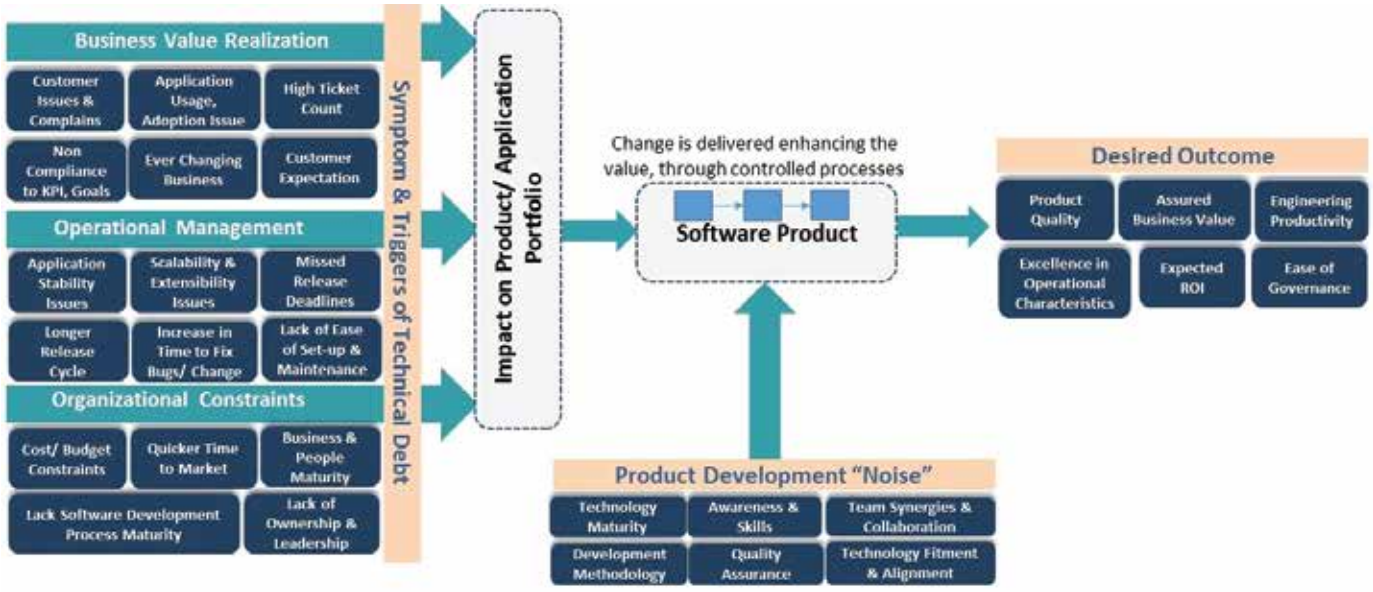

Fig 2: Portfolio Technical Debt Forces & Factors

Every organization should consider having a framework to compute technical debt across application portfolio and a matrix to determine threshold for doing refactoring (or modernization). The framework to score technical debt can have a mix of the above business & operational attributes along with technology specific codebase parameters (like Cyclomatic complexity, Halstead complexity, Test Coverage, Coupling & Dependencies, Duplicate & Unused Code, Documentation etc.).

This framework typically gets customized and fine-tuned for an organization during the assessment phase and is used to baseline TD at the organization level. This then helps in coming up with inferences and then a strategic road map to tackle TD on an ongoing basis.

Other than complete ground-up development, it is also necessary to do refactoring of existing product needs to identify & analyze changes and bring out "value". The value needs to be modeled into the solution delivery framework (as discrete set of changes) in order to trace, test, measure and improve product capabilities. So even a technical debt revision (say a reduction of score from 8 to 5) needs to be decomposed into "changes" needed and overall incremental "value" to the product.

Once the "changes" needed to bring the desired "value" are determined, it needs a framework to enable development team to effectively deliver the value (across the software life cycle).

Hexaware's Debt Management Framework – A Modeling Approach

The key to successful & continuous technical debt reduction within any organization lies in a defined strategy on TD management. TD strategy needs to encompass right from assessment & monitoring of incurred debt, standardization around process, tools, technology & guidelines to KPI & value measurement framework and knowledge management. Given the breadth of TD, it is easier to use a modelling platform as a starting point for documenting change, understanding its impact from TD perspective, incorporating simulation to see value of the change and ensuring that correct design is in place. The model simulation helps to understand the post-implementation failure points and TD parameter value variance (more than threshold) before it is actually implemented. Once the model is validated & approved, the change can be taken to implementation without the risk of failure to realize the desired value from the change.

Before going to the model, let us understand the debt management strategies that we need to implement within an organization such that TD removal & management is more of a continuous defined process instead of ad hoc reactive situations.

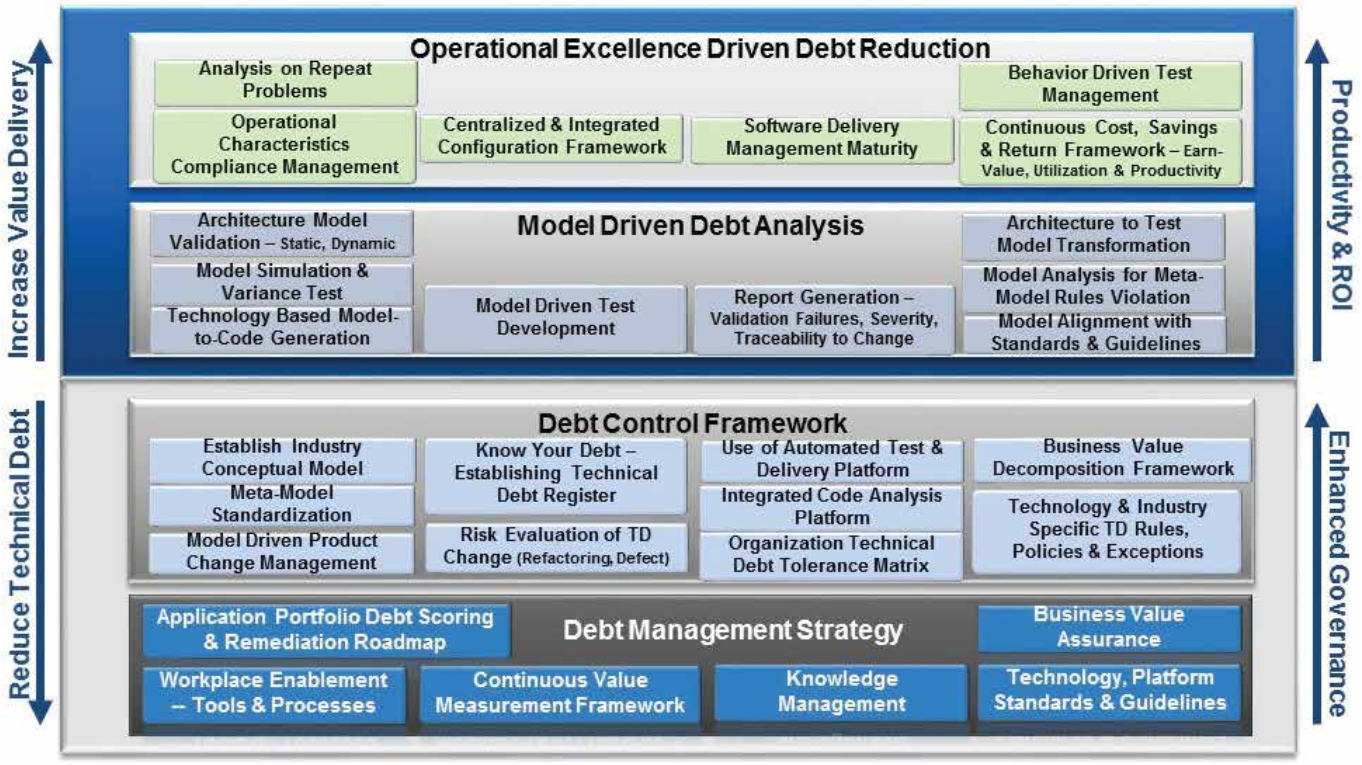

Fig 3: Debt Management Framework

Debt Management Strategy	
Application Portfolio Monitoring for Debt	• Architectural Debt analysis – Understand the code from usage of right architectural style, patterns and best practices: o UML Model analysis based on meta-model & rules o Understanding key architecture decisions, principles & guidelines and constraints governing the codebase and its impact on TD o Analyzing non-functional requirements, infrastructure and deployment model • Business Value Debt analysis – Assessment of "functional" boundary of applications for rules, processes, policies to gauge their maturity. • Implementation Debt analysis – For technology best practices, standards & guidelines compliance within codebase. • Preparing a scoring model based on portfolio data (from above) with threshold values determining the roadmap (when to do refactoring, modernization or keep-as-is).
Workplace Enablement	• Matured Software Delivery Process – Standardized delivery processes based on industry best-practices along with method, tools and people alignment. This includes organization structure to enable application and portfolio owner responsible for a debt management portfolio. • Business stakeholder awareness on technical debt and its need for continuous monitoring & correction.
Continuous Value Measurement	• Measurement of any change with respect to KPI, usage, operational characteristics etc. to understand the value through defined framework.
Knowledge Management	• Mechanism to harness data & information from multiple applications (across portfolios), both operational & transient, to enable better intelligence and drive optimization (of processes, tools, technology & governance). • Right knowledge management enables creation of tailored approach, guideline & roadmap towards removal of various forms of technical debt.
Technology Standards & Guidelines	• Standardization of technology, platform, framework, design patterns & reference architecture to reduce technical debt through consistent usage & practice and better awareness amongst teams. • Common forms of TD violation are managed by defining "Solution Templates" and enforcing it on development team.
Business Value Assurance	• Method & framework to measure (for any TD-driven change to product) delivered business value across entire development process through quality control, testing process & tools, automated defect management & analysis and knowledge harvesting.

As a part of strategy & control frameworks, we prefer using a lot of architecture model driven "change" validation, simulation and generation to ensure TD is addressed correctly. The idea is to standardize the meta-model based on industry, technology and business area to detect early TD non-compliance and fix the same with ease. We have to remember that if TD remains during architecture & design phase, it will very much likely propagate to codebase because of model-to-code generation. As believers of model driven delivery, we advocate using architecture model to design & generate test cases to allow complete coverage and assurance on critical functional value to be delivered.

So, as first point of "change" realization in software life-cycle, the architecture & design is critical and needs validation on what debt it's inheriting or getting introduced to. Modeling is a best practice towards understanding anomalies, failure points and alternatives. Depending on an organization's maturity of using model to drive any change, areas like risk management, TD threshold matrix & reporting can be integrated within modeling platform.

Operational Strategies to Reduce Technical Debt

Once we have the debt management strategy & framework in place and baked-in the practice to use models as a basis of product engineering, we still need to build and operationalize the platform. We have to figure out how to decompose & document the business needs, incrementally design, code & test it, and integrate & deploy all pieces needed to deliver the business value. In this section we talk about the realization mechanisms towards successful debt management focusing on areas like project execution methodology, technology & engineering best practices, and innovations.

From a development methodology perspective, we strongly recommend "Agile" with integrated "DevOps" as a platform to do continuous build & release for any product changes. By doing this, test-to-deployment gets completely automated, thereby reducing release cycles. This helps in early discovery of any non-value behavior and subsequent course correction. Organizations embracing methodology and practices of DevOps & Agile find it easy to tackle their TD better. Following are some of the operational strategies you can consider for reducing TD:

01 Continuous Application Innovation & Re-engineering

With rapid changes to technology, onset of Cloud infrastructure & computing platforms, container-based services etc., many of the traditional codes & frameworks needed to handle scalability, channel & device management, clustering etc. are now included in Cloud services also. This reduces the responsibility of application/ codebase in the sense that it can focus on only functional needs instead of worrying about infrastructure, deployment, scalability and cross-cutting concerns.

02 Business/ Stakeholders' Awareness on Impact of Technical Debt

Align business sponsors/ product owners to the impact of technical debt on overall health, stability and maturity of the product. Create a cost model based on score of technical debt and make product owners aware of the impact of short-term gains on increase in cost of long-term maintenance & operations.

03 Improved Visibility of Technical Debt within Application Portfolio

Map all the technical debt parameters in the application using a dashboard/ tool (like Jira, Confluence) with clear action items/ to-do list to solve the problem. Enhanced visibility & tracking of these technical defects with documented plans (effort & timeline) for refactoring & removal (within product delivery cycles) allows a managed technical debt reduction. Following points can give you an elaborative idea on this: From here

a. Prioritization of what to address first – A clear understanding of what areas/ parameters of technical debt need to be tackled and included within development plan/ agile sprint with clear success conditions.
b. Impact of change – Some areas of technical debt may involve architecture & design or underlying platform change with wider impact on budget, cost & operational aspects. This needs to be managed separately. Moving from one platform or product stack to another needs to be planned, budgeted and delivered as almost new product development.
c. Cost Savings/ Repayment per Issue Addressed – Have a clear viewpoint on total effort required and cost saved for technical issues addressed/resolved as a part of technical debt resolution.

04 Delivery Process with integrated Technical Debt Management

Have monitoring & control of technical debt ingrained within software delivery process with architecture & design driven optimization, Devops-based build process with code quality checks & automated tests and intelligent infrastructure (with configuration, distributed and autonomous).

Software Architecture and Design Best Practices for TD Reduction

Even with agile-based development, we advocate a foundation sprint specifically to lay down the architecture (solution, application, integration, data and deployment architecture) followed by a high-level design to define the cross-cutting frameworks, services etc. that can then be the basis/ core of all the functional sprints. This makes sure that you have structure, framework, guidelines & policies that govern and manage the overall codebase to comply with technical debt measures. Now in order to reduce technical risk, the architecture needs to follow certain best practices like:

- Create comprehensive domain model (or message model) and align it to line-of-business for avoiding redundancies.
- Identify the business services (or interfaces) and processes with patterns & rules to handle variations.
- Define framework and non-intrusive mechanisms for all cross-cutting concerns like use of common controller or agent to do authentication, aspect-oriented-programs (AOP) for logging & transactions etc.
- Define the application configurability parameters and have a single-point approach towards managing all configurations (preferably using platform services instead of custom code).
- Lastly, reduce coupling with underlying platform (OS, VM, COTS) or implementation stack especially when using niche cutting-edge technologies. This will also include decoupling for transport mechanism, native message formats or proprietary protocols.

Implementation Methodology

In a model-driven approach, a skeleton structure with associated framework components gets generated from the design model. Other than tool-based code generation, Maven plugins can also be used to assist in generating framework/ platform specific codebase. Before doing any development across sprints, it is recommended to update the design model and generate delta code to ensure codebase doesn't violate the design model (by large extent).

Although we recommend model-driven development, we envisage that developers are likely to write quick & erroneous codes, especially during defect fixes, minor enhancements or tight project timelines. Hence, we recommend performing checks during build & unit testing process to highlight the technical debt.

From writing tests per unit that the developers have coded such as embracing Test-first Development to including mock frameworks that can be integrated with codebase, we can leverage the best coding practices to reduce/ manage technical debt. Also, by having a robust DevOps Platform for executing units tests daily, which includes usage of tools for Technical debt evaluation, we can look at efficient coding practices to be followed.

Innovation & Cloud Strategy

Innovation in technology results in continuous reduction in lines-of-code. This is because of the availability of platform services, frameworks & assets that take care of many technical needs & cross-cutting concerns that are normally coded & maintained in any application codebase. Migration of an application to Cloud forces codebase refactoring for areas like codebase configuration & resource management, single point of failures & servicing incoming requests, decomposing monolithic/ large deployable unit and lastly and standardization of all codebase (with respect to Cloud infrastructure & services). Below are a few cases where Cloud can enforce reduction of technical debt:

- Code & Technology Standardization – Private, Hybrid and especially Public Cloud (like AWS) enforce certain ways in which an application needs to be coded & deployed to Cloud environment (like EC2, S3 storage etc.) which includes necessary container services.
- Minimized cross-cutting concerns in codebase – Any application on Cloud doesn't have to bother on scaling, load balancing, failover etc. as it's implicitly handled as a part of Cloud. For example, a mobile or web application doesn't have to deal with ways of performing request throttling, queuing of extra load or caching static data as these concerns are now handled by Cloud. Some Cloud services also provide security features like OAuth services for applications so that they do not need to build their own.
- Simplified Operations Management – As monitoring services are now offered by Cloud providers along with rapid infrastructure provisioning & deployment of application, operations team can now focus only on the core application (and they do not have to worry about underlying infrastructure set-up, performance degradation, upgrades etc.).
- Container Services – Container services like Docker/ Kubernetes or Mesos allow abstraction of underlying infrastructure layer thereby allowing applications to use dynamic provisioning of computing container instead of whole VMs. This increases the efficiency & performance of application code without much overheads along with necessary load balancing & fault tolerance from container.

Going beyond code for success

To summarize, the problem of technical debt can be eliminated the moment development teams look at it as much more than just coding issues. The entire process around software delivery, right from how we capture & understand business problem, decompose & build it through multiple phases & iterations to finally operationalize it, needs to focus on value creations. If the desired value is not delivered to business, a mere code quality tool analysis & fix will not help it and needs a broader introspection across the entire software lifecycle. The key thumb rules that should be followed within any organization are:

1. Have an organization debt management strategy in place with guidelines, matrix & measurement and finally timely remediation roadmap
2. Define a method, approach & governance towards managing the technical debt at application portfolio.
3. Focus on delivering value and measure it continuously as we build the product. Value is at times more about non-functional & operational characteristics and not just focused on business code.
4. Finally, checking technical debt only at code level is too late in the game. So one should start analyzing & validating the same right from architecture & design phase through models.

References

1. Gartner's report on technical debt – http://www.gartner.com/newsroom/id/1439513
2. Managing Technical Debt – http://2013.icse-conferences.org/documents/publicity/MTD-WS-McConnell-slides.pdf

About Hexaware

Hexaware is the fastest growing next-generation provider of IT, BPO and consulting services. Our focus lies on taking a leadership position in helping our clients attain customer intimacy as their competitive advantage. Our digital offerings have helped our clients achieve operational excellence and customer delight by 'Powering Man Machine Collaboration.' We are now on a journey of metamorphosing the experiences of our customer's customers by leveraging our industry-leading delivery and execution model, built around the strategy— 'Automate Everything, Cloudify Everything, Transform Customer Experiences.'

We serve customers in Banking, Financial Services, Capital Markets, Healthcare, Insurance, Manufacturing, Retail, Education, Telecom, Professional Services (Tax, Audit, Accounting and Legal), Travel, Transportation and Logistics. We deliver highly evolved services in Rapid Application prototyping, development and deployment; Build, Migrate and Run cloud solutions; Automation-based Application support; Enterprise Solutions for digitizing the back-office; Customer Experience Transformation; Business Intelligence & Analytics; Digital Assurance (Testing); Infrastructure Management Services; and Business Process Services.

Hexaware services customers in over two dozen languages, from every major time zone and every major regulatory zone. Our goal is to be the first IT services company in the world to have a 50% digital workforce.

NA Headquarters
Metro 101, Suite 600,101 Wood Avenue South, Iselin,
New Jersey - 08830
Tel: +001-609-409-6950
Fax: +001-609-409-6910

India Headquarters
152, Sector – 3
Millennium Business Park
'A' Block, TTC Industrial Area
Mahape, Navi Mumbai – 400 710
Tel: +91-22-67919595
Fax: +91-22-67919500

EU Headquarters
Level 19, 40 Bank Street,
Canary Wharf,
London - E14 5NR
Tel: +44-020-77154100
Fax: +44-020-77154101

APAC Headquarters
180 Cecil Street,
#11-02, Bangkok Bank Building,
Singapore - 069546
Tel: +65-63253020
Fax: +65-6222728