Universidad de Buenos Aires
Facultad de Ciencias Económicas
Escuela de Estudios de Posgrado

MAESTRÍA Y ESPECIALIZACIÓN EN GESTIÓN ESTRATÉGICA DE SISTEMAS Y TECNOLOGÍAS DE LA INFORMACIÓN

Trabajo Final de Maestría

La gestión de la Deuda Técnica en los Sistemas de Información

Autor: Lic. Gabriel Belingueres

Director: Dr. Claudio Freijedo

Año de Presentación 2017

Cohorte 2014

Índice general

Índice de cuadros

Índice de figuras

Resumen

Durante el desarrollo de un sistema de información a menudo se toman decisiones cuya consecuencia es que ciertas tareas no sean apropiadamente realizadas. La deuda técnica puede pensarse como las tareas que no se han concluido al terminar el producto, pero que idealmente deberían ser concluidas antes de declararlo terminado.

Poseer un alto grado de deuda técnica puede producir dificultades en el cumplimiento de los plazos comprometidos por el equipo de desarrollo o deteriorar su productividad. También puede afectar negativamente en la calidad final del producto o servicio, ya sea porque los defectos son costosos de corregir, o bien porque el agregado de nueva funcionalidad introduce defectos en funcionalidad existente.

Por otra parte, muchas veces existen razones para priorizar alguna variable estratégica del negocio sobre la deuda técnica, convirtiéndola así en un blanco móvil que el CIO debe saber gestionar.

El presente Trabajo Final se enfoca en investigar el fenómeno de la deuda técnica, dando una definición y clasificación, cuales son los motivos de su presencia, los problemas que introduce en una gerencia de sistemas y cómo afecta a la empresa. Se aportarán elementos a tener en cuenta para gestionarla convenientemente, a la vez que se investigan métodos y herramientas de soporte disponibles para que el CIO pueda brindar una adecuada gestión, alineada a los objetivos y estrategias del negocio.

Palabras Clave: Deuda técnica, Mantenimiento de software, Tecnologías de la Información.

Dedicatoria y Agradecimientos

Dedicado a mi familia: Daniela, Martina y Marisel.

Agradezco a mi familia por todo su apoyo durante estos años de cursadas de martes y jueves por la noche y los sábados.

A mis padres, por ser un estímulo de auto superación y por su apoyo incondicional.

Al Consejo Profesional de Ciencias Económicas de la Ciudad Autónoma de Buenos Aires, institución a la que he estado vinculado laboralmente estos últimos 13 años, en particular la Mesa Directiva, el equipo de la Gerencia de Sistemas y el de Recursos Humanos, por su apoyo económico para realizar esta Maestría.

A mi director de Trabajo Final de Maestría, Prof. Dr. Claudio Freijedo, por su tiempo, consejos y revisiones para que este trabajo llegue a buen puerto.

Capítulo 1

Introducción

El término “deuda técnica” se refiere a una metáfora, una analogía proveniente del mundo financiero, y hace alusión a las consecuencias que producen las decisiones del trabajo no terminado apropiadamente en la calidad final de un producto o servicio. Este problema es muy común en el mundo del desarrollo de software, donde se realizan decisiones sobre el diseño y codificación a efectos de priorizar otros aspectos del negocio. Debido a las modificaciones que van sufriendo los sistemas a lo largo del tiempo, la deuda técnica puede ir creciendo hasta el punto extremo de que sea menos costoso reemplazar el software que seguir introduciendo modificaciones para adaptarlo a las necesidades del negocio.

Como consecuencia de la generación de la deuda técnica, pueden manifestarse los siguientes problemas:

Baja productividad. Realizar modificaciones sobre una base de software con deuda técnica conlleva más esfuerzo, lo que reduce la productividad del equipo de desarrollo.

Pobre calidad final del producto o servicio. Ante la presencia de una importante cantidad de deuda técnica, es más probable que el software desarrollado contenga una alta tasa de defectos, algunos difíciles de detectar y corregir.

Fragilidad ante los cambios. La modificación del software puede afectar o introducir defectos en funcionalidad ya existente.

Dificultad para cumplir los plazos comprometidos. Ante un alto grado de deuda técnica, realizar un dimensionamiento inicial del problema sin un conocimiento profundo de los artefactos a modificar puede producir estimaciones erróneas de esfuerzo y tiempo.[1]

Por lo enunciado anteriormente, se desea realizar una investigación sobre el fenómeno de la gestión de la deuda técnica en proyectos de desarrollo de software, en particular en los de desarrollo de los sistemas de información en los cuales se apoyan las empresas para su operación diaria y la toma de decisiones. Mediante el presente trabajo se pretende dar respuesta a las siguientes preguntas:

- ¿Qué es la deuda técnica y cómo se origina?
- ¿Cuales son las similitudes y diferencias entre la deuda técnica y un defecto de software? ¿Existen distintos tipos de deuda técnica?
- ¿La deuda técnica puede cuantificarse? ¿Puede traducirse a variables de negocio comprensibles por un directivo de una empresa (tiempo o costo)?
- ¿La presencia de deuda técnica siempre impacta negativamente? ¿O existen ocasiones en la cual la deuda técnica es justificable en relación a otras variables del negocio?
- ¿La deuda técnica se puede gestionar? ¿Puede considerarse una variable más que un CIO debe tener en cuenta al momento de la toma de decisiones?
- ¿Puede identificarse un conjunto de buenas prácticas de desarrollo de software que redunden en una mínima generación de deuda técnica?
- ¿La deuda técnica en los sistemas *legacy* debe ser tratada en forma diferente de la generada en los nuevos sistemas?

[1] La dificultad para cumplir con los plazos comprometidos no solo es una consecuencia, sino que también es una posible causa de la presencia de la deuda técnica, como se verá luego en la Sección 2.2.

- ¿La deuda técnica debe ser una variable de negocios que deben comprender y gestionar los directivos de las empresas? ¿O se trata de un tecnicismo que solo deben comprender los CIOs?

- ¿En relación a la estrategia de aprovicionamiento de la empresa, la deuda técnica debe gestionarse de igual manera ya sea si se abastece mediante *insourcing* o el *outsourcing*?

- ¿Cual es el estado del arte actual en métodos y herramientas que sirvan de apoyo para la toma de decisiones relacionadas a la deuda técnica?

1.1. Objetivos y Alcance

El objetivo general es abordar el problema de la aparentemente inevitable generación de deuda técnica en el ámbito del desarrollo de software dentro de las organizaciones. Como primer objetivo específico se procederá a definir en detalle el concepto de deuda técnica e investigar las causas de su presencia dentro de los sistemas de información en las empresas. Como segundo objetivo específico, se investigará como gestionarla adecuadamente de acuerdo a los objetivos y estrategias de la organización. Como tercer y último objetivo específico, se describirán algunos de los métodos y herramientas que sirven de apoyo para la toma de decisiones sobre la gestión de la deuda técnica para desarrollos de sistemas de información realizados con tecnología Java, describiendo el estado del arte actual.

En la experiencia del autor, quien se ha desempeñado en distintas empresas y posiciones en el área del desarrollo de software, entiende que los problemas descritos son comunes y relevantes en todas las organizaciones, tanto PYMEs como grandes empresas, tanto del sector público como privado, de modo que el conjunto de recomendaciones obtenidas a partir de este trabajo pueda servir como guía a un CIO o gerente en la resolución de problemas para gestionar su Gerencia de Sistemas.

El alcance del trabajo estará circunscrito en dar respuesta a las preguntas problemáticas enunciadas anteriormente. Los métodos y herramientas

que representen el estado del arte en la gestión de la deuda técnica que serán investigadas estarán limitadas a las que sean aplicables en sistemas desarrollados con tecnología Java (en la cual el autor tiene 19 años de experiencia), no obstante esto no implica, desde el punto de vista de la gestión de un CIO, que las soluciones ofrecidas por los métodos y herramientas no puedan ser útiles y extrapolados al uso de otras tecnologías de desarrollo (por ejemplo: .NET, PHP, etc.)

1.2. Justificación y Relevancia

El resultado del presente trabajo es relevante porque las empresas dependen cada vez en mayor medida de los sistemas de información que dan soporte al negocio, tanto de índole operativo como otros que ofrecen información más relevantes para fines estratégicos. Resulta indispensable que estos sistemas puedan ser desarrollados en tiempos razonables, que posean la calidad adecuada para una confiable operación diaria, y que se comporten en forma robusta ante cambios en su funcionalidad.

En la medida en que se tensan los tiempos de entrega, costos o alcance (la triple limitación de los proyectos), el ejercer presión sobre algunas de estas variables dispara una generación de deuda técnica que resulta prácticamente inevitable, por lo general debido a que es una variable solo conocida por el equipo de desarrollo y no suele trascender hacia el cliente.

Asimismo, si bien la deuda técnica tiene una connotación negativa, no siempre su presencia se considera mala en tanto su generación dentro del proceso de desarrollo del software obedezca a priorizar otras necesidades del negocio de manera que esas necesidades no se vean afectadas. Por ejemplo, si el objetivo de la empresa es ser el primero en el mercado en ofrecer cierto producto o servicio. Por ello, es necesario que el CIO de una empresa brinde una gestión adecuada de la deuda técnica a efectos de alinearse a los objetivos y estrategias del negocio.

1.2.1. Costos asociados a la Deuda Técnica

En el Informe sobre Salud de Software de Aplicación 2011/12 (Sappidi y cols., 2012), realizado por la firma *CAST Software*, donde se analiza el código fuente de 745 aplicaciones, se encontró que existe en promedio $3,61 dólares por LDC[2] de Deuda Técnica, esto es, una pequeña aplicación de 10000 LDC acarrea un costo promedio de $36100 dólares en concepto de Deuda Técnica. En el estudio también se encontró que el 30 % de la Deuda Técnica representa un riesgo potencial para el negocio debido a que se refiere a problemas de robustez, performance y seguridad, mientras que el 70 % evidencia problemas de mantenimiento (específicamente la dificultad de modificar el software o transferir el mantenimiento del mismo a otro equipo de desarrollo).

1.2.2. Concientización

En una reciente encuesta (Ernst y cols., 2015) contestada por profesionales de desarrollo de software con un promedio de 6 años de experiencia, encontraron que el 79 % está de acuerdo o muy de acuerdo que la "falta de concientización es un problema" (Figura 1.1). Los autores atribuyen este resultado a que la población encuestada son *early adopters* del concepto de deuda técnica (concepto que está en una etapa de crecimiento en el campo de la ingeniería de software) y por tanto es de gran importancia "convencer a gerentes y otros interesados en la proposición de valor que aporta la gestión de la deuda técnica."

Otra encuesta (Ambler, 2015) también revela la falta de concientización entre ejecutivos senior de negocios. En ella se pedía que calificaran en el rango de -10 (muy inconsciente) a 10 (muy consciente) sobre cuán consciente estaban los distintos roles en el negocio e involucrados en el proceso de desarrollo, arrojando los resultados de la Figura 1.2.

En este sentido, el presente trabajo intenta ofrecer al lector las herramientas necesarias para transmitir el concepto hacia niveles directivos y gerenciales, así como servir como proceso de inducción hacia niveles tácticos y

[2]LDC: Líneas de Código.

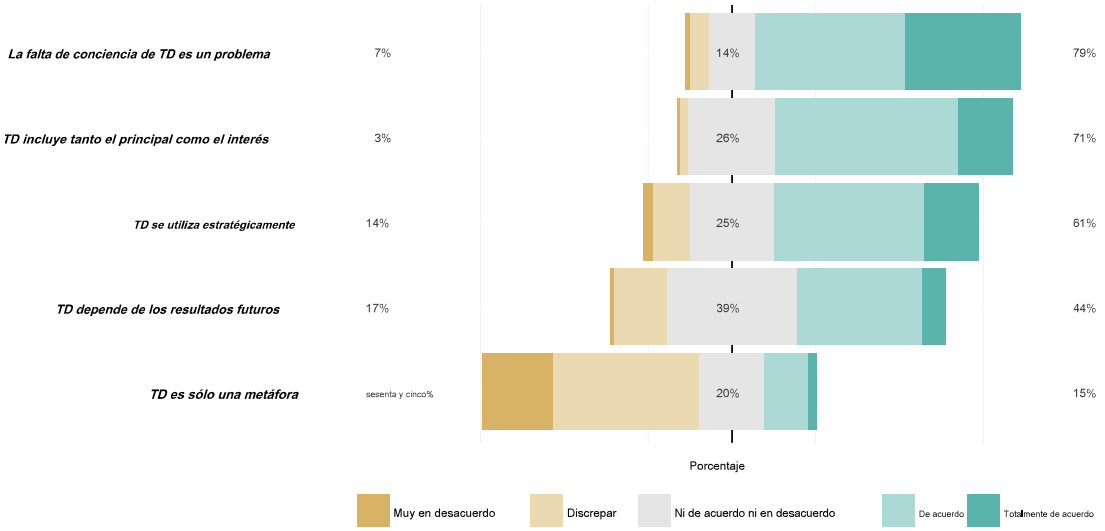


Figura 1.1: Deuda Técnica: 79 % respondieron que la “falta de concientización” es un problema. Fuente: (Ernst y cols., 2015).

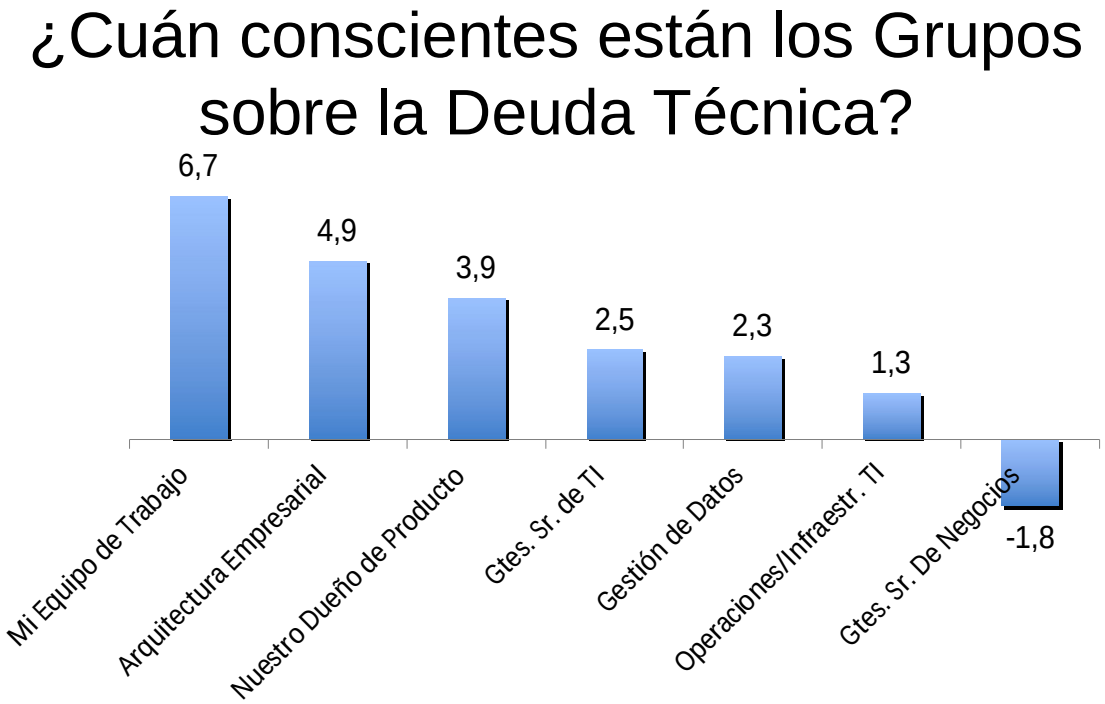


Figura 1.2: La “falta de concientización” entre los gerentes de negocio. Fuente: (Ambler, 2015).

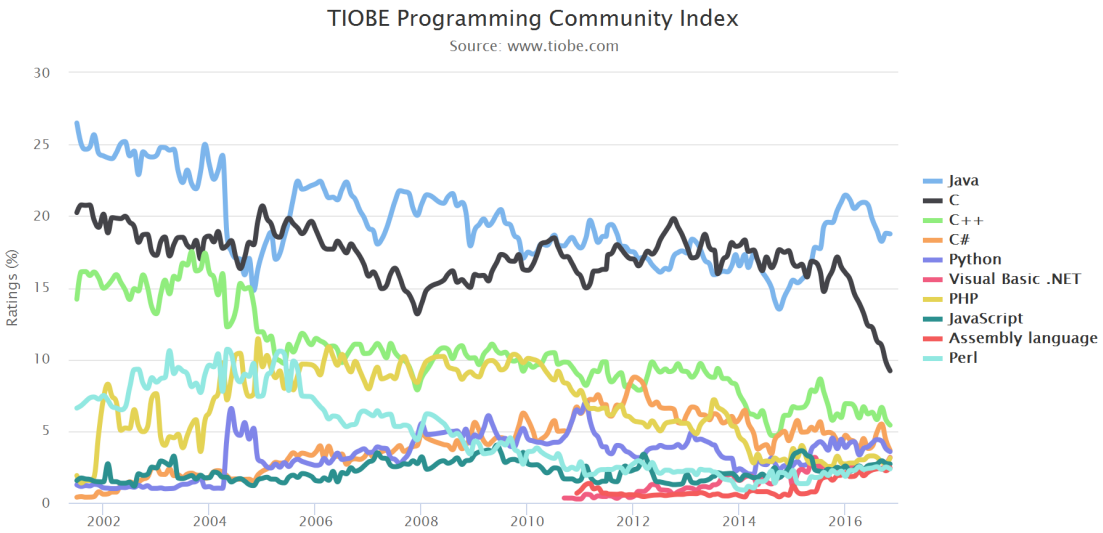


Figura 1.3: Índice TIOBE de popularidad de lenguajes de programación, desde 2002 a Noviembre/2016. Fuente: TIOBE.

operativos con menor experiencia.

1.2.3. El lenguaje Java y JavaEE

Con respecto a la elección del lenguaje Java como referente para el presente trabajo, es relevante debido a que Java es el lenguaje de programación más popular en Internet, encabezando el índice TIOBE mostrado en la Figura 1.3, que muestra los índices de popularidad de los diez lenguajes de programación con más búsquedas en Internet.[3]

Por otro lado, se encontró que la plataforma JavaEE (*Java Enterprise Edition*), que representa el conjunto de tecnologías estandarizadas para el desarrollo de software empresarial usando el lenguaje Java, obtuvo el puntaje más alto en el Informe sobre Salud de Software de Aplicación 2011/12 (Sappidi y cols., 2012), promediando los $5,42 dólares por LDC (ver Figura 1.4). Otro dato interesante es la gran variabilidad en la distribución de Deuda Técnica en JavaEE detectada en el informe. Como JavaEE es una de las plataformas más utilizadas en el mercado, resulta importante hacer foco en las aplicaciones desarrolladas con dicha tecnología el momento de estudiar la gestión de la Deuda Técnica.

[3](TIOBE Software, s.f.) describe la metodología de construcción del índice.

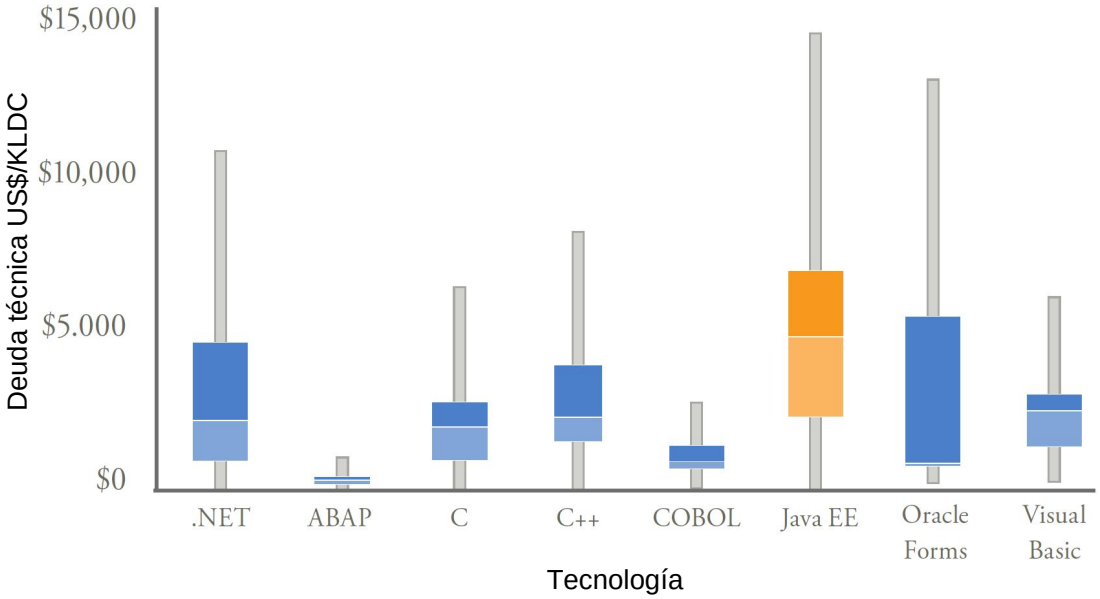


Figura 1.4: Deuda Técnica por tecnología. Fuente: CRASH Report 2011/2012 (Sappidi y cols., 2012).

1.3. Metodología

La presente investigación se abordará desde una perspectiva descriptiva y explicativa.

En primera instancia se realizó una investigación descriptiva, que involucra la recolección de datos mediante la observación, experiencia propia y lectura tanto del material bibliográfico como de fuentes secundarias.

En segunda instancia, se procedió en forma explicativa con el objeto de ilustrar las causas que generan deuda técnica en los sistemas, así como aportar cómo con una gestión adecuada de la deuda técnica es posible conducir a la empresa a desarrollar soluciones de software con los atributos cualitativos necesarios para suplir a la organización con sistemas de información que estén alineados a los objetivos y estrategias del negocio. Este aporte a la gestión representa la proposición de valor central de este trabajo, enfocando la solución desde la experiencia propia y alineada a la investigación bibliográfica realizada.

Por último, la obra no está libre de las limitaciones típicas de las investigaciones descriptivas, esto es, no se presenta evidencia cuantificable y es-

tadísticamente significativa sobre los resultados de aplicar la metodología de gestión aportada, lo que requeriría un esfuerzo considerablemente mayor en relación al alcance de este Trabajo Final de Maestría. En otras palabras, no se presentan respuestas definitivas al fenómeno, así como tampoco se pretende validar o refutar hipótesis.

Capítulo 2

La deuda técnica

El concepto de deuda técnica apareció por primera vez en un informe de experiencias en la OOPSLA[1] durante el año 1992. El informe, presentado por Ward Cunningham (Cunningham, 1992) explicaba lo que ocurría en el proyecto *WyCash*, un software de gestión financiera:

> La primer entrega del código es como entrar en una deuda. Un poco de deuda acelera el desarrollo siempre y cuando se devuelva rápidamente con una re-escritura. Los objetos hacen que el costo de esta transacción sea tolerable. El peligro ocurre cuando la deuda no es pagada. Cada minuto gastado en código que no sea del todo correcto cuenta como interés de esa deuda. Organizaciones de ingeniería enteras pueden ser paralizadas bajo una montaña de deuda de una implementación no consolidada, ya sea orientada a objetos o no.

Con esta metáfora, Cunningham le explicaba a sus superiores los problemas que estaba teniendo el proyecto utilizando términos que ellos pudiesen entender. Teniendo en cuenta que se trataba de un software financiero, utilizó el concepto de un préstamo.

En un préstamo, ante la necesidad de contar con una cantidad de dinero en forma rápida, se acuerda con una entidad financiera la obtención de cierto

[1] *Object-Oriented Programming, Systems, Languages and Applications* (OOPSLA), es una conferencia anual de investigación organizada por la *Association of Computing Machinery* (ACM).

monto con disponibilidad inmediata (llamado el Capital), con la obligación de devolver dicha suma de dinero en un plazo de tiempo acordado, junto con una suma de dinero determinada por una tasa de interés. Como consecuencia de lo anterior, existe una utilidad genuina en solicitar dinero prestado, aunque mientras no se pague totalmente la deuda, se estarán pagando intereses.

Lo que ocurría en el proyecto es que se hacían entregas tempranas a efectos de poder aprender del software y mostrar su progreso, pero las cosas que se aprendían o que necesitaban mejorarse no se estaban realizando apropiadamente para incorporar el nuevo conocimiento al software, dificultando así el agregado de nueva funcionalidad o el mantenimiento de la funcionalidad existente.

La analogía compara la entrega temprana con la solicitud del préstamo, la dificultad extra de realizar modificaciones al software sin introducir las correcciones necesarias con el pago del interés de la deuda, y el pago completo de la deuda cuando las correcciones y trabajos necesarios se han incorporado al software. Básicamente, lo que Cunningham sugería es que en tanto no se incorporen al código fuente del software las correcciones necesarias, las futuras modificaciones demandarán más esfuerzo.

En este Capítulo se abordará el primer objetivo específico de esta obra, esto es, brindar una definición más detallada del concepto, junto con las causas de su presencia.

2.1. Calidad del Software

Comúnmente cuando se hace referencia a la Calidad de un producto o servicio de software se adopta el punto de vista del cliente[2], es decir, la calidad queda definida por los atributos del software que el cliente puede observar o percibir. En base a lo que el cliente puede experimentar, éste realizará una apreciación de la calidad del software, ya sea medible mediante algún indica-

[2]En el contexto de este trabajo, los clientes pueden ser cualquier entidad física o jurídica que haga uso del software. Por ej. una empresa, clientes internos de una organización, usuarios del sistema o inclusive un departamento externo de testeo funcional del software.

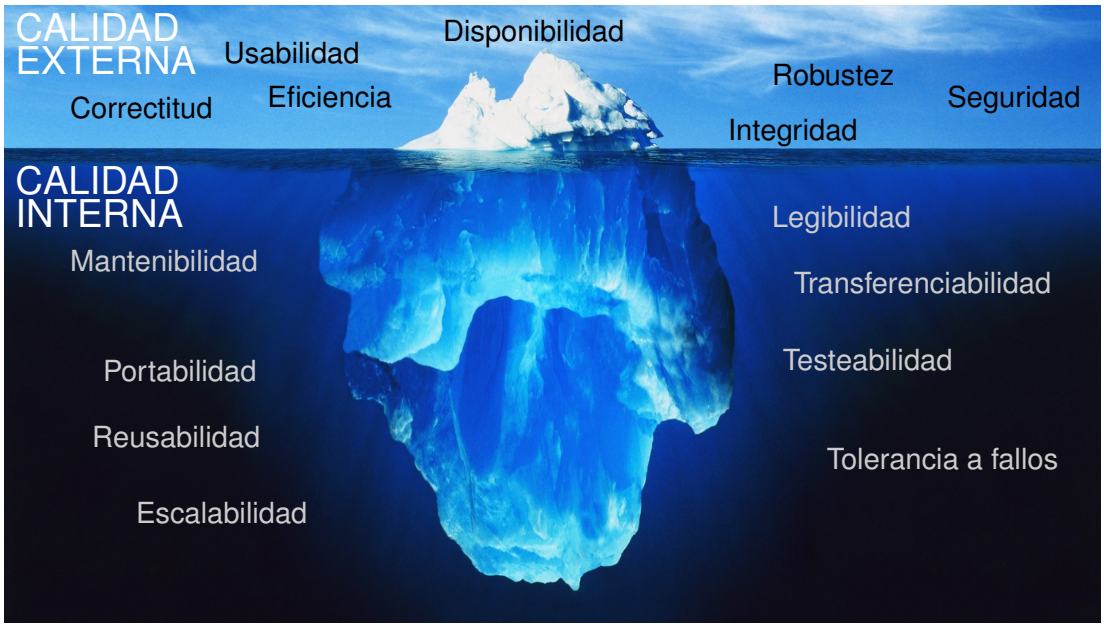


Figura 2.1: Calidad Externa e Interna

dor o bien informalmente describiendo la sensación de realización o el grado de satisfacción con respecto al nivel con que cumple con sus necesidades.

En general, las características del software que percibe el cliente son las resultantes de la ejecución y operación del mismo. Al cliente le interesa que sea fácil de usar pero no le interesa si el software es fácil de modificar. Le interesa que pueda ejecutar sobre una nueva versión del Sistema Operativo pero no le interesa si el código es portable entre diversas plataformas. O bien le interesa que una misma funcionalidad esté disponible en múltiples sistemas pero no le interesa si el código es reusable.

En este sentido, McConnell (2004, p. 463) advierte que el software posee características de calidad tanto externas como internas, como se describen en las dos siguientes subsecciones, y ejemplificadas en la Figura 2.1 mediante la imagen de un iceberg.

2.1.1. Calidad externa

La noción de calidad externa está vinculada con la satisfacción del cliente en cuanto a cómo se ajusta el software a sus necesidades y requerimientos. La siguiente es una lista de algunos atributos que influyen positiva-

mente a la calidad externa del software:

- Correctitud: Implementa correctamente las funcionalidades.
- Usabilidad: Es intuitivo y fácil de utilizar.
- Eficiencia: Resuelve las operaciones utilizando una cantidad adecuada de recursos.
- Disponibilidad: Es posible utilizarlo en el momento en que se requiera.
- Integridad: La información gestionada es confiable y precisa.
- Seguridad: La información es accesible solo por las personas autorizadas.

2.1.2. Calidad interna

La calidad interna representa el conjunto de atributos del software que es invisible al cliente, ya sea porque no tiene acceso a ellos, porque no son evidenciables por medio de la simple utilización del software, o bien porque sencillamente no le resultan interesantes. Las siguientes son algunas de las características de calidad interna del software:

- Mantenibilidad: La facilidad con la que el software puede modificarse para agregar nuevas capacidades, corregir defectos.
- Portabilidad: La facilidad de modificar el software para funcionar en ambientes para los cuales no fue diseñado.
- Reusabilidad: El grado en el cual puede utilizarse parte del software en otros sistemas.
- Legibilidad: La facilidad con la cual el desarrollador puede leer y entender el código fuente.
- Testeabilidad:[3] El grado en el cual el software está diseñado para poder someterse a pruebas y verificar su funcionamiento.

[3] Del inglés *testability*.

	Externa	Interna
Positivo	Funcionalidad	Arquitectura
Negativo	Defecto	Deuda Técnica

Cuadro 2.1: Valor de negocio versus Calidad (Kruchten, 2011, p. 6).

- Transferenciabilidad:[4] La facilidad con que un nuevo equipo de desarrollo puede entender el software y rápidamente tornarse productivo.

2.1.3. Relación entre valor de negocio y calidad

El valor que aporta el software al negocio depende del comportamiento que manifieste tanto hacia sus usuarios como hacia sus desarrolladores. Esto es, de cara al usuario existirá comportamiento que contribuya positivamente al negocio, y otros que contribuyan negativamente (calidad externa). De igual manera, de cara a los desarrolladores expondrá características internas positivas y otras negativas (calidad interna).

Estos comportamientos son tipificados en el Cuadro 2.1, adaptado de Kruchten (2011, p. 6). En cada cuadrante se ubican las cualidades del software dependiendo de si aportan un valor positivo o negativo al negocio, y si es una característica de calidad interna o externa (es decir, es directamente visible por el cliente o es invisible).

Las características de calidad externa del software, como se explicó, tienen un impacto directo en la satisfacción del cliente. Entre las que aportan

[4] Del inglés *transferability* (Sappidi y cols., 2012, p. 3).

positivamente al negocio están las funcionalidades del software, mientras que los defectos, no conformidades o *bugs* tienen un impacto negativo.[5] De la misma manera, las características de calidad interna (si bien no tienen un impacto directo en la satisfacción del cliente) pueden aportar positivamente al negocio (como es el caso de la arquitectura) o negativamente, como es el caso de la deuda técnica.

Como resultado, se observa que no es lo mismo un defecto en una funcionalidad determinada del software que la deuda técnica que se genera debido a una programación descuidada o no del todo correcta de esa misma funcionalidad. Mientras ambas tienen consecuencias negativas, sus causas pueden diferir. La deuda técnica podría ser una causa posible de la presencia del defecto (entre otras, por ej. si el requerimiento no fue bien especificado), mientras que lo opuesto no ocurre.

2.1.4. Relación entre las expectativas del cliente y calidad

Las expectativas iniciales que el cliente tenga sobre el software entregado juegan un rol importante. A veces las expectativas son tan altas que un sistema que sea adecuado a sus necesidades puede ser percibido como poco exitoso.

Particularmente en contratos de terciarización, Taylor (2006) identifica dos riesgos que son difíciles de gestionar por parte del proveedor: (1) cronogramas y presupuestos demasiado optimistas, y (2) expectativas exageradas por parte del cliente. Estos riesgos son típicamente causados por el proveedor con el objetivo de ganar el contrato de trabajo, aunque, al mismo tiempo, ingenuamente el cliente cree ciegamente en las estimaciones y promesas del proveedor sin haber realizado una valoración propia del esfuerzo necesario.

En el caso de que el proyecto haya sido aprobado con un cronograma demasiado ajustado o bien con un presupuesto demasiado acotado, es común que el proveedor decida exigir a su equipo de desarrollo que intensifique su

[5] Existen casos en que una funcionalidad puede impactar negativamente, por ej. el caso Volkswagen, con su algoritmo que detecta si el motor está operando normalmente o está siendo testeado por emisión de óxidos de nitrógeno (Environmental Protection Agency, 2015).

trabajo a efectos de alcanzar los hitos del proyecto, cumpliendo además con todas las funcionalidades y características de calidad externa contratadas o bien mesurables fácilmente por el cliente. Este stress implica normalmente el deterioro de las características de calidad internas del software, con su consiguiente generación de deuda técnica.

Por otro lado, si el proveedor ha generado demasiadas expectativas en el cliente, la calidad del producto final debe poder ser percibida fácilmente por el cliente y estar a la altura de lo prometido, caso contrario el proveedor resultará con un problema de credibilidad, pérdida de confianza y otros problemas de imagen. Se debe procurar relevar los requerimientos lo más sensata y responsablemente posible para que la funcionalidad, las estimaciones de cronogramas y presupuestos sean realistas. Esto rara vez se cumple cuando el proveedor tiene un equipo de pre-venta que negocia el alcance, cronogramas y presupuestos, mientras que otro equipo de desarrollo debe ejecutar el proyecto.

Como corolario, una inadecuada gestión de las expectativas del cliente, además de erosionar la relación entre cliente y proveedor, puede llevar a la generación de deuda técnica. Prometer la generación de resultados positivos en exceso eleva el nivel de expectativas del cliente, que puede derivar en una presión adicional sobre el equipo de desarrollo, debido a que la funcionalidad con la calidad especificada deba ser desarrollada en un lapso de tiempo más reducido del necesario, incurriendo en atajos que afectan la calidad interna del producto final.

2.1.5. Relación entre deuda técnica y calidad

Tanto los proveedores del software como los clientes deberían preocuparse por la calidad interna, debido a que en algún punto ésta comenzará a afectar a la calidad externa. Por ej. si el software sufre de problemas de mantenibilidad (agregar nuevas funcionalidades requiere mucho esfuerzo) entonces posiblemente sufra de problemas de correctitud (la funcionalidad no cumple por completo con las expectativas, o se ha introducido algún bug en funcionalidad existente). Si el software no escala lo suficientemente para lidiar

con la demanda de accesos de usuarios (escalabilidad), en algún momento se tornará muy lento o colapsará (disponibilidad).

El problema es que lo expuesto arriba no siempre se cumple. Por lo general la satisfacción del cliente se verá alcanzada dependiendo de la percepción del valor que el software le aporte a su negocio, en conjunto con las características de calidad externas del mismo.

Lo que normalmente ocurre cuando el proveedor se siente presionado por entregar el proyecto, es que se va relajando la calidad de aquellos aspectos que no han sido de particular interés para el cliente, típicamente aspectos de calidad interna. Gradualmente se van deteriorando el diseño y estructura del software a efectos de cumplir en forma oportuna con las entregas pautadas, lo que causa problemas de legibilidad, mantenibilidad e inclusive la introducción de defectos, debido a que etapas importantes como las de pruebas no se realizan con la profundidad necesaria, entre otros. En otras palabras, el software va acumulando Deuda Técnica.

En definitiva, la deuda técnica provee un hilo conductor que une la calidad del software y la satisfacción del cliente. A medida que la deuda técnica se incremente, se irán degradando los atributos de calidad interna, lo que a su vez, con el tiempo se percibirá como una degradación en los atributos de calidad externa, afectando el valor que el software aporta al negocio y en última instancia a la satisfacción del cliente.

Gestionando adecuadamente la deuda técnica puede llegarse a un punto de equilibrio entre los recursos necesarios para obtener la calidad deseada y el valor de negocio que se intenta alcanzar.

2.2. Causas de la deuda técnica

Diversas pueden ser las razones o causas de la generación de deuda técnica. A continuación se enunciarán las que aparecen más frecuentemente.

2.2.1. Presiones del proyecto o negocio

Los proveedores de software ya sean internos o externos, como cualquier otro negocio, se esfuerzan para alcanzar el objetivo principal: la satisfacción del cliente. A tal efecto, deben cumplir con las restricciones del proyecto. El PMBOK (Project Management Institute, 2013, p. 6) especifica (en forma no exhaustiva) las siguientes restricciones: Alcance, Calidad, Cronograma, Presupuesto, Recursos y Riesgo. Estas restricciones son variables que están en tensión, esto es, modificar alguna implica un cambio en otras. Por ej. al aumentar el Alcance (agregando una nueva funcionalidad) implica un aumento en el Cronograma (más tiempo para desarrollarla) o bien un aumento en el Costo (contratar más personas en el equipo de trabajo).

Es común (como se mencionó previamente, en referencia al estudio de Taylor) que al momento de ejecutar el proyecto, el equipo de desarrollo se encuentre con que los requerimientos no están claramente especificados, o bien el equipo de ventas lo ha sub-presupuestado, o bien ha generado en el cliente expectativas tan altas que hacen que un software adecuado para sus necesidades se perciba como un incumplimiento. En este sentido, cualquier negociación posterior pondrá en riesgo la imagen del proveedor, con la consiguiente pérdida de confianza del cliente.

Esto último implica que el equipo de desarrollo tendrá la labor de cumplir con todas las restricciones del proyecto de la manera que sea. Puesto que exteriorizar el problema hacia el cliente suele ser una última opción, el equipo tratará de cumplir relajando los atributos de calidad que el cliente “no ve” o bien no ha expresamente especificado. Esto se traduce en que habrá ciertas tareas que no se realizarán completamente o se realizarán pobremente (por ej. arquitectura, diseño, bases de datos, pruebas), también es posible que la primer versión de los requerimientos “que funcione” no sea luego revisada para mejorar la mantenibilidad del código fuente. A su vez, se deseará que el código fuente que esté en funcionamiento no sea modificado, introduciendo así bloques duplicados con el consiguiente deterioro de su estructura.

2.2.2. Escasez de recursos

En el contexto actual del mercado, existe una gran cantidad de puestos de trabajo que no pueden ser cubiertos debido a la escasez de profesionales de TI que afecta al crecimiento del sector (OPSSI, 2016, p.16). Como resultado, esta restricción hace que proyectos que necesiten desarrollarse con más personal (para satisfacer las metas de alcance y tiempo requeridas) se ejecuten con una cantidad menor, lo que tiende a aumentar el estrés y las horas de trabajo asignadas al personal.

Esta presión extra generada sobre el personal hace que el equipo no tome los recaudos necesarios para que las cualidades internas del software sean adecuadas, limitándose a generar la funcionalidad acordada de la manera más directa y rápida. Woodard, Ramasubbu, Tschang, y Sambamurthy (2012) citan la munificencia de recursos como factor clave en el diseño de soluciones con baja deuda técnica.

2.2.3. Falta de experiencia

Simplemente, hay organizaciones donde los equipos de trabajo no conocen el concepto de deuda técnica y se toman decisiones sin considerar sus consecuencias sobre el producto final. Por ejemplo, en un equipo donde todos los programadores tienen escasa experiencia laboral es probable que se genere deuda inadvertidamente.

2.2.4. Falta de colaboración

En organizaciones donde no hay una cultura de compartir conocimientos, los miembros de un equipo no logran comunicarse abiertamente y por lo tanto el *know-how* queda confinado a ciertos sectores, lo que hace que se pierda eficiencia. Si los miembros no colaboran entre si pueden generarse duplicaciones de funcionalidad en los sistemas.

Asimismo, cuando no hay entre los miembros del equipo un sentido de ser “dueños” del sistema, entonces sufre la comunicación y la colaboración, con el efecto de que la calidad del código fuente del sistema no sea de par-

ticular interés para nadie, eliminando el incentivo para no generar deuda. En este sentido, Fowler (2006) sugiere que la *apropiación fuerte* (donde se asigna cada módulo a un desarrollador, quien es el único que puede realizar modificaciones en ese módulo) es el más problemático, mientras que en el otro extremo del espectro la *apropiación colectiva* (donde el código fuente es compartido por el equipo completo y cada integrante puede realizar modificaciones donde necesite) resulta lo más adecuado para metodologías ágiles como XP (Beck y Andres, 2004).

2.2.5. Falta de patrocinantes u objetivos claros

La carencia de patrocinantes o *sponsors* claramente identificables durante el desarrollo del proyecto, puede ser causa de pérdida de impulso y decaimiento en la moral del equipo de trabajo, lo que puede impactar en la calidad del trabajo realizado, con la consecuente generación de deuda técnica. En este sentido, es necesario que el personal de gestión dé claras señales de compromiso ante el equipo (*management commitment*).

Asimismo, si los objetivos que se pretenden alcanzar con el desarrollo del software son confusos, entonces el equipo de trabajo tal vez decida enfocar esfuerzos en los lugares equivocados, en todos o bien en ninguno. Cuanto más tarde se disipen las dudas sobre los objetivos que se pretenden alcanzar, es posible que la deuda técnica se vea incrementada, en compensación por el tiempo perdido anteriormente en tareas que no eran prioritarias o las más relevantes con respecto al cumplimiento de los objetivos. En definitiva, es importante el valor de definir los objetivos y metas que se desean alcanzar, así como su comunicación efectiva al equipo de trabajo.

2.2.6. *Time to Market*

La causa de la deuda técnica puede no obedecer a forzar el cumplimiento de un contrato, sino a participar en un mercado lo antes posible, a efectos de posicionarse primero y de esa forma poder generar una ventaja competitiva frente a posibles futuros competidores.

Los pequeños emprendimientos o empresas *startup*, en su afán por llegar primeras a cierto mercado y lograr un rápido crecimiento, les interesa introducir productos y servicios con velocidad. Están más enfocadas en encontrar un flujo de ingresos que les permita crecer que una empresa que ya tenga un portafolio de productos o servicios bien establecidos. Mientras las primeras se encuentran en una búsqueda constante de oportunidades, las segundas tienden a enfocarse en una operación más eficiente, por lo que las causas de deuda técnica son diferentes.

Cuando se quiere crear productos o servicios innovadores, la atención está puesta en el éxito comercial de la idea, de modo que es frecuente encontrar un ciclo de aprendizaje más rápido en una startup que en una empresa más grande. En estas iteraciones de aprendizaje de una startup suele ser más importante experimentar y validar la viabilidad del negocio que la generación de deuda técnica.

2.2.7. Presentación de un producto

Las razones para acelerar el desarrollo de un producto sin considerar la calidad interna del mismo pueden obedecer al aprovechamiento de una oportunidad concreta, la cual puede tener una ventana de tiempo de duración limitada. Tal es el caso de la presentación de un nuevo producto o servicio en una convención de negocios (Lim, Taksande, y Seaman, 2012).

En ese contexto solo se intenta mostrar las capacidades a un público presente, pero no es necesario para ese momento que el producto sea fácilmente modificable o que sea escalable a millones de usuarios. Aquí la deuda técnica se usa como una estrategia para enfocarse en las cualidades que produzcan mayor impacto a corto plazo.

2.2.8. Asegurar capital de inversores

Con el objetivo de asegurar fondos aportados por inversores (Lim y cols., 2012), se trabaja en un prototipo de las funcionalidades completas a efectos de mostrar las capacidades o demostrar el potencial de mercado del

producto o servicio. No es necesario hacer énfasis en la mantenibilidad del software ya que se trata de un producto que representa una prueba de concepto. Aquí la generación de deuda técnica se usa inteligentemente como una herramienta porque en este momento el futuro del producto es incierto.

2.3. Fuentes de deuda técnica

La deuda técnica puede adoptar diferentes formas, puede tratarse de actividades que no se han realizado o realizado pobremente, o bien puede estar presente en artefactos producidos durante diferentes etapas del proceso de desarrollo. Los lugares, actividades o artefactos afectados constituyen distintas fuentes de deuda técnica. En contraste, las causas de la deuda técnica (como se describió en la Sección anterior) representan el motivo o causa raíz de la generación de la deuda. En general, las fuentes de deuda técnica serán específicas de la organización, proyecto o software en particular.

Sin embargo, en una encuesta realizada durante el año 2015 (Ernst y cols., 2015), se solicitó a los participantes que ordenen un listado de 14 opciones “con respecto a la cantidad de deuda (1=alta, 14=baja) que representan en el proyecto”, donde las opciones representan posibles fuentes de deuda.

Como resultado, la Figura 2.2 muestra las calificaciones obtenidas, evidenciando (al menos en esa población en particular) una fuerte opinión en que malas decisiones de arquitectura son fuente de deuda técnica, seguidas en menor nivel por código complejo y falta de documentación del código fuente, entre otras. Debido a que las decisiones de arquitectura de software se han tomado mucho tiempo atrás (con respecto al momento en que comienzan a evidenciarse), y porque tiene un impacto profundo en la estructura del software, esta fuente de deuda es una de las más complejas de resolver.

2.4. Definiciones

A efectos de estudiar más a fondo la metáfora, es conveniente definir la Deuda Técnica en forma más precisa. Con una perspectiva económica,

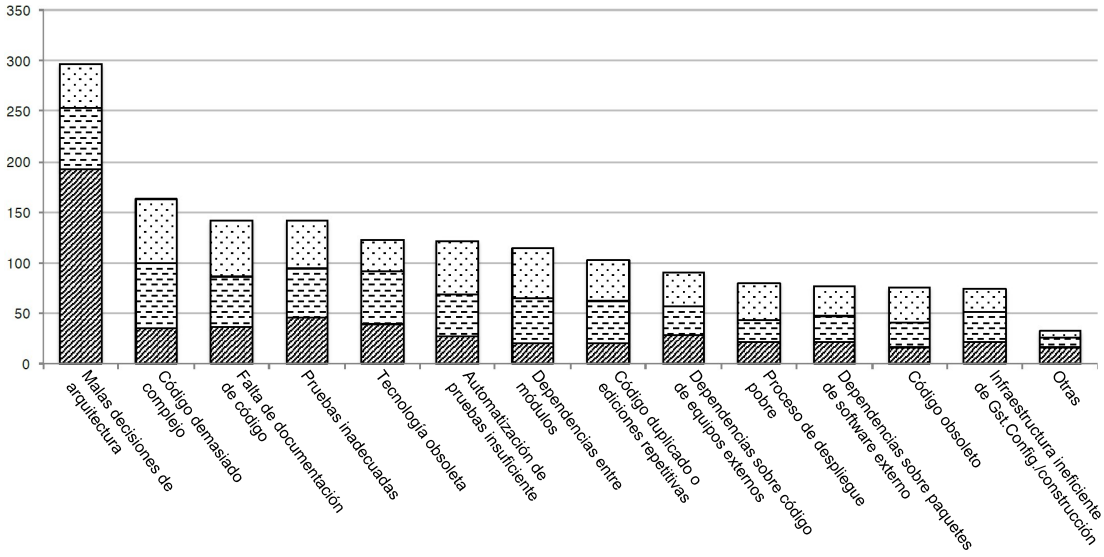


Figura 2.2: Fuentes de deuda técnica: Opción 1 representada por líneas oblicuas, Opción 2, guiones; Opción 3, puntos. Extraída de Ernst y cols. (2015).

Nugroho y cols. (2011) la define como “el costo de reparar los temas de calidad en sistemas de software para alcanzar un nivel ideal de calidad”.

Jones y Bonsignour (2011) ofrecen una visión más orientada al negocio, definiendola como “el costo de solucionar problemas de calidad estructural en una aplicación que, si no se reparan, pone al negocio en un riesgo serio”. Además, clarifican que no incluye todos los problemas, sino aquellos que tienen una alta probabilidad de causar disrupciones en el negocio.

Otra definición, con una perspectiva de desarrollo de software la enuncia como “los costos futuros atribuibles a las violaciones conocidas en el código en producción que debería ser arreglado–un costo que incluye tanto el capital como el interés” (Curtis, Sappidi, y Szynkarski, 2012).

La definición que aporta McConnell (2007) se centra en las consecuencias de la toma de decisiones, indicando que “es la obligación que incurre una organización de software cuando elije un enfoque de diseño o construcción conveniente en el corto plazo pero que incrementa la complejidad y es más costosa a largo plazo”.

Esta última es la definición que se adoptará en el presente trabajo, ya que es la que más se ajusta al enfoque de gestión que se pretende dar a la temática.

2.5. Capital e Interés

Los elementos esenciales de la metáfora de la Deuda Técnica son el Capital (o *Principal* en inglés) y el Interés.

El Capital indica la magnitud de la brecha entre la calidad actual versus la calidad ideal. Puesto que la literatura no siempre lo especifica, cabe aclarar que esta magnitud numérica que suele establecerse en términos de costo o esfuerzo (horas-hombre), no es otra cosa que la cantidad o monto de Deuda Técnica. En otras palabras, la Deuda Técnica como metáfora se materializa en un guarismo llamado Capital, que es la *cantidad de Deuda Técnica* detectada.

Asimismo, el Interés representa el esfuerzo extra realizado a consecuencia de no "pagar" la Deuda Técnica, esto es, de no corregir los temas de calidad hasta alcanzar el nivel ideal. Diversos autores proporcionan distintas definiciones de estos conceptos.

Fernández-Sánchez, Garbajosa, y Yague (2015) definen el Capital como "el costo que debería ser pagado para eliminar la deuda técnica", y el Interés como "el costo extra que debe ser pagado en el tiempo si la deuda técnica no es eliminada".

Ampatzoglou, Ampatzoglou, Chatzigeorgiou, y Avgeriou (2015), por su parte, centran su definición en el diseño del software, y define como Capital "el esfuerzo que es requerido para dirigir la diferencia entre la calidad de diseño actual y en su nivel óptimo, en un artefacto de software inmaduro o sistema de software completo", en tanto que el Interés es "el esfuerzo adicional que se necesita gastar para mantener el software, debido al decaimiento en el nivel de calidad de diseño".

Otra definición, pero enfocada en elementos de arquitectura y prácticas de codificación, se encuentra en Curtis y cols. (2012), que describe el Capital como "el costo de remediar las violaciones en el código de producción", el Interés como "los costos continuos atribuibles a violaciones en el código de producción que no ha sido remediado, tales como más horas de mantenimiento y uso ineficiente de recursos", donde por *violaciones* se refiere a violaciones de buenas prácticas de arquitectura y codificación que son conocidas por tener una probabilidad inaceptable de contribuir a problemas operativos severos

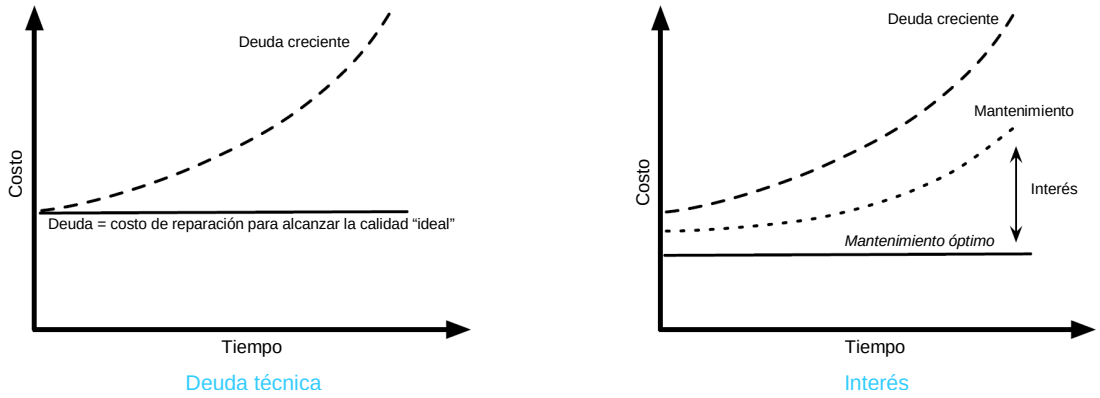


Figura 2.3: Deuda Técnica y su Interés. Extraída de Nugroho y cols. (2011).

o tener que aplicar un esfuerzo excesivo para implementar cambios.

Nugroho y cols. (2011), con su visión económica, aporta una definición para el Interés como "el costo de mantenimiento extra gastado por no alcanzar el nivel de calidad ideal", pero no hace lo mismo para el Capital, aunque con su perspectiva centrada en los costos, podría sugerirse que la definición para el Capital coincide con la de Deuda Técnica (vea sección anterior de Definiciones).

En la Figura 2.3 puede apreciarse, según los últimos autores, la Deuda Técnica y el Interés: a la izquierda se compara el costo del mantenimiento en relación a una calidad ideal (línea continua) versus el costo con la calidad real (línea intermitente), resultando en una diferencia de costos que representa la Deuda Técnica (o el Capital a pagar). Mientras que el gráfico de la derecha muestra el Interés que se va acumulando (línea punteada) a causa de no pagar la deuda, lo que lleva a un aumento de los costos de mantenimiento.

2.6. Clasificación de deuda técnica

No todo lo identificado como deuda técnica se refiere a las mismas deficiencias en la calidad del software, sino que distintas clases de problemas pueden tipificarse diferentemente. A continuación se presentan las clasificaciones más utilizadas encontradas en la literatura.

Tipo de Deuda	Definición
Deuda de Arquitectura	Problemas encontrados en la arquitectura de un proyecto. Por ej. violación de modularidad, la cual puede afectar los requerimientos de arquitectura (rendimiento, robustez, entre otros).
Deuda de Construcción	Temas relacionados a la construcción que hacen esta tarea más ardua, y que consumen más tiempo/procesamiento innecesariamente.
Deuda de Código	Problemas encontrados en el código fuente que pueden afectar negativamente la legibilidad del código haciéndolo más difícil de mantener.
Deuda de Defectos	Consiste en defectos conocidos, usualmente identificados durante las pruebas o el usuario, y debería ser solucionado, pero debido a otras prioridades y recursos limitados tienen que ser diferidas para otro momento.
Deuda de Diseño	Deuda que identifica el uso de prácticas que violan los principios de un buen diseño orientado a objetos.
Deuda de Documentación	Problemas encontrados en la documentación del proyecto de software.
Deuda de Infraestructura	Temas de infraestructura que, si están presentes en la organización, pueden retrasar o impedir algunas actividades de desarrollo.
Deuda de Gente	Temas con la gente que, si están presentes en la organización, pueden retrasar o impedir algunas actividades de desarrollo (ej. experiencia).
Deuda de Proceso	Procesos ineficientes o ineficaces, por ej. cuando un proceso que fue diseñado para realizar cierta tarea ya no es apropiado.
Deuda de Requerimientos	Concesiones realizadas con respecto a lo que el equipo de desarrollo necesita implementar o cómo implementarlo.
Deuda de Servicio	La sustitución de un servicio por otro puede estar dirigida por motivos del negocio o técnicos. Esta sustitución puede introducir deuda técnica relacionada a la selección, composición y operación del servicio.
Deuda de Automatización de Pruebas	Se define como el trabajo relacionado a la automatización de tests de funcionalidad desarrollada previamente para soportar integración continua o ciclos de desarrollo más cortos.
Deuda de Pruebas	Temas encontrados en actividades de prueba que pueden afectar la calidad de esas actividades de prueba (por ej. pruebas no ejecutadas o deficiencias conocidas en el suite de pruebas.)

Cuadro 2.2: Definiciones de tipos de deuda identificados. Adaptada de (Alves y cols., 2014).

2.6.1. Por artefacto o actividad

Cuando se habla de deuda técnica, de acuerdo a la metáfora original de Cunningham, se hace referencia a deficiencias en el código fuente del software. De la misma manera, otros artefactos y elementos del software pueden estar afectados al mismo concepto. Por ej. si la arquitectura del sistema no se alinea completamente con los atributos funcionales y no funcionales del software, suele hablarse de “deuda de arquitectura”.

Otras veces se habla de la deuda de la actividad o proceso que produce dicho artefacto. Siguiendo el ejemplo anterior, se denominaría “deuda de arquitectura” a la deuda técnica que genera la etapa del proceso de desarrollo que define la arquitectura de software.

Alves y cols. (2014) realizaron una revisión de la literatura e identificaron múltiples tipos de deuda técnica, enumerados en el Cuadro 2.2.

	Audaz	Prudente
Intencional	"No tenemos tiempo para hacer diseño"	"Debemos entregar ahora y lidiar con las consecuencias"
Inadvertida	"¿Qué es una arquitectura multicapa?"	"Ahora sabemos qué deberíamos haber hecho"

Cuadro 2.3: Clasificación según el nivel de intencionalidad y audacia. Adaptado de Fowler (2009).

2.6.2. Por intencionalidad y nivel de audacia

Una de las clasificaciones más conocidas es el cuadrante propuesto por Fowler (2009), y representado en el Cuadro 2.3. Fowler advierte distintos orígenes de deuda, encuadrándolas dependiendo del nivel de audacia con que se produzca (prudente o audaz) y la intencionalidad de la misma (inadvertida o intencional).[6] Algunos ejemplos de deuda que se ajusta a cada categoría:

- Intencional y audaz: Cuando un equipo decide no dedicar recursos para producir un buen diseño de la solución, debido a presiones del entorno.
- Intencional y prudente: Cuando un equipo decide que es mejor posponer la realización de ciertas actividades (caso típico es la documentación) para después de una entrega.
- Inadvertida y audaz: Cuando un equipo tiene poca experiencia y no se percata de que no está usando buenas prácticas.

[6] Prudente (*prudent*), audaz (*reckless*), inadvertida (*indavertent*), intencional (*deliverate*).

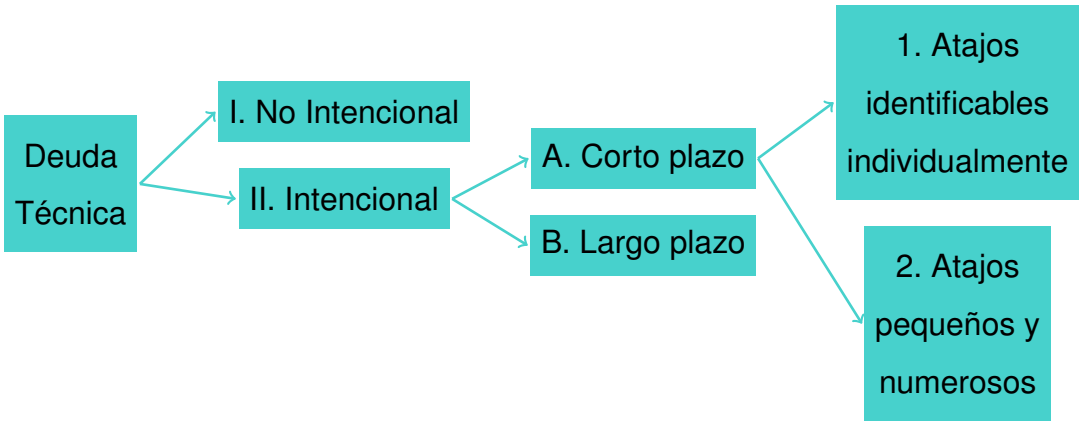


Figura 2.4: Clasificación según el nivel de intencionalidad y horizonte de tiempo. Extraída de McConnell (2007).

- Inadvertida y prudente: Cuando un equipo se da cuenta que debería haber adoptado un diseño distinto luego de haber entregado el software (se va aprendiendo sobre el dominio del problema mientras se desarrolla).

2.6.3. Por intencionalidad y horizonte de tiempo

Otra clasificación interesante es la planteada por McConnell (2007), que las encuadra en diferentes tipos de acuerdo a la Figura 2.4.

De manera similar al cuadrante de Fowler, McConnell en primera instancia las diferencia por su intencionalidad. La no intencional (tipo I) es la que se genera inadvertidamente por el equipo, como en el caso de un programador junior que simplemente no tiene suficiente experiencia como para producir con el nivel de calidad deseada.

La otra es la Intencional (tipo II) y se produce luego de tomar una decisión. Asimismo, la deuda Intencional la subdivide en deuda según el horizonte de tiempo en que se esperan obtener los beneficios: de Corto plazo (tipo II.A) y de Largo plazo (tipo II.B). La deuda de corto plazo es tomada reactiva y tácticamente, mientras que la de largo plazo se toma alineada a una decisión estratégica. Un ejemplo de deuda intencional a largo plazo es el no desarrollar soporte para un nuevo motor de bases de datos relacional porque el equipo de trabajo sabe que no será utilizado hasta dentro de tres años.

La deuda intencional a corto plazo, a su vez la diferencia entre los atajos

identificables individualmente (tipo II.A.1) y los atajos pequeños y numerosos (tipo II.A.2). En el primer caso, se trata de deudas que se toman específicamente para un fin puntual, por ej. desarrollar cierta funcionalidad en forma precaria con el objetivo de realizar la entrega tan pronto como sea posible, sabiendo que luego tendrán que hacerla más robusta (conceptualmente es como pedir un préstamo para comprar un vehículo). En contrapartida, las deudas generadas por atajos pequeños y numerosos representan muchos y reiterados pequeños o triviales atajos (por ej. nombres de variables poco significativos, documentación escasa, etc.) los cuales al acumularse finalmente representan una deuda considerable (conceptualmente es como comprar con una tarjeta de crédito, ya que es más fácil perder la noción de la deuda total acumulada).

Es claro, a partir de las clasificaciones de Fowler y McConnell que deben evitarse a toda costa las deudas generadas en forma inadvertida, entendiendo que el primer paso para solucionar un problema es reconocerlo. Asimismo, McConnell también advierte sobre la excesiva toma de deuda de atajos pequeños y numerosos, puesto que individualmente pueden ser triviales pero la acumulación de las mismas resulta nociva.

2.7. Otras metáforas

La metáfora de Deuda técnica descrita por Ward Cunningham, relacionando un préstamo con los problemas es la más popular y la más fácil de entender por personal no técnico, sin embargo no es la única ni fue la primera.

2.7.1. Leyes de Lehman

En 1974, Lehman reconoció cierto comportamiento que se daba durante el mantenimiento de sistemas, que luego fueron catalogados como las Leyes de la Evolución del Software de Lehman (Lehman, 1996). Citando las primeras dos leyes (año 1974) y la séptima (año 1996):

Cambio Continuo: Un programa tipo-E[7] que sea usado debe ser

[7]Los sistemas tipo-E son aquellos sistemas que resuelven problemas o implementan una

continuamente adaptado de lo contrario se torna progresivamente menos satisfactorio.

Complejidad Creciente: A medida que un programa evoluciona su complejidad se incrementa a menos que se realice trabajo para mantenerla o reducirla.

Calidad Declinante: Los programas tipo-E serán percibidos como de calidad declinante a menos que sean rigurosamente mantenidos y adaptados a un ambiente operativo cambiante.

Como es de esperarse, las leyes están dirigidas hacia una comunidad técnica y son difíciles de comunicar hacia el personal jerárquico, razón probable por la cual no son muy conocidas y no han trascendido más allá de ambientes académicos.

2.7.2. La pérdida de agua

Olivier Gaudin presenta una metáfora relacionando la calidad del software con una pérdida de agua (Gaudin, 2015):

> Cuando tienes una pérdida de agua en casa, ¿que haces primero? ¿Sellar la pérdida o limpiar el piso? La respuesta es muy simple e intuitiva: primero sellar la pérdida. ¿Por qué? Porque sabes que cualquier otra acción será inútil y que es solo cuestión de tiempo antes de que la misma cantidad de agua vuelva a caer sobre el piso.

Aquí, Gaudin se refiere a que los problemas de calidad se manifiestan como una pérdida de agua, es decir que la deuda técnica no se plantea como la causa de los problemas de calidad sino como un síntoma, sugiriendo que la solución no es resolverlos *ex-post*, sino que es necesario encontrar la causa raíz que tiende a la generación de deuda.

---

aplicación del *mundo real* (Lehman, 1996).

Capítulo 3

Visión estratégica de la deuda técnica

En los Capítulos anteriores se introdujeron los elementos fundamentales para comprender el fenómeno de la deuda técnica, explicando las causas de su presencia, donde puede encontrarse y generarse, junto con diversas clasificaciones de la misma.

Si bien la deuda técnica tiene un impacto negativo sobre la calidad del software (Figura 2.1), con una gestión adecuada puede ser utilizada para alcanzar importantes objetivos de negocio. De esta manera, la gestión de la deuda técnica constituye una estrategia puesto que es un medio para alcanzar un fin determinado. En particular, una *estrategia de gestión de TI*, y como tal es necesario que el CIO no vea la deuda técnica como una amenaza, sino como una oportunidad para mejorar la toma de decisiones y elevar las fortalezas de la gestión de TI.

En el presente Capítulo se abordará parte del segundo objetivo específico del presente trabajo, esto es, la gestión propiamente dicha de la deuda técnica, suministrando así a un CIO herramientas para poder accionar de acuerdo a los objetivos y estrategias de la organización. En este sentido, se describirá la relación entre la deuda técnica y los elementos que suelen conformar la visión estratégica de la gestión de TI en una empresa. Si esta visión es clara y bien comunicada, apalanca la organización hacia el cumplimento de las metas propuestas puesto que forman los cimientos para una correcta

gestión de TI, y en el contexto del presente trabajo, servirá como base para una correcta gestión de la deuda técnica.

La relación entre la deuda técnica y la gestión de TI se expondrá en un nivel ejecutivo, enfocándose en diferentes perspectivas de negocio para ilustrar –en opinión del autor– cual sería el *perfil de la deuda técnica* cuando está adecuadamente gestionada, o bien cual el *perfil de gestión* adecuado para alguna situación particular.

3.1. Perfil de la deuda técnica

El perfil de deuda indica cómo se espera que varíen *realmente* los niveles de deuda técnica si la gestión de la misma es la adecuada a una situación particular. Note el énfasis en la variación real de la deuda, la cual puede llegar a ser distinta que la ideal. Por ejemplo, al mantener un sistema legacy, lo *ideal* seria minimizar la deuda a niveles manejables, sin embargo este objetivo es muy difícil de lograr, puesto que lo que ocurre normalmente en *realidad* es que se mantendrá en niveles altos por el alto riesgo de introducir defectos o afectar la disponibilidad del servicio.

En los gráficos y cuadros presentados en este capítulo, se mostrará el perfil de deuda técnica con una curva color naranja, ilustrando la variación de la misma en distintas situaciones.

3.2. Perfil de gestión de la deuda técnica

Se identifican tres perfiles de gestión, esto es, tres formas de gestionar la deuda técnica que abarcan los casos más relevantes. Cada perfil consta de un número de actividades y están nombrados de acuerdo al objetivo de gestión que se intenta satisfacer. Los perfiles son: (1) Seguimiento, (2) Estabilización y (3) Reducción, y en las ilustraciones se identificarán con los íconos[1] que aparecen en la Figura 3.1.

[1] Íconos de `https://icons8.com`. Licencia *Creative Commons Attribution-NoDerivs 3.0 Unported* (`https://creativecommons.org/licenses/by-nd/3.0/legalcode`).

Figura 3.1: Perfiles de gestión de deuda técnica.

En líneas generales, cuando se aplica el perfil de Seguimiento, se realizan actividades para identificar la deuda técnica y observarla sin realizar intervenciones para disminuirla. En el perfil de Estabilización, además se efectúan estimaciones de la deuda técnica del sistema y se toman las medidas necesarias para que este nivel de deuda no se incremente más allá de un límite superior establecido por el CIO y el equipo de trabajo. Por último, en el perfil de Reducción, se realizan todas las actividades anteriores, con la diferencia de que se efectúan las modificaciones necesarias en el sistema para pagar la deuda, es decir reducir el Capital de la misma.

Como se ve en la Figura 3.1, los íconos muestran flechas con diferentes pendientes de nivel que –si la gestión de la deuda técnica es efectiva– indica si la cantidad de deuda técnica en el sistema resultará en un posible incremento, una estabilización (se mantiene en niveles más o menos constantes), o una reducción (la deuda técnica disminuye).

A medida que transcurre el tiempo y se van alcanzando los objetivos de gestión establecidos, el CIO puede ir cambiando el perfil de gestión para los diversos sistemas a efectos de racionalizar los recursos disponibles para este fin.

3.3. Participación de las TI en el negocio

Descripción: Desde el punto de vista de la visión estratégica de TI, es conveniente identificar a la organización según el nivel de participación que las tecnologías de información posean sobre el negocio. Si bien cada organización y mercado es único al momento de gestionar las TI, los negocios pueden clasificarse con respecto a su “dependencia” con la tecnología.

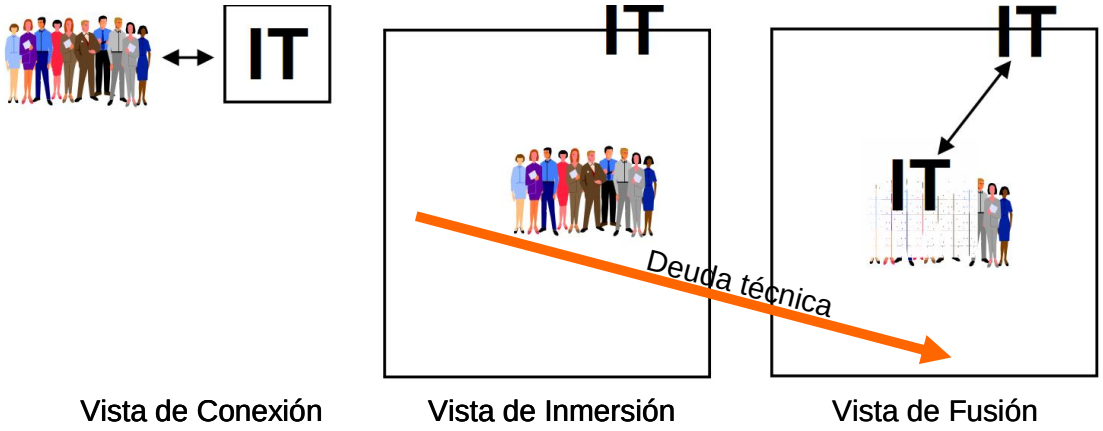


Figura 3.2: Niveles de participación de las TI en los negocios. Adaptada de El Sawy (2003).

En este sentido, se tomará la clasificación realizada por El Sawy (2003) como referencia, donde identifica el nivel de participación de las TI en tres categorías (Figura 3.2):

Conexión. Las TI son usadas por la gente como una herramienta para ayudarlos a realizar su trabajo. Es un artefacto separable que puede ser conectado a las acciones y comportamientos del trabajo de la gente. Puede ser dejada de lado si es necesario, y se enfoca en temas intra-organizacionales.

Inmersión. Las TI están inmersas como parte del entorno de trabajo y no puede ser separada del trabajo y las propiedades sistémicas de las relaciones inter-organizacionales. Los procesos de trabajo y la TI están entretejidas, son altamente interdependientes, e íntimamente se influencian mutuamente.

Fusión. Las TI están fusionadas dentro del ambiente de negocios de tal manera que son indistinguibles a nuestra percepción y forman una tela unificada. Cambian los límites entre el trabajo y la vida personal, y funden información pública y personal. El trabajo *per se* es el trabajo habilitado por las TI.

Un ejemplo de un negocio en que las TI participan en un nivel de Conexión es un taller mecánico, donde las TI sirven como un soporte al trabajo

central, esto es, si fallaran los sistemas de información de la empresa, igualmente podría seguir trabajando ya que el negocio (reparación de automóviles) puede proceder separadamente a las TI.

Un ejemplo de empresa con una participación de nivel Inmersión podría ser el sistema de expedientes del Poder Judicial, donde al mismo tiempo en que la documentación se reúne en un expediente impreso, también debe incorporarse en formato digital por parte de los letrados que actúan en las causas judiciales: aquí se ve la dependencia del negocio con partes fuera de la organización (letrados).

Por último, como ejemplo de negocio con una participación a nivel de Fusión puede citarse la banca electrónica, donde se puede ver que se funde la naturaleza del negocio (gestión de depósitos, extracciones, transferencias, compras, pago de servicios) con información personal de la gente (DNI, CBUs, tarjetas de crédito, etc.), y la manera de realizar negocios depende inexorablemente del manejo y explotación de la información, esto significa que el negocio de banca electrónica no existe sin la TI. Otros ejemplos de Fusión son los negocios netamente digitales (Netflix, Spotify, App Stores, etc.) donde la TI sirve como único medio y mercado para brindar el servicio.

Impacto sobre la deuda técnica: El nivel de participación de la TI en el negocio define el nivel de "dependencia" del negocio con respecto a las TI. Esta dependencia es relativa a nivel de Conexión, más importante a nivel de Inmersión y absoluta a nivel de Fusión. Esto significa que una buena gestión de TI es más crítica en un negocio en condiciones de Fusión que en Inmersión, y esta última más importante que en condiciones de Conexión.

En condiciones de Fusión, el alineamiento de TI con respecto a las estratégicas del negocio cobran mayor importancia, puesto que el alineamiento no proviene gratuitamente con las condiciones de fusión del negocio, por el contrario, debido a su fuerte dependencia con la TI, los resultados negativos producto de la falta de alineamiento crecen con ésta (Freijedo, 2015). Ambas, el negocio y la gestión de TI, deben trabajar en consonancia tanto para soportar las operaciones diarias del negocio como para poder adaptarse rápi-

damente a nuevas oportunidades y amenazas. Como resultado, la gestión de TI tiene un rol preponderante en condiciones de Fusión.

Debido a esto último, la deuda técnica en el contexto de Fusión debe observarse muy de cerca a fin de mantener una calidad de servicio adecuada, así como sostener y mejorar los niveles de disponibilidad del mismo, puesto que tanto la calidad y disponibilidad son indispensables para la existencia del negocio. En este contexto, la gestión de la deuda técnica debe ser tan exigente como se requiera, tratando de mantenerla en un mínimo adecuado para garantizar la operación diaria.

Asimismo, en condiciones de Inmersión, si bien el negocio requiere las TI para su normal funcionamiento debido a que requiere interacción con partes externas al negocio, no es imperativo para su existencia. En el ejemplo del sistema del Poder Judicial, si el sistema de expedientes queda fuera de servicio, aún puede continuarse la operación del negocio mediante el formato de expediente impreso. Por ello, la deuda técnica aquí debe mantenerse a niveles tolerables en la medida que el negocio lo requiera.

Por último, en el contexto de Conexión la gestión de la deuda técnica podría o no existir. Esto es debido a que el negocio utiliza la TI como herramienta de soporte pero no representa la parte central del mismo.

Como resultado, la Figura 3.2 muestra una linea descendiente en color naranja desde un nivel de Inmersión hasta Fusión (sin alcanzar el nivel de Conexión), indicando la cantidad de deuda técnica que debería tener el negocio en relación a esta clasificación.

3.4. Perfil tecnológico

Descripción: El perfil tecnológico de la organización determina si la misma estará orientada a la exploración o a la explotación. Desde este punto de vista, los procesos de negocio, de toma de decisiones y la forma de determinar el éxito de un emprendimiento variarán. De acuerdo a Nicolau-Juliá, Expósito-Langa, y Tomás-Miquel (2015), “Estos 2 conceptos requieren diferentes estructuras, procesos, estrategias, capacidades y culturas, y pueden tener

diferentes impactos en el desempeño de una organización. Además, tanto la exploración como la explotación suponen diferentes tipos de aprendizaje en la empresa. Así, mientras que la esencia de la exploración recae en la experimentación con nuevas alternativas que tienen unos retornos inciertos, distantes y a menudo negativos, la de la explotación recae en el perfeccionamiento, la ampliación de competencias existentes y las tecnologías con retornos positivos, próximos y predecibles."

De esta manera, el perfil tecnológico de la organización determinará el tipo de emprendimientos en los que se puede embarcar. Si se desea ser una organización innovadora, deben asignarse recursos para realizar las innovaciones, tener el personal adecuado, una aversión al riesgo más elevada, junto con el establecimiento de objetivos que sean más laxos en cuanto al retorno de la inversión. Por el contrario, una organización con perfil de explotación priorizará mejorar su línea de negocios a través de pasos evolutivos más pequeños pero menos arriesgados, donde el éxito esté basado en los beneficios obtenidos a corto plazo.

En definitiva, lo que se intenta determinar es el perfil de la organización con respecto a la innovación. En este sentido, debería poder ubicarse la misma de acuerdo a la curva de Rogers de difusión y adopción de innovaciones (Figura 3.3).

Impacto sobre la deuda técnica: En el contexto de la gestión de la deuda técnica, se puede esperar que las organizaciones orientadas a la exploración tengan una cultura de experimentación que redunde en una alta cantidad de deuda técnica, ya que mientras se experimenta es probable que lleguen a soluciones con variable calidad, por lo tanto, alineado a esto es esperable que sean más permisivos al momento de gestionarla.

Por otro lado, las organizaciones orientadas a la explotación tienen objetivos más predecibles y esperan retornos de inversión concretos sobre iteraciones evolutivas del negocio, es por ello que mantengan un control más riguroso en cuanto a la cantidad de deuda técnica generada, ya que la misma puede llegar a impedir futuras mejoras.

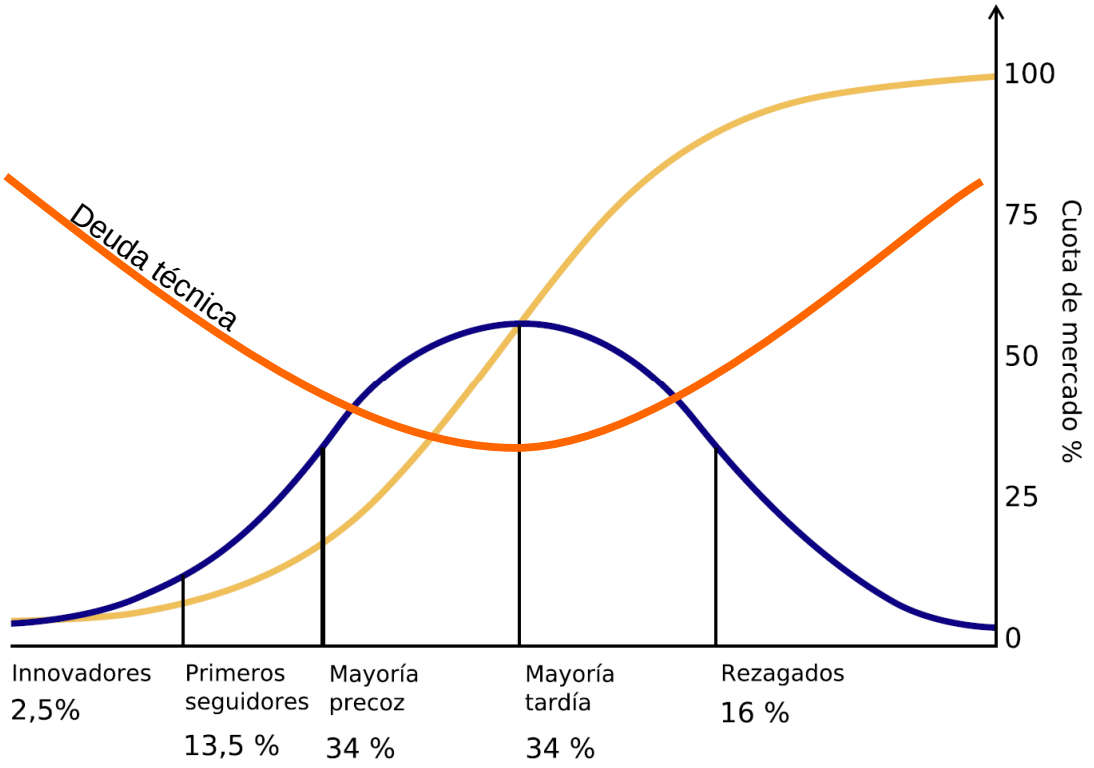


Figura 3.3: Curva de difusión y adopción de innovaciones. Extraída de Wikipedia (2016b).

A su vez, siguiendo la curva de difusión y adopción de innovaciones (Figura 3.3), en un extremo están los Innovadores (afines a la exploración) y en el otro extremo los Rezagados (afines a la explotación). La deuda técnica generada puede ser elevada en ambos casos en lo relativo al desarrollo de sistemas. La razón es que los Innovadores pueden estar utilizando tecnologías que no están bien consolidadas o probadas (ej. productos en etapa Beta, prototipos, etc.) por lo cual la falta de experiencia normalmente conlleva a desarrollar soluciones con un uso no adecuado de las mismas. Por otro lado, los Rezagados utilizan nueva tecnología sólo cuando no tienen otro remedio, por lo que retrasan la adopción de nuevas herramientas y métodos de trabajo hasta el punto en que no les es posible progresar, razón por la cual suelen tener problemas de integración de nuevas tecnologías con sus actuales sistemas, que son eficientes pero rígidos.

3.5. Estructura organizativa

La estructura organizativa de la empresa define la forma en que se toman las decisiones, y la misma tiene un profundo impacto en la forma en que se comunica, esto es, cómo fluye la información y cómo el trabajo es realizado. El impacto de la estructura organizativa sobre los sistemas de información es tan importante, que dio lugar a lo que se conoce como Ley de Conway (Conway, 1968): “organizaciones que diseñan sistemas (...) están restringidas a producir diseños que son copias de las estructuras de comunicación de esas organizaciones”.

3.5.1. Centralización vs. Descentralización

Descripción: Con referencia a la forma en que se toman las decisiones, las organizaciones pueden adoptar (en su estructura general, en su gobierno de TI o gestión de TI) un esquema que sea centralizado o descentralizado.

El esquema centralizado define un único organismo de control que toma todas las decisiones, y se utiliza cuando el foco está puesto en la reducción de costos y control (estandarización). El control centralizado ayuda a conseguir economías de escala, y facilita la integración de las TI. Por otro lado, pueden causar rigidez en cuanto a la adaptación de aspectos locales de las unidades de negocio, con lo que las mismas pierden agilidad ante cambios en el contexto de mercado.

En el enfoque de la descentralización se definen múltiples organismos que pueden tomar decisiones a la medida de sus propias necesidades, por lo tanto las unidades de negocios se vuelven más ágiles a las necesidades de oferta y demanda, cambios en el contexto del negocio, así como para seleccionar los métodos y herramientas más adecuadas para su óptima operación. Como desventaja, la descentralización dificulta la homogeneidad de soluciones a problemas comunes, lo que agrega complejidad y costos a los procesos de negocio, arquitectura e infraestructura de TI.

Por supuesto, estas dos formas de tomar decisiones son los extremos del espectro. Weill y Ross (2005) detallan distintos enfoques con respecto al

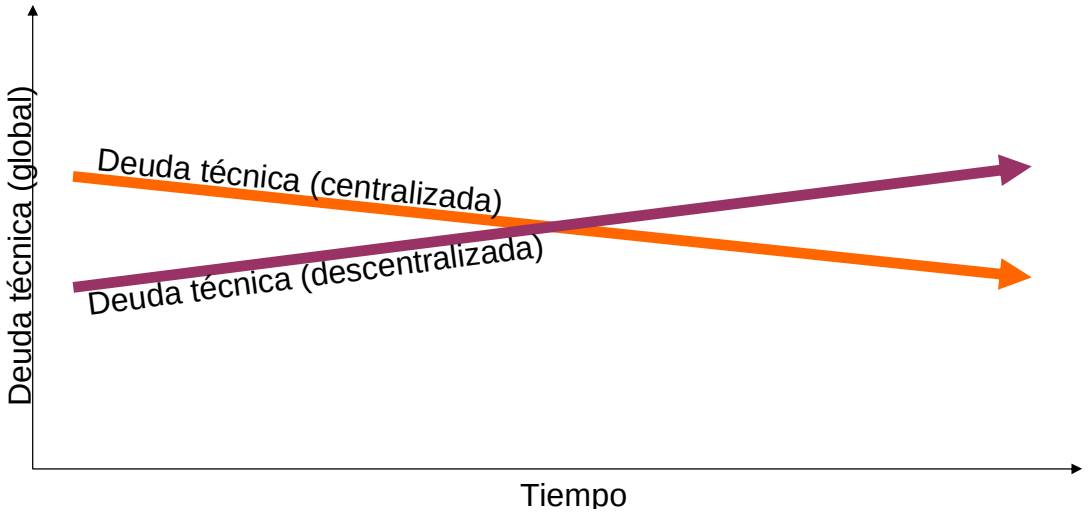


Figura 3.4: Variación de la deuda técnica global a largo plazo, en un modelo centralizado y descentralizado.

grado de centralización/descentralización de la toma de decisiones. En su trabajo se destaca que las decisiones de TI se pueden tomar en distintos niveles: a nivel corporativo, a nivel de unidad de negocio, a nivel funcional, o alguna combinación de los tres.

Impacto sobre la deuda técnica: La forma que adopta la estructura organizativa (general, de gobierno de TI y del área funcional de gestión de TI) es consecuencia de las circunstancias y el contexto de negocios presente al momento de definirla. Si este contexto cambia, y debe cambiar la estructura organizativa, se tendrá que hacer frente a los desafíos que conlleva este cambio, entre ellos, las consecuencias de pasar de un esquema centralizado hacia uno descentralizado o viceversa.

Centralizada a descentralizada Si cambia de un esquema centralizado a uno descentralizado, los activos y prácticas de TI probablemente sean lo suficientemente homogéneos y estandarizados pero no sean lo suficientemente adecuados para adaptarse bien en algunas unidades de negocio o mercados locales. Posiblemente se necesite mejorar el producto final y elevar la satisfacción de los clientes. A un nivel macro, la deuda técnica inicialmente estará acotada a las cantidades gestionadas en forma central.

La gestión de la deuda técnica podría estar descentralizada, en el caso de que las necesidades diverjan significativamente de una unidad de negocio a otra. La decisión de los niveles de deuda técnica tolerables o admisibles en una unidad de negocios quedará a cargo de un responsable de TI local, así como la estrategia y acciones para disminuirla.

Debe tenerse en cuenta que a medida que pase el tiempo y cada unidad de negocio evolucione en forma independiente, la deuda técnica será mejor gestionada a nivel local pero la divergencia en los procesos de negocios, sistemas o infraestructura impactará negativamente en la deuda técnica a nivel global (Figura 3.4).

Descentralizada a centralizada En el caso de que se cambie de un esquema descentralizado a uno centralizado, los activos de TI probablemente sean heterogéneos y de compleja integración, cada producto o sistema estará basado en una tecnología e infraestructura particular, por lo que se necesita un perfil de expertise único para mantenerlos. Desde una visión macro, la deuda técnica seguramente será elevada en su totalidad, aunque todas o algunas unidades de negocio la hayan gestionado de acuerdo a sus necesidades. Suponiendo que lo que se desee realizar es mejorar los costos operativos por medio de la estandarización e integración de soluciones, deberán realizarse interfaces entre sistemas, refactoreos para estandarizar resultados, así como migraciones en caso de que se decida terminar el ciclo de vida de algún sistema y sea reemplazado por otro.

Para gestionar la deuda técnica, es conveniente relevar, investigar y acordar los criterios a utilizar para cada unidad de negocio. La gestión de la deuda técnica será impuesta por una estructura a cargo del CIO. La estructura que comúnmente cumple con las condiciones de sugerir e imponer requerimientos de calidad internos en los sistemas de información, buenas prácticas, así como su auditoría y emisión de excepciones (si correspondiere) es la de Arquitectura, que depende del CIO.

Es de esperarse que siendo la gestión centralizada, la deuda técnica de la organización vaya disminuyendo a lo largo del tiempo, debido a la eficiencia

lograda en lo relativo a las economías de escala (ej. servicios con tecnologías y servicios estandarizados) y la utilización eficiente de los activos (ej. reuso de componentes). La Figura 3.4 muestra este comportamiento.

Enfoque mixto Naturalmente, podría tomarse un enfoque mixto sobre la gestión de la deuda técnica. Un ejemplo puede ser que un comité de gobierno de TI (constituido por el CIO y gerentes de cada unidad de negocio) puedan definir y transparentar la importancia de la gestión, y en base a eso definir políticas y principios de TI que promuevan y establezcan criterios para gestión de la deuda. Asimismo, el CIO (centralizada) o responsables de TI de las unidades de negocio (descentralizada) puedan aplicarlos, quedando a disposición para que la casa central pueda auditar y seguir la evolución de la misma en cada unidad.

En forma centralizada, también puede definirse una suerte de "línea base" para la gestión de la deuda técnica, los tipos de errores o advertencias sobre problemas que se deben evitar, como por ej. problemas de testeabilidad, confiabilidad, mantenibilidad, seguridad, etc. Además, cada unidad de negocio puede solicitar excepciones, normalmente justificables presentando casos de negocio habilitados por oportunidades o amenazas.

Otro aspecto que facilita la centralización es que las herramientas de apoyo que calculan la deuda técnica pueden configurarse con distintos "perfiles", cada uno de los cuales pueden conformarse para detectar o ignorar cierto tipo de advertencias. Esto permite una administración del portafolio de sistemas, es decir, sistemas que dan soporte a un negocio en particular serán agrupados en el mismo perfil, lo que da homogeneidad y consistencia a la gestión. Como ejemplo puede citarse la instancia pública del producto Sonarqube,[2] que alberga el análisis de más de 800 proyectos de código abierto, cada uno usando diferentes tecnologías y desarrollados en forma distribuida por programadores de todo el mundo.

[2]Se puede acceder aquí: `https://sonarqube.com/`

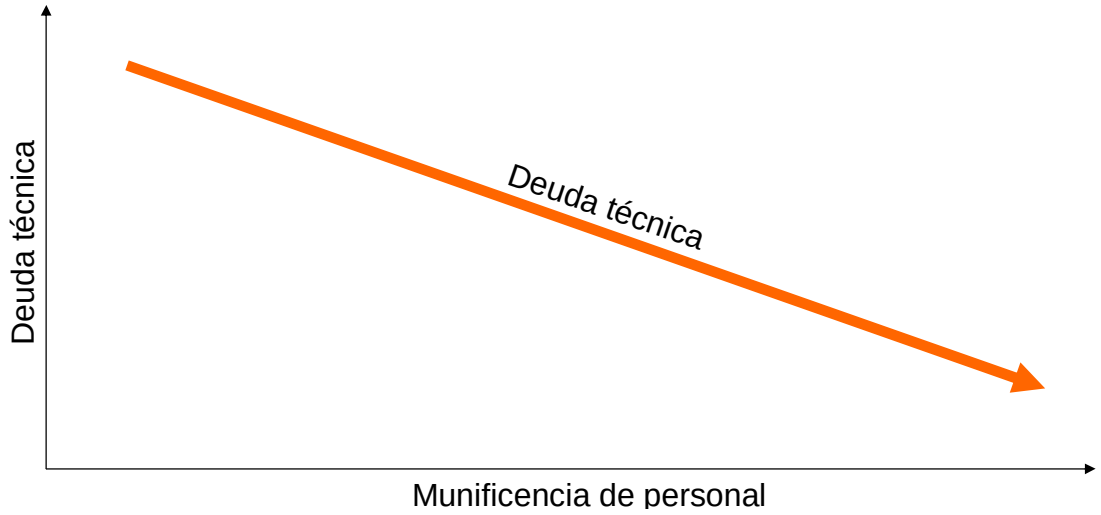


Figura 3.5: Munificencia de personal vs. Deuda técnica.

3.5.2. Recursos Humanos

Descripción: Otro aspecto es la capacidad ociosa de la organización[3] o munificencia de recursos, esto es, "la disponibilidad de recursos críticos para operar dentro de un ambiente" (Woodard y cols., 2012). En un contexto de baja munificencia (escasez de recursos), la gente tiende a trabajar más cantidad de horas bajo la presión de responder a las metas fijadas, mientras que en el contexto de una alta munificencia (abundancia de recursos) la gente tiende a trabajar más distendida. En este sentido siempre es bueno llegar a un equilibrio: si bien tener capacidad ociosa es costoso, también lo es no poder responder a tiempo ante oportunidades o amenazas de negocios, o bien tener una alta tasa de rotación de personal (especialmente si en el mercado laboral existe una alta demanda de profesionales de TI).

Impacto sobre la deuda técnica: La escasez de personal ocasiona que los equipos de trabajo tengan que trabajar con niveles de estrés elevados, esta es una de las causas de la presencia de deuda técnica examinadas en la Sección 2.2. Si la gestión de la deuda técnica es adecuada, es de esperarse que ante una alta munificencia se obtengan menores niveles de deuda técnica, como se muestra en la Figura 3.5.

[3] En inglés: *organizational slack*.

La justificación de lo anterior, como indica Woodard y cols. (2012): “bajo escasez de recursos, inversiones para reducir la deuda técnica a través de esfuerzos de desarrollo adicionales es improbable que sean factibles”, lo que intuitivamente significa que en el contexto de escasez, el personal priorizará las tareas que tengan un impacto externo sobre las que tengan impacto solamente interno. Como resultado, solo con una cierta capacidad ociosa de recursos pueden hacerse esfuerzos de desarrollo para reducir la deuda.

3.6. Arquitectura empresarial

El estándar ISO 44010:2011 (ISO, 2011) define la arquitectura como “los conceptos fundamentales o propiedades de un sistema en su entorno incorporados en sus elementos, las relaciones, y en los principios de su diseño y evolución”. Como se ilustró en el Cuadro 2.1 en la página 15, tanto la funcionalidad como la arquitectura impactan positivamente en el valor que un sistema aporta al negocio.

Mientras que la funcionalidad es lo que se expone a los usuarios o clientes, la arquitectura constituye los cimientos sobre los que está apoyada esa funcionalidad. En otras palabras, mientras que un buen sistema expone funcionalidades de utilidad para sus usuarios, una buena arquitectura aporta nuevas capacidades al negocio, lo que le otorga mayor maniobrabilidad al momento de ejecutar las estrategias planificadas.

Todos los sistemas, sin excepción, tienen y siguen una arquitectura determinada, ya sea creada conscientemente (arquitectura incidental) o bien sin haber sido contemplada (arquitectura accidental), lo que resulta en sistemas con diferentes grados de calidad (Booch, 2006). Es de esperarse que las organizaciones cuyo proceso de desarrollo posean una etapa de definición de arquitectura (o bien un sector dentro de la Gcia. de Sistemas) produzcan arquitecturas y sistemas que se adecuen mejor a las necesidades de la organización, de las que no poseen este proceso.

Soportando la definición de arquitecturas, la práctica de *arquitectura empresarial* es una visión de gestión de TI que permite abordar el desarrollo

de arquitecturas que habiliten las capacidades necesarias para cumplir con los objetivos de la organización. Para ello, normalmente se siguen las recomendaciones aportadas por algún *framework* de arquitecturas empresariales, por ej. TOGAF (The Open Group, 2011) o Zachman (Zachman, 1996).

3.6.1. Modelo operativo

Descripción: Mientras que una estrategia de TI sirve como guía para diseñar las actividades que necesitan cambiar, el modelo operativo es una guía que establece lo que *no va a cambiar*, un compromiso sobre cómo la organización hace negocios. De acuerdo a Ross y cols. (2006), "un modelo operativo descríbe cómo una compañía desea prosperar y crecer. Proveyendo una vista más estable y concreta de la compañía que la estrategia, el modelo operativo guía el diseño de los cimientos para la ejecución". Robertson (2012) sugiere preguntarse ¿Cuales son las *transacciones sagradas*?, ¿Cuáles son las actividades que hicimos ayer, que vamos a hacer hoy, y que vamos a realizar mañana que son el centro del negocio, que no van a cambiar?

La adopción de un modelo operativo es de vital importancia para la organización, ya que por medio de este se compromete a realizar sus negocios de cierta manera, y afecta a los procesos de negocio, cómo se maneja la información, las aplicaciones y la infraestructura. Acordar y decidirse por un modelo operativo es una tarea difícil, sin embargo el esfuerzo bien vale la pena: Ross y cols. (2006) encontraron que las que lo lograron "reportaron un incremento del 17 % en efectividad estratégica que otras compañías (...). Estas compañías también informaron una mayor eficiencia operacional (31 %), cercanía con el cliente (33 %), liderazgo de producto (34 %), y agilidad estratégica (29 %) que las compañías que no han desarrollado un cimiento para la ejecución".

Asimismo, Ross y cols. (2006) identifican dos dimensiones que impactan en la conformación del modelo operativo: una es el nivel de estandarización y la otra el nivel de integración.

La *estandarización* de procesos de negocios define cómo un proceso será ejecutado sin importar quien lo realiza o donde. Niveles altos de estan-

darización tienen como ventaja que los procesos se realizan en forma eficiente, consistente y predeciblemente en toda la organización. Como desventaja, estos procesos limitan la innovación a nivel local, ya que requiere que los sistemas se comporten de manera estándar en todas las unidades de negocios y situaciones.

La *integración* vincula el esfuerzo de las unidades organizativas a través de datos compartidos. Como ventaja, la integración aporta más eficiencia, coordinación, transparencia y agilidad. Además ofrece mejor información para la toma de decisiones, permitiendo a una parte del negocio actualizar o compartir datos. La mayor dificultad es desarrollar definiciones estándares para los datos de manera que al ser compartidos tengan una única interpretación posible, y lograr estas definiciones requiere trabajo.

Estas dos dimensiones dan lugar a cuatro tipos de modelos operativos: (1) Diversificación (bajos niveles de estandarización y de integración), (2) Coordinación (bajos niveles de estandarización y altos de integración), (3) Replicación (altos niveles de estandarización, y bajos de integración) y (4) Unificación (altos niveles tanto de estandarización como de integración).

Impacto sobre la deuda técnica: La arquitectura juega un rol clave en la gestión de la deuda técnica. En su estudio, Ernst y cols. (2015) encontraron que las (malas) decisiones de arquitectura, frecuentemente tomadas demasiado temprano en el ciclo de vida del sistema, son la fuente más importante de deuda técnica, como muestra la Figura 2.2 en la página 24.

El modelo operativo, que representa la estructura de las operaciones diarias de la organización, esto es, la forma en que la organización decidió crecer y operar, impacta fuertemente en las decisiones de arquitectura. Así, la definición del modelo operativo sirve de guía para definir la arquitectura empresarial, esto es, los procesos de negocio, la arquitectura de los sistemas de información, la estructura y semántica de la información misma, y la arquitectura tecnológica de infraestructura.

Una vez que la arquitectura esta definida y operativa, se espera que esté alineada y arraigada al modelo operativo y por lo tanto, cuando este últi-

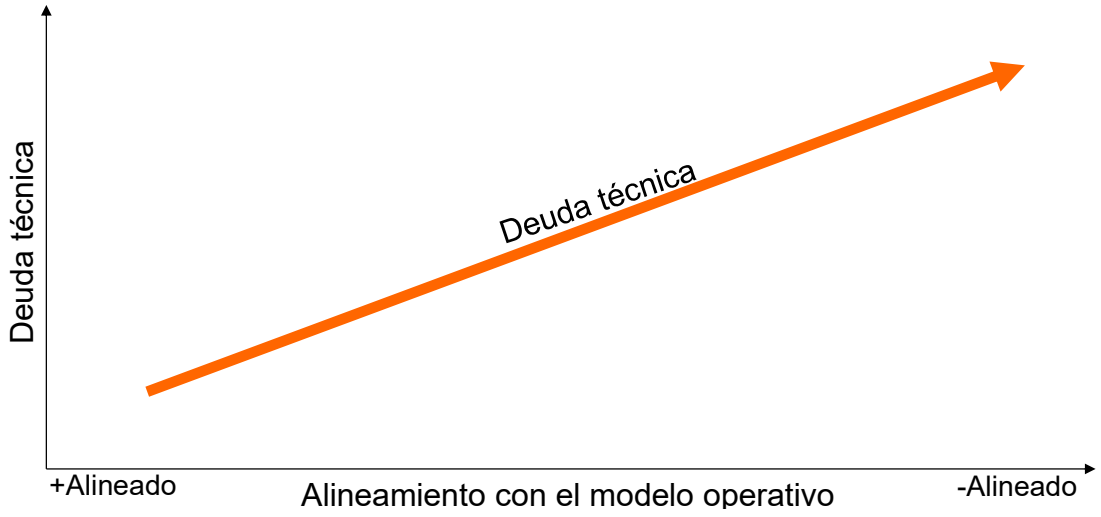


Figura 3.6: Alineamiento al modelo operativo vs. Deuda técnica.

mo ya no satisfaga las necesidades del negocio, comenzará a sentirse dificultad para adaptar los nuevos requerimientos de los sistemas a la arquitectura, generando deuda técnica en el proceso (Figura 3.6).

Por ejemplo, si se desea pasar de un modelo operativo con bajos niveles de integración a uno más integrado, se necesitará modificar los procesos de negocio, arquitectura de sistemas, las estructuras de los datos para operar de forma centralizada. Posiblemente procesos que antes se hacían en forma *batch*, ahora deben realizarse en tiempo real. En el caso inverso, si se pasa de un modelo con alto nivel de integración a uno menor, deben desagregarse los procesos de negocio, arquitectura de sistemas y de datos para aprovechar la agilidad requerida por la organización.

3.6.2. Madurez de la arquitectura empresarial

Descripción: A medida que la organización aprende y adquiere distintas capacidades de gestión de TI en relación a sus procesos de negocio, las arquitecturas irán cambiando de forma para adaptarse a nuevos desafíos, esto es, las arquitecturas empresariales adoptan ciertas características y priorizan distintos aspectos de acuerdo a la etapa de madurez en que se encuentren. Ross y cols. (2006) identifican cuatro niveles de madurez de las arquitecturas, enumeradas a continuación en orden creciente de madurez: (1) Silos de ne-

gocio, (2) Tecnología Estandarizada, (3) Centro Optimizado, y (4) Modularidad de Negocio.

En la etapa 1 –Silos de negocio– el rol de las TI es reducir costos por medio de la automatización de procesos de negocios. En esta etapa, los gerentes de los procesos de negocio especifican al responsable de las TI sus requerimientos. La función de TI es 100 % reactiva a estos requerimientos, con lo que se satisfacen las necesidades de negocio en forma local, lo que promueve la innovación y mejora la competitividad localmente. Como desventaja, luego de cierto tiempo en este modo de operación, se habrán construido sistemas de compleja integración.

En la etapa 2 –Tecnología Estandarizada– se sigue priorizando la automatización de procesos de negocio locales, pero ahora se desplaza el foco hacia el costo y confiabilidad de los sistemas de la organización por medio de la estandarización de la base tecnológica, lo que incluye: infraestructura, conectividad, hardware, software de base (sistemas operativos, bases de datos, servidores de aplicaciones, etc.). El CIO provee servicios compartidos sobre los cuales los negocios locales deben negociar la mejor solución posible dentro de ese marco. Inclusive, la cantidad de productos ofreciendo funciones similares puede llegar a reducirse, impactando tanto en la reducción de costos como en eficiencia, asimismo la reducción de base tecnológica no representa una pérdida en la competitividad ya que esta infraestructura no aporta grandes factores de diferenciación, como sugiere Carr (2003).

En la etapa 3 –Centro Optimizado– se desplaza el foco de los datos y aplicaciones desde una visión local hacia una visión a nivel de empresa. Las inversiones se priorizan para consolidar y eliminar la redundancia en los datos gestionados por las aplicaciones para que sean accesibles por todos los procesos de negocio de la organización, con lo que las aplicaciones también deben adaptarse.

Por último, en la etapa 4 –Modularidad de negocio– la empresa apuesta a la agilidad en sus negocios usando como estrategia de TI la construcción de componentes reusables, lo que requiere un mayor esfuerzo de modularización de los mismos. El desafío es extender los sistemas en el nivel de madurez

anterior en forma modular, y satisfacer las necesidades locales componiendo estos módulos. Entre los estilos de arquitectura que habilitan esta modularización están SOA (*Service Oriented Architecture*) y MSA (*Micro-Service Architecture*).

Impacto sobre la deuda técnica: En líneas generales, en cada una de las etapas de madurez de la arquitectura existen elementos que requieren cambios que tienen un profundo impacto. Por ej. al elevar el nivel de madurez de la etapa 1 a la 2, se requiere transformar los servicios de infraestructura de TI, así como los procesos que soportan su evolución. Es aconsejable priorizar la gestión de la deuda técnica sobre estos elementos ya que representan los cambios más significativos y son la base para conseguir el objetivo deseado (en el caso de migrar a la etapa 2, el objetivo es mejorar la eficiencia y costos de la función de TI).

La Figura 3.7 ilustra en columnas las cuatro etapas de madurez y para cada una de ellas, los niveles de deuda técnica esperados para lograr la disponibilidad de datos compartidos, infraestructura compartida, sistemas empresariales (disponibles para toda la organización) y aplicaciones locales (para las unidades de negocios). Se puede apreciar la evolución de las prioridades de gestión para bajar los niveles de deuda en los diversos elementos de la arquitectura para lograr cada uno de los niveles de madurez. Adicionalmente, la Figura muestra un perfil de gestión de deuda técnica recomendado para iniciar la transición hacia un nuevo nivel de madurez. Por supuesto, este perfil puede ir cambiando acorde a las necesidades.

En la etapa de madurez 1 –Silos de negocio– los sistemas se desarrollan (o adquieren) de acuerdo a los requerimientos locales de cada una de las unidades de negocio, creando aplicaciones aisladas (los “silos”) que se concibieron sin una visión integrada de los datos y procesos de negocio, por lo cual la tarea de integrarlos será muy difícil, lo que genera deuda técnica en el proceso.

Ya que lo que se priorizan son las necesidades locales de cada unidad de negocio, debe priorizarse la gestión de la deuda técnica de los sistemas

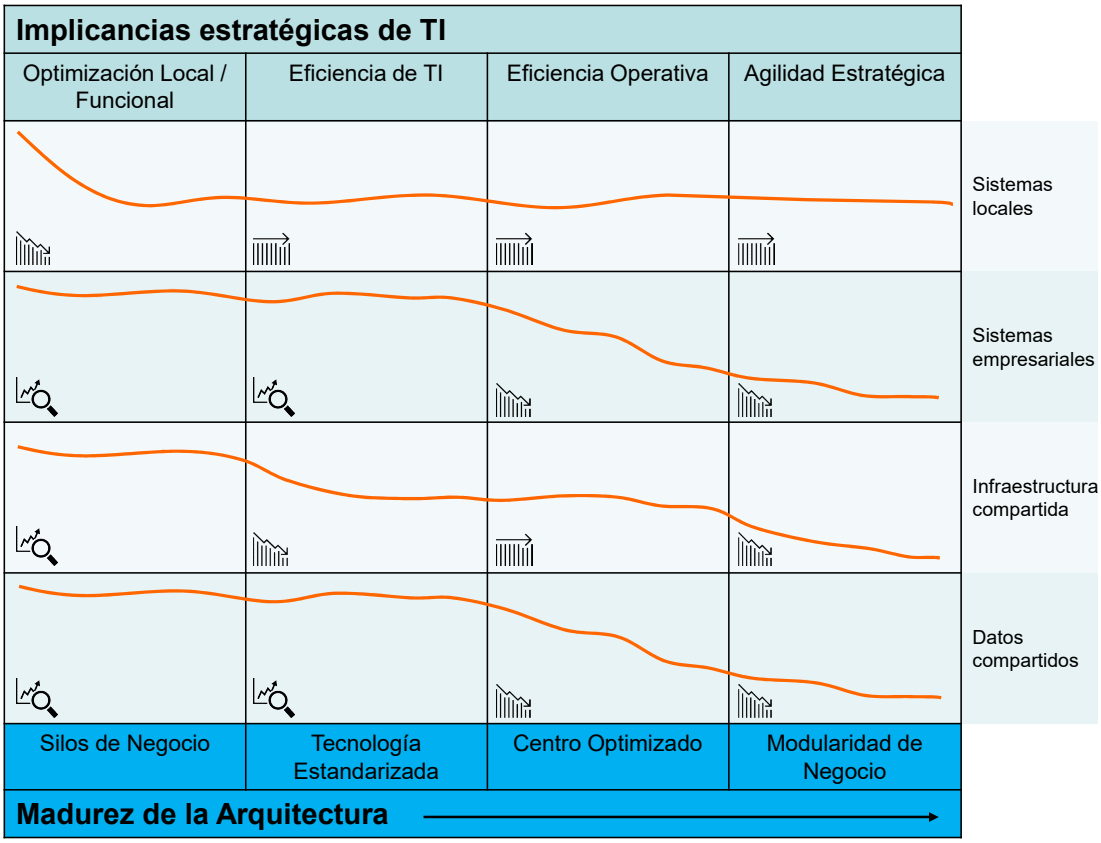


Figura 3.7: Deuda técnica según las etapas de madurez de las arquitecturas. Adaptada de Ross y cols. (2006).

locales a un nivel riguroso tratando de mantenerla en niveles acotados o bien tratando de reducirla para no afectar las unidades de negocio. Asimismo, es conveniente observar de cerca la deuda técnica sobre la infraestructura, sistemas y datos compartidos, con el propósito de conocer las principales fuentes cuando se requiera integrar o estandarizar algún proceso particular, como se muestra en la Figura 3.7.

En la etapa 2 –Tecnología Estandarizada– se intentan reducir costos comprometiéndose a estandarizar la tecnología. Una base tecnológica muy diversificada necesita distintas habilidades en el equipo de trabajo para gestionarse correctamente, lo que dificulta las pruebas de integración, la puesta en producción y la gestión de configuración de los sistemas. Las actividades que dificultan estas tareas representan deuda técnica a nivel de infraestructura, lo que típicamente se gestiona con una división de *DevOps*[4] y la automatización

[4] *DevOps* es una combinación de las palabras *development* y *operations*, y es una práctica

de tareas. La gestión de la deuda técnica sobre los aspectos de infraestructura debe ser rigurosa, no solo para mantener, sino para disminuir los niveles de heterogeneidad de la infraestructura para conseguir el objetivo deseado.

En la etapa 3 –Centro Optimizado– se pone el foco en compartir la misma información en múltiples unidades de negocio, con lo que probablemente deba consensuarse y cambiar la estructura y semántica de la misma para que se pueda usar de forma efectiva. Estos cambios en el alcance de la información obligará a modificar las aplicaciones que la manipula para presentarla en forma consistente para cualquier parte de la organización. Ya que estas modificaciones tienen un impacto global, la gestión de la deuda técnica sobre estos elementos debe realizarse en forma rigurosa para que vaya reduciéndose a medida que se van realizando modificaciones a efectos de que esta información global esté normalizada e integrada para garantizar su disponibilidad en forma oportuna. Asimismo, las aplicaciones locales que deban modificarse deben ser vigiladas para que su deuda no aumente en relación a sus niveles anteriores.

Finalmente, la etapa 4 –Modularidad de negocio– involucra un esfuerzo particular para modularizar los sistemas en funcionalidades independientes y reusables para construir soluciones componiendo estos módulos. Estas aplicaciones se comportan en forma consistente con el resto de la empresa, mientras que permiten adaptaciones con objetivos locales. Estos módulos no solo son la base técnica para componer rápidamente aplicaciones que satisfagan los requerimientos, sino que representan un elemento diferenciador aportando a la competitividad de la organización. Puesto que las transformaciones son sobre los datos compartidos y los sistemas impactando a nivel global, se requiere que la deuda técnica siga gestionándose rigurosamente para mejorar las oportunidades de modularización. Asimismo, la infraestructura sobre la que ejecutan los componentes cobra una mayor relevancia que en el nivel de madurez anterior ya que deben mantenerse y mejorarse los niveles de disponibilidad al ser necesarias para la ejecución de los módulos de negocio, es por

donde un equipo de trabajo hace de puente colaborando entre el equipo de desarrollo, el de QA y el de operaciones (Wikipedia, 2016a), normalmente asistido por herramientas de automatización.

eso que la gestión de la deuda técnica deba intensificarse.

3.7. Priorización de proyectos

Como sucede en la mayoría de las empresas, existen más proyectos e iniciativas estratégicas que recursos disponibles para llevarlas a cabo, lo que da lugar a una priorización de las mismas para asignar los recursos a la selección de proyectos que mejor se ajustan a los fines buscados.

En este trabajo no se analizarán los atributos que debe tener un esquema de priorización de proyectos para que sea efectivo puesto que está fuera del alcance propuesto. Sin embargo, la realidad es que un CIO no solo tiene la responsabilidad de ejecutar los proyectos de TI priorizados en un nuevo ciclo de planeamiento estratégico, sino que además debe lidiar con nuevos requerimientos y evolución de los sistemas existentes. Así, como los recursos de TI son limitados, también existe un esquema de priorización dentro de la función de TI, y en particular, en el contexto de la gestión de la deuda técnica, se asignarán recursos solo a los sistemas que lo justifiquen.

En esta sección, se presentarán algunos modelos que pueden ser utilizados como base para la priorización de recursos asignados a la gestión de la deuda técnica, como parte de los análisis que un CIO puede utilizar para racionalizar el portafolio de sistemas de una organización.

3.7.1. Matriz BCG

Descripción: La Matriz BCG, o matríz de crecimiento-participación es un modelo para analizar un portafolio de productos (Henderson, 1970) en términos de: (1) su participación en el mercado (vinculado a los ingresos) y (2) el crecimiento de este mercado (vinculado a los egresos). El modelo toma como supuesto que un producto con una gran participación en el mercado generará grandes dividendos, y uno que opere en un mercado en crecimiento requerirá grandes inversiones para posicionarse como líder. Esto genera cuatro tipos de producto: (1) perros, (2) signos de interrogación, (3) estrellas y (4) vacas lecheras.

Los perros tienen baja participación en un mercado con bajo crecimiento, con lo cual no generan ni consumen grandes cantidades de dinero, por lo tanto son candidatos a desmantelarse. Debe evaluarse si realmente es necesaria realizar una gestión de deuda técnica sobre estos negocios: como hay un bajo nivel de inversión sobre éstos, los mantenimientos evolutivos sobre aplicaciones asociadas a este negocio se estima que serán infrecuentes, con lo que no se justifica una gestión de deuda técnica.

Los signos de interrogación consumen dinero para poder crecer rápidamente, pero generan muy pocos ingresos, por lo que suelen ser deficitarios. Si logran crecer pueden convertirse en una estrella o si fracasan, se convierten en perros. Estos negocios deben analizarse cuidadosamente para determinar si vale el esfuerzo de invertir en ellos.

Las estrellas tienen una posición fuerte en un mercado que sigue creciendo, por lo que generan ingresos pero a su vez requieren inversiones en proporción semejante, lo que típicamente resultan ser compensados. Si el mercado deja de crecer puede convertirse en una vaca lechera.

Las vacas lecheras son líderes de mercados maduros con poco crecimiento, con lo que se asegura un fuerte ingreso de dinero y bajos niveles de inversión, con lo cual suelen ser superavitarias, y aportan el dinero necesario para que las estrellas y signos de interrogación puedan crecer.

Impacto sobre la deuda técnica: Según una clasificación de matriz BCG de las unidades de negocios o productos, existen varias alternativas de gestión de la deuda técnica (Figura 3.8), entendiendo que la deuda se gestiona sobre el portafolio de sistemas que gestionan o dan soporte a esa unidad de negocios.

En el caso de los perros, donde el negocio no termina de ser lo suficientemente atractivo para generar interés de inversión puesto que el mercado no tiene el potencial deseado y se tiene una baja participación, entonces no parece ser necesario realizar una gestión de la deuda técnica ya que el futuro de estos sistemas es incierto. Alternativamente, puede hacerse un seguimiento de la misma, observando si se genera deuda técnica pero sin invertir esfuerzo en bajarla.

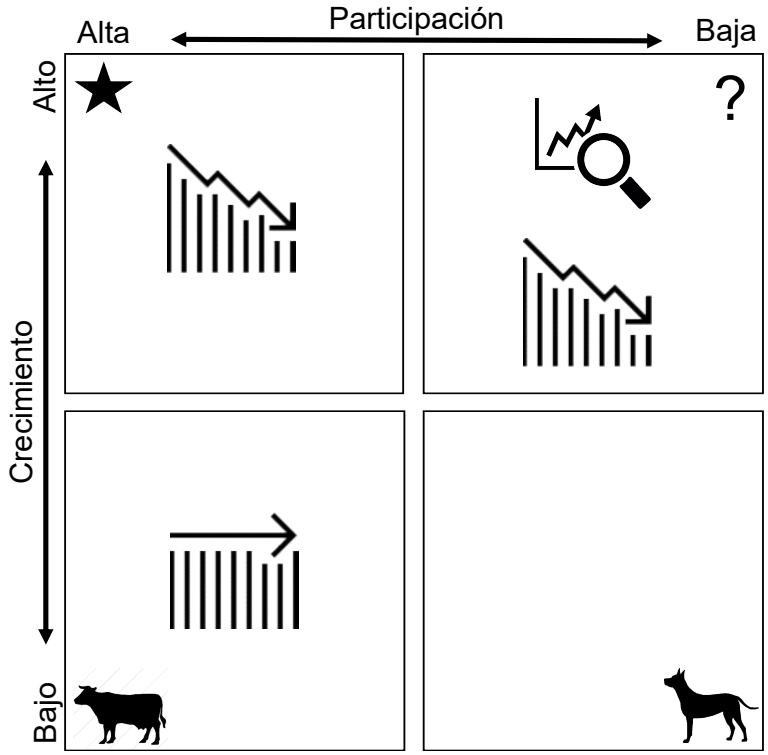


Figura 3.8: Gestión de deuda técnica recomendada, según Matriz BCG.

Para el caso de los signos de interrogación, que suelen ser deficitarios, la deuda técnica debe gestionarse con precaución. Aquí el CIO y su gestión tienen una doble responsabilidad: por un lado el poder seguir mejorando el posicionamiento en el mercado (mejorando el producto con nuevas funcionalidades rápidamente, mejor información de gestión) y por otro lado no afectar la calidad actual (disponibilidad, escalabilidad, performance). Esto es, por un lado se pretende velocidad y por otro calidad, variables que suelen estar en tensión y son afectadas por la cantidad de deuda técnica. Aquí puede ser recomendable una doble gestión: para aplicaciones que expongan nueva funcionalidad y beneficios de cara al cliente, gestionar la deuda técnica con un perfil de seguimiento para alcanzar tiempos de entrega reducidos, mientras que los sistemas de información que dan soporte al negocio, así como componentes y módulos reusables, gestionarla con un perfil de reducción, para asegurar cierta calidad interna que tienda a que futuras modificaciones sean posibles sin deteriorar el mantenimiento de los sistemas.

Los negocios estrella tienen una posición fuerte en el mercado y exis-

te espacio para seguir creciendo, aunque los ingresos e inversiones queden compensadas y no se generen ganancias genuinas. Si la estrategia de la empresa es seguir invirtiendo hasta convertir el negocio en una vaca lechera, entonces esta inversión debe también propagarse hacia sus sistemas, a efectos de poder mantener la base de clientes y los atributos de calidad (externos e internos). En particular, se recomienda invertir tiempo y esfuerzo para que la deuda técnica tienda a reducirse hasta lograr estabilizarla en niveles manejables cuando se convierta en una vaca lechera.

Finalmente, las vacas lecheras son unidades de negocios que generan ganancias. Es importante no perder la base de clientes (ya que hay poco mercado para atraer nuevos). Ya que esfuerzos anteriores por reducir la deuda técnica han tenido éxito, es un momento adecuado para reducir la inversión y mantenerla en los niveles actuales. Se supone que el negocio tiene cierta madurez en este momento y posiblemente algunos sistemas se hayan convertido en *legacy*, eficientes pero rígidos, donde las modificaciones serán escasas pero posibles, en cuyo caso, de nuevo, una gestión para intentar reducir la deuda técnica rendirá pocos frutos.

3.7.2. Tres horizontes de crecimiento

Descripción: Si la organización tiene la capacidad de planificar y ejecutar un portafolio de negocios con variados niveles de innovación, riesgo y retorno de inversión, los mismos pueden clasificarse de acuerdo al modelo de tres horizontes de crecimiento (Coley, 2009). En este modelo, se clasifican los negocios actuales y futuros de la empresa en tres grupos llamados H1, H2 y H3. El grupo H1 representa los negocios donde actualmente se obtienen las mayores beneficios, el grupo H2 identifica aquellos emprendimientos que probablemente sean redituables en el mediano plazo aunque requieran inversiones en la actualidad, y H3 son los emprendimientos, investigaciones, programas piloto o ideas que pueden llegar a generar beneficios a largo plazo. La Figura 3.9 ilustra el concepto.

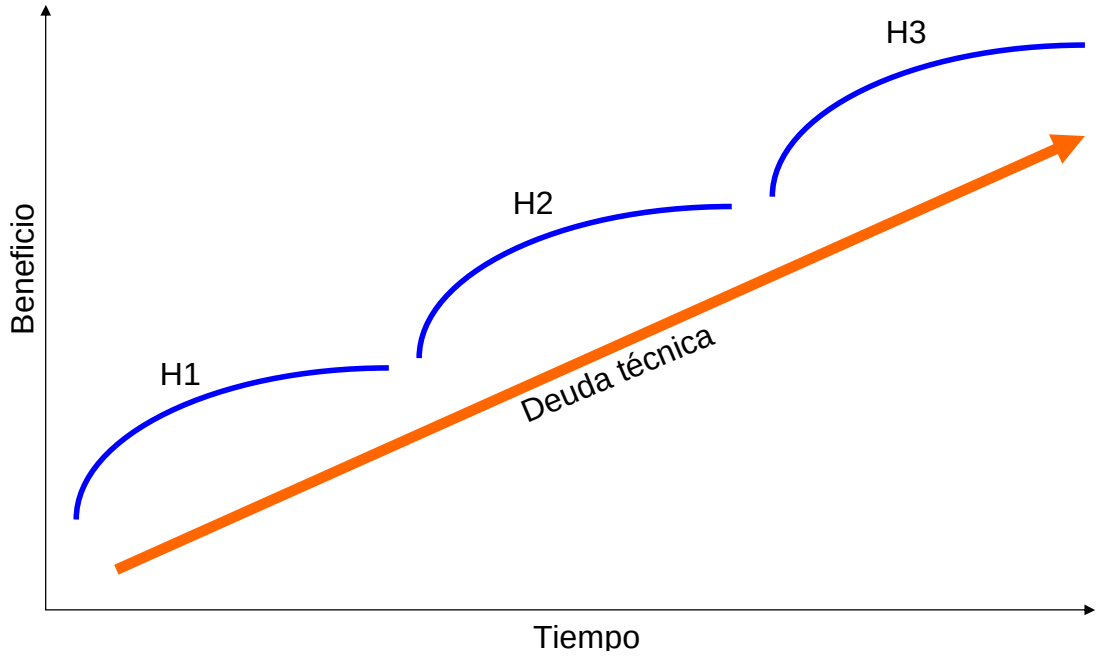


Figura 3.9: Modelo de tres horizontes de crecimiento (Coley, 2009).

Impacto sobre la deuda técnica: Clasificando los negocios bajo el modelo de los tres horizontes de crecimiento (Figura 3.9), deberían esperarse niveles de deuda técnica más reducidos en los sistemas que son la base actual del negocio, y niveles de deuda técnica más elevados en prototipos estratégicos y programas piloto que representan la apuesta de crecimiento del negocio futuro, donde posiblemente sea más prioritario la capacidad de innovación o el *time-to-market* que la mantenibilidad de los sistemas u otros atributos internos.

Como resultado, la gestión de la deuda técnica debería ser más estricta para los sistemas asociados a los negocios en H1, ya que son los que aportan los mayores beneficios a la organización, una gestión menos rigurosa para los sistemas asociados a los negocios en H2, y una gestión más permisiva para los clasificados bajo H3. Blank (2015) también apoya esta estrategia: a medida que el negocio sea exitoso (pasando desde H3 hacia H1) irá aumentando su importancia para la organización, y por lo tanto la operación estable conlleva comenzar a pagar la deuda generada en etapas previas del mismo.

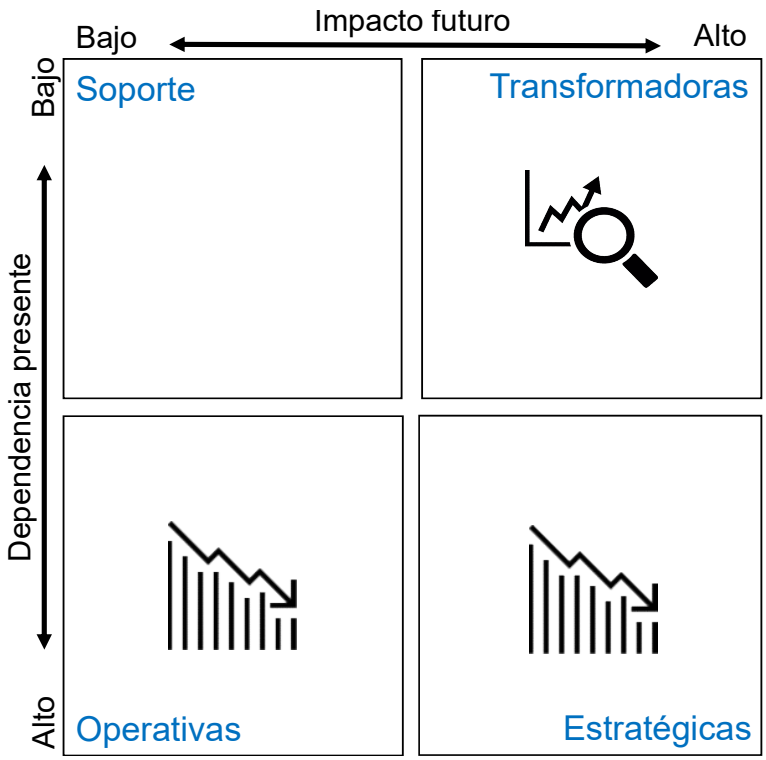


Figura 3.10: Gestión de deuda técnica recomendada, según Matriz de McFarlan.

3.7.3. Matriz de McFarlan

Descripción: La Matriz de McFarlan (Freijedo, 2015; McFarlan, McKenney, y Pyburn, 1983) es un modelo que permite clasificar las aplicaciones (sistemas de información) de acuerdo a la importancia actual y su impacto futuro para el negocio, y cataloga las aplicaciones en cuatro clases: (1) Soporte, (2) Operativas, (3) Estratégicas y (4) Transformadoras, como se ilustra en la Figura 3.10.

Las aplicaciones de Soporte aportan funciones complementarias al desarrollo de los negocios actuales y no son críticas para la operación diaria, además no poseen funcionalidades que sean importantes para el desarrollo futuro del negocio. Un ejemplo de este tipo de aplicaciones puede ser el envío de un email de felicitación por el cumpleaños de un cliente.

Las Operativas son críticas para las operaciones diarias, donde las fallas o problemas de disponibilidad tienen un impacto directo sobre las operaciones, pero no es crítica para poder desarrollar una estrategia de negocio con

impacto a futuro. Por ejemplo, los cajeros automáticos dentro de la sucursal de un banco son fuente importante de operaciones diarias, de modo que una falla en los mismos tiene un impacto en las operaciones, pero seguramente no tenga impacto futuro si la estrategia del banco es impulsar la realización de operaciones por *home banking*.

Las aplicaciones Estratégicas tienen una gran importancia tanto en las operaciones diarias como en las futuras operaciones, ya que están alineadas con los objetivos y estrategia de crecimiento del negocio. Un fallo en alguna aplicación no solo puede afectar las operaciones diarias sino también la imagen de la empresa o su participación en el mercado. Un ejemplo es el sistema de *home banking* de un banco, donde una falla afecta tanto a las operaciones diarias como también ocasiona una pérdida de imagen.

Finalmente, las aplicaciones Transformadoras tienen un gran valor potencial a futuro pero el negocio no depende de ellas para subsistir económicamente. Típicamente son aplicaciones que exploran nuevas tecnologías, o complementan negocios existentes para aportar ventajas competitivas o un factor de diferenciación. Por ejemplo, el reciente lanzamiento de la billetera virtual Todo Pago por parte del grupo empresario que controla Visa, Pagomiscuentas y la red Banelco: es un nuevo servicio que tal vez no se espera que aporte gran cantidad de operaciones diarias, pero es de importancia estratégica ya que le permite competir a la empresa directamente contra Mercadopago.

Impacto sobre la deuda técnica: Para las aplicaciones de Soporte, que no representan aspectos críticos ni presentes ni futuros, no es necesario realizar una gestión de la deuda técnica, o si se realiza, tal vez con un perfil de seguimiento solamente.

Las aplicaciones Operativas son importantes para el funcionamiento normal del negocio, por lo que es recomendable tratar de reducir la deuda técnica hasta niveles tolerables que no impidan ni demoren el desarrollo de nuevos requerimientos operativos. Lo mismo puede decirse de las aplicaciones estratégicas.

Las Transformadoras son aplicaciones que no son vitales para las ope-

raciones diarias pero si tienen potencial impacto en el futuro. Por un lado no se requieren grandes niveles de confiabilidad y calidad puesto que no son críticos en el presente, y por el otro probablemente estén explorando nuevas tecnologías o conceptos, con lo que están en una etapa de aprendizaje y evolución rápida, y la aplicación de las nuevas herramientas no sea la más adecuada, por lo que el perfil de deuda técnica se espera que sea alto. Asimismo, se espera que estas aplicaciones sean fuente de diferenciación y provean una ventaja competitiva, de modo que variables como el *time-to-market* sean importantes al momento de introducir nueva funcionalidad, por lo tanto la gestión de la deuda técnica puede usarse como variable estratégica (en vez de ser una necesidad) por lo que solo debe limitarse a hacer un seguimiento a efectos de tener una valoración confiable al momento en que se consoliden como aplicaciones Operativas o Estratégicas.

3.8. Esquema de abastecimiento

Descripción: El esquema de abastecimiento define *quién* llevará a cabo las actividades de TI de la organización, en particular para esta obra, lo referente a la provisión y mantenimiento de los sistemas de información. Las dos alternativas habituales son que las satisfaga un departamento interno (*insourcing*) o un proveedor externo (*outsourcing* o terciarización). Sin embargo, no es una decisión binaria, puesto que los mejores resultados suelen venir de adoptar un esquema mixto, como sugiere Lacity, Willcocks, y Feeny (1996), donde los sistemas críticos y que generen una diferenciación clara de los competidores puedan proveerse internamente, y los que aporten beneficios marginales y no sean fuente de diferenciación puedan ser terciarizados sin comprometer las fortalezas de la empresa.

El desarrollar los sistemas internamente tiene como ventaja que pueden gestionarse con mayor transparencia, ya que los roles de gestión del proyecto quedan completamente dentro de la organización, así como el *know-how* para llevarlo a cabo. Asimismo, puede disponerse de los recursos internos con más libertad en caso de que las prioridades cambien abruptamente.

Sin embargo, las empresas han optado por la terciarización de proyectos por diferentes motivos: porque no posee la capacidad técnica para llevarlos a cabo, porque es menos costoso que hacerlo internamente, porque la dirección no considera que el área de TI aporte a la competitividad, entre otras.

No obstante, la terciarización tiene sus riesgos. Las empresas proveedoras tienden a subestimar el esfuerzo, costo o tiempo requerido para el desarrollo de software, con el objetivo de ofrecer una cotización atractiva y ganar un contrato (Taylor, 2006). La consecuencia es que el proyecto de desarrollo termina extendiéndose en el tiempo más allá de lo comprometido, o bien termina sufriendo la calidad final del producto. Finalmente la relación con el proveedor se va deteriorando, perdiendo la confianza del cliente, hasta el punto de que la relación se termine.

Lo anterior es otro riesgo de la terciarización: el de quedar atrapado en una relación con un proveedor en particular que no alcanza las expectativas, o bien que se torna demasiado costoso para la organización. Las alternativas son buscar otro proveedor o bien continuar con los proyectos en forma interna. El costo de cambio puede ser alto. Para mitigar estos costos, Whitten (2010) recomienda mantener una mínima capacidad técnica interna (personal, infraestructura, etc.) y establecer relaciones de terciarización con múltiples proveedores (para diversificar el riesgo de terminar una relación y reducir la dependencia).

Impacto sobre la deuda técnica: En lo relativo a la deuda técnica, las herramientas de gestión presentadas en este trabajo pueden ser utilizadas con confianza bajo un esquema de abastecimiento interno, ya que el CIO puede ejercer el control necesario para que los objetivos definidos para los niveles de deuda puedan ser alcanzados. En sistemas de información en los que se decide terciarizar su desarrollo, esta capacidad de control disminuye. Para mitigar el riesgo, deben establecerse claramente los criterios de aceptación en el contrato celebrado con el proveedor, no solo los relativos a las funcionalidades que deben ser desarrolladas y la calidad externa, también los aspectos de calidad interna.

Al momento de seleccionar un proveedor adecuado, se sugiere tener en cuenta las recomendaciones de DiRomualdo y Gurbaxani (1998), en particular asegurarse que el proveedor tenga las competencias y *know-how* necesarios, así como que su cultura y prácticas de trabajo sean compatibles. Esto hará más fluida la comunicación y el trabajo entre las partes.

Como la deuda técnica es un aspecto interno que no siempre el proveedor está dispuesto a compartir, se deben establecer criterios para poder fijar objetivos concretos con respecto a las expectativas de calidad interna, que luego la empresa sea capaz de verificar. En efecto, se requiere acordar con el proveedor un SLA (*Service Level Agreement*, o Acuerdo de Nivel de Servicio).

El CIO no gestionará la deuda técnica pero establecerá los objetivos de gestión que el proveedor debe seguir, esto se logra con la asistencia del personal técnico (arquitectos, desarrolladores, administradores de bases de datos, diseñadores gráficos, etc.) para redactar la arquitectura a respetar, los esquemas de base de datos, las reglas de diseño, buenas prácticas adoptadas por la empresa, etc. Todo esto conformará un conjunto de directivas que el proveedor debe respetar y cada cierto tiempo verificadas por el equipo a cargo del CIO.

El control de estas directivas o reglas de gestión de deuda técnica se simplifica si pueden ser verificadas automáticamente por una herramienta. Por ej. Sonarqube permite definir un conjunto de reglas que pueden exportarse hacia otra instancia de Sonarqube (la que posea el proveedor) de manera que en su gestión pueda verificar si se están cumpliendo. Finalmente, cuando el proveedor entrega el producto de software, junto a su código fuente, el cliente (la organización) puede ejecutar la verificación y determinar si el SLA se cumplió.

Hoy en día se está llevando a cabo un esfuerzo de estandarización para construir los SLAs. El CISQ (*Consortium for IT Software Quality*)[5] es un grupo compuesto por expertos comprometidos en un estándar de métricas computables para medir la Calidad y Tamaño del software. Este consorcio apoya la definición de SLAs para incentivar la performance de los proveedores, en particular, se encuentra en proceso de estandarización medidas que reflejen

[5] `http://it-cisq.org/`

los niveles de defectos o violaciones de buenas prácticas de arquitectura y programación (Curtis, Dickenson, y Kinsey, 2015). Asimismo, está definiendo métricas para la medición estandarizada de la Seguridad, Confiabilidad, Eficiencia y Mantenibilidad, esta última de particular interés para gestionar la deuda técnica (*Automated Source Code Maintainability Measure (ASCMM) V1.0*, 2016).

Por último, cabe notar que no se recomienda contratar el mantenimiento de un sistema a un proveedor con el único objetivo de transferir su deuda técnica fuera de los indicadores del negocio. La deuda técnica no puede *transferirse* hacia un proveedor con objeto de disminuirla, como sería el caso de la transferencia de un riesgo hacia una compañía de seguros. La gestión de la deuda técnica todavía debe realizarse, lo que genera trabajo por parte del proveedor y auditorias por parte de la empresa.

Capítulo 4

Gestión de deuda técnica

En el capítulo anterior se ha cubierto parte del segundo objetivo específico propuesto, analizando cómo influyen elementos y modelos típicamente utilizados para dar forma a la gestión estratégica de TI a la generación y gestión de la deuda técnica.

En este capítulo se continúa con el segundo objetivo del presente trabajo, cubriendo la gestión de la deuda técnica con una visión más detallada, describiendo las actividades que componen su gestión diaria. Se amplían las características y actividades que abarcan los *perfiles de gestión* propuestos en la Sección 3.2. Por último, se propone que la deuda técnica sea gestionada en el marco de un proceso de mejora continua, puesto que no es posible concebirla como un proyecto puntual para solucionar problemas de desarrollo de software, por el contrario, la gestión de la deuda técnica y la del proceso de desarrollo avanzan en tándem.

4.1. Actividades de gestión

Se han identificado un conjunto de actividades esenciales para llevar a cabo una gestión de la deuda técnica (Li, Avgeriou, y Liang, 2015; Avgeriou, Kruchten, Nord, Ozkaya, y Seaman, 2016). Esta obra toma estas actividades como base para luego elaborar perfiles de gestión para algunos casos comunes.

Las actividades para gestionar la deuda técnica pueden clasificarse en

(1) centrales y (2) de soporte, descritas a continuación.

4.1.1. Actividades centrales

Las actividades centrales son las que sirven para gestionar la deuda técnica propiamente dicha. A continuación se describe brevemente el propósito de cada una de ellas.

Identificación. Se identifican los elementos o artefactos que presentan deuda técnica.

En el mejor de los casos se identifican usando herramientas que automatizan esta tarea, como las de análisis estático de código fuente. Otro medio es mediante reuniones sobre las decisiones de diseño tomadas con interesados.

Desde el punto de vista de la madurez de la función de TI, dicha tarea se ve simplificada si la organización ya cuenta con un inventario de sistemas que relaciona las unidades de negocios y los sistemas asociados a cada una de ellas, o aún mejor disponer de una herramienta APM (*Application Portfolio Management*), sin embargo no es un requisito excluyente: se puede ir construyendo este inventario a medida que se van analizando los sistemas y otros artefactos en búsqueda de deuda técnica.

Medición. Se cuantifican los costos y beneficios de la deuda técnica por medio de técnicas de estimación.

El costo puede calcularse razonablemente estimando el Capital (Sección 2.5) en términos monetarios o de esfuerzo. En este sentido, las herramientas de análisis realizan un excelente trabajo midiendo el Capital en aspectos que son automatizables (ej. violaciones de buenas prácticas sobre el código fuente). Sin embargo, esto representa solo una porción de la deuda técnica: temas como malas decisiones efectuadas a nivel de arquitectura no son tan fácilmente cuantificables, por lo que se recomienda tomar la deuda técnica detectada por las herramientas como una *cota inferior* para la misma (Sharma y Samarthyam, 2015).

Otro aspecto relacionado a los costos es el Interés (el esfuerzo extra requerido por el equipo de trabajo por no haber pagado el Capital de la deuda técnica), que es de difícil estimación (ver Apéndice A.4). Además de la calidad interna del sistema, el esfuerzo extra podría llegar a estar influido por muchos otros factores que disminuyen la productividad del equipo, por ej. la cantidad de llamadas telefónicas recibidas por desarrollador por día, o la cantidad de emails que tenga que contestar. A diferencia del Capital, donde se puede realizar una proyección a futuro basándose en un promedio de, por ej., la cantidad estimada en las últimas tres entregas, el Interés es menos predecible: solo si los nuevos requerimientos afectan componentes o porciones de código fuente con gran deuda técnica este interés es relevante, por lo que un cálculo basado en datos históricos tiene una ventana de utilidad en el tiempo muy pequeña.

La cuantificación de los beneficios también en más subjetiva que la del Capital: la tarea es dar un sentido positivo a la deuda técnica encontrada, entonces se debe tomar como un supuesto que esta deuda técnica ha sido generada en forma *intencional* (ver el Cuadro 2.3), es decir que representa una estrategia para alcanzar un objetivo particular. En ese contexto, podría decirse que si la deuda técnica equivale al esfuerzo que el equipo de trabajo realiza durante un *Sprint*, implica que el equipo ha podido entregar los requerimientos solicitados *un Sprint antes* comparado a una estrategia de no adquirir deuda técnica, entonces el beneficio es el valor de negocio extra que se obtuvo por haber entregado antes.

Priorización. Elaborar un ranking de items de deuda técnica a efectos de identificar los que convenga solucionar primero y cuales pueden ser tolerados.

Una vez identificados y medidos los artefactos, es necesario priorizarlos por algún criterio para poder concentrar el foco en cumplir con los objetivos de negocio. Por ejemplo, cada artefacto medido puede contener deuda de diferentes niveles de severidad (crítica, mayor, menor, informativa), con lo que luego habilita una estrategia de pago donde primero se soluciona la deuda de mayor severidad que requieran el menor esfuerzo posible.

Otra priorización posible es asignar un rating o escala cualitativa a un

artefacto, dependiendo de la comparación entre el Capital de la deuda y el costo estimado de desarrollar nuevamente el artefacto. Así, si el Capital representa hasta el 1 % del costo de desarrollo, se le asigna el rating A (el mejor), si es mayor al 1 % y hasta el 2 % se asigna el B, etc. El método SQALE emplea un rating de cinco niveles (A, B, C, D, E) siendo E el peor (Letouzey, 2016).

A nivel de unidad de negocio, las herramientas proporcionadas en el Capítulo 3 pueden usarse en el contexto de una priorización de gran escala para determinar donde concentrar los esfuerzos de gestión.

Repago. También conocida como Remediación. Durante esta actividad es que la deuda técnica es solucionada o mitigada.

Implica modificar los artefactos que poseen deuda técnica utilizando buenas prácticas de desarrollo de software, por ejemplo: *refactoring*, TDD (*Test Driven Development*), entre otras. No es necesario solucionar todos los items de deuda técnica encontrados. Debe notarse que el repago implica contar con la conformidad del cliente, y en la mayoría de los casos no generar nueva deuda técnica, o si lo hace debería tener menor severidad que la que se está pagando.

Monitoreo. Se realizan observaciones sobre los items de deuda técnica que no han sido solucionados, para evaluar su evolución en términos de costo y beneficio.

Los artefactos que no han sido repagados o donde la deuda técnica se ha remediado parcialmente, son objeto de monitoreo. Por ej. si un componente expone problemas tanto de concurrencia como de seguridad, dependiendo de los objetivos del negocio puede ser que se solucione uno y el otro permanezca en forma residual para ser remediado en otro momento.

4.1.2. Actividades de soporte

Las actividades de soporte cruzan el proceso de gestión, aportando información valiosa para poder realizar una gestión efectiva y complementan las actividades centrales asistiendo en la toma de decisiones.

Documentación. Los items de deuda técnica pueden ser documentados de diferentes formas, especialmente los que no son identificados por herramientas automáticas, como los encontrados durante reuniones de diseño o arquitectura. La documentación puede adoptar diversas formas, como documentos de diseño, un backlog, comentarios en código fuente, wikis, etc.

Comunicación. La comunicación de la deuda técnica es una de las actividades fundamentales, ya que le otorga transparencia al proceso de desarrollo de software, otorgando visibilidad hacia los interesados (niveles técnicos y gerenciales, clientes, usuarios, etc.), para que las partes puedan ser debidamente informadas y los problemas puedan ser discutidos.

La situación ideal es comunicar con evidencias y problemas concretos, acompañados de su nivel de severidad, urgencia, costos y beneficios para poder evaluar el riesgo y poder tomar con la mayor información posible la decisión de pagar la deuda o continuar con los requerimientos del negocio.

Trazabilidad. Al momento de realizar el pago de la deuda técnica, es conveniente conocer la traza entre los items que contienen la deuda técnica a pagar y otros artefactos. Conocer las dependencias entre los diversos componentes que integran un sistema aporta información valiosa que ayuda a analizar el impacto de los cambios.

A modo de ejemplo, es posible que –en el contexto de un sistema con un diseño orientado a objetos– haya una sola clase que tenga una gran responsabilidad convirtiéndola en una *God Class*. Se ha estudiado que estas clases son modificadas más frecuentemente y contienen más defectos que las que tienen menos responsabilidades (Zazworka, Shaw, Shull, y Seaman, 2011). Identificar las clases que dependen de ellas es importante al momento de evaluar el posible impacto de un defecto.

Otro ejemplo en el que conocer las dependencias es útil es cuando se detecta alguna vulnerabilidad de seguridad en un componente en que se basa el sistema, por lo tanto haciéndolo vulnerable también. Esto permite evaluar si es posible solucionar la vulnerabilidad por medio de una simple actualización

del componente o bien se necesita modificar el sistema para evitar que sea explotada.

La traza no solo implica disponer de la relación de dependencia entre los artefactos, tal vez más importante que eso son las decisiones de diseño y arquitectura que confluyeron en la implementación actual. Una buena documentación que ayude a explicar el por qué de las decisiones es invaluable.

En un caso extremo, conocer la traza puede ayudar a renegociar requerimientos funcionales y no funcionales del sistema. El saber y conocer la cadena de las preguntas y respuestas, decisiones de negocio y de diseño que dieron lugar al sistema como existe actualmente puede poner al descubierto objetivos difíciles de alcanzar, o que ya no son prioritarios, lo que habilitaría en última instancia a una renegociación de requerimientos que permita continuar con la evolución del sistema a un costo menor.

Prevención. Reducir la deuda técnica una vez que existe es la alternativa de gestión más frecuente. Otra alternativa es evitar generarla. La prevención es una actividad que resulta más efectiva cuando el equipo de trabajo esta capacitado y es disciplinado en cuanto a la aplicación de buenas prácticas de desarrollo de software.

La idea es evitar generar deuda técnica en forma inadvertida (Cuadro 2.3) y no intencional, o bien intencional de corto plazo (ver Figura 2.4), esto es, tratar de acumular solamente deuda que represente un beneficio estratégico.

La Figura 4.1 nombra algunas estrategias de prevención utilizadas de acuerdo a una reciente encuesta. Asimismo, las siguientes son algunas de las buenas prácticas que son útiles en la prevención de la deuda técnica:

Incentivos. A veces es necesario realizar un cambio cultural en el equipo de trabajo. Establecer incentivos recompensando una baja generación de deuda técnica o de defectos es una forma de prevenir (Rao, 2016).

Definición de *TERMINADO*. Es importante acordar y establecer lo más claramente posible el significado de que un trabajo esté “terminado”. Esta definición debe ser visible y conocida por todos los miembros del equipo,

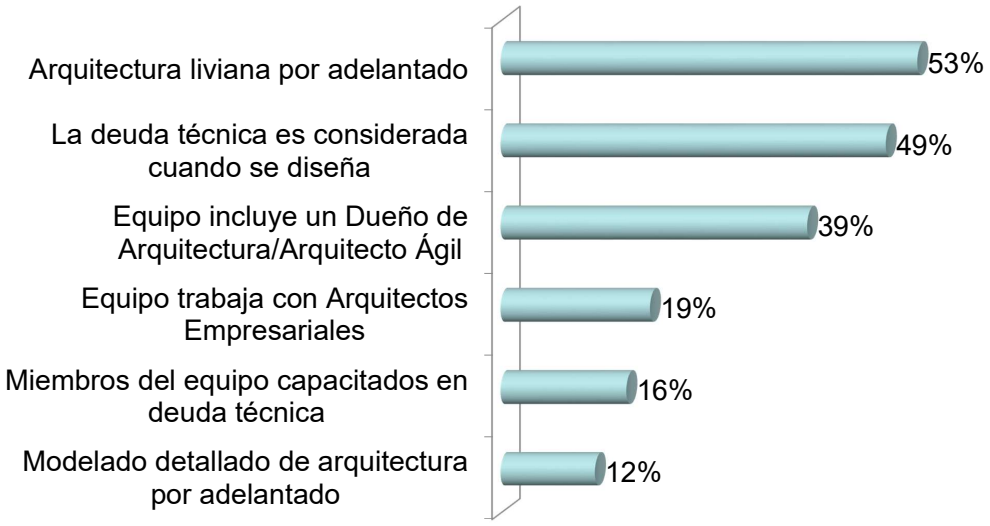


Figura 4.1: Estrategias de prevención típicamente usadas. Fuente: *SA+A 2015 Q1 Agile State of the Art Survey* (Ambler, 2015).

indicando las propiedades que debe tener un artefacto terminado y los procedimientos que deben cumplimentarse. Esto evita que haya elementos del software que queden incompletos en forma inadvertida, lo que constituye deuda técnica (Rao, 2016; Sterling, 2010, p. 44).

TDD. *Test Driven Development*, o desarrollo dirigido por tests, es una práctica que pone el acento en que todo el software construido debe tener tests automatizados que pongan a prueba su correcto comportamiento. Esto previene la introducción de defectos o regresiones al momento de realizar modificaciones. Además, la práctica de TDD tiende a producir porcentajes de cobertura mayores, una característica que impacta positivamente en el mantenimiento (Sterling, 2010, p. 37).

Análisis estático de código fuente. Con herramientas automatizadas que analizan el código fuente en búsqueda de defectos y potencial deuda técnica. Ejecutándolas lo más frecuentemente posible, ayuda a detectar en forma temprana la deuda introducida en forma inadvertida.

Integración continua. Práctica donde en cada modificación del código fuente se construye el software, se testea, se ejecutan pruebas de inte-

gración y se realizan otras actividades en forma automática. Esto provee una retroalimentación temprana, que detecta en cuestión de minutos si se ha introducido defectos (Sterling, 2010, p. 58).

Sesiones de revisión. Son reuniones entre los miembros del equipo, en forma periódica, para obtener feedback y discutir opciones de diseño. Esto habilita al equipo a tomar decisiones que tiendan a clarificar la estructura del sistema (Sterling, 2010, p. 147).

Programación de a pares. Dos programadores se sientan frente a una estación de trabajo. Uno de ellos escribe el código fuente y el otro colabora proveyendo opinión y feedback inmediato sobre lo que se está haciendo, mejorando así las decisiones de diseño. Esta práctica tiende a elevar la calidad del código (Sterling, 2010, p. 148). Incluso, esta práctica es parte esencial de la metodología XP (*Extreme Programming*) (Beck y Andres, 2004).

Refactoring. Es una práctica fundamental, tanto para prevenir como para remediar la deuda técnica. El refactoring es el “proceso de modificar el sistema de manera tal que no se altere el comportamiento externo pero mejorando su estructura interna” (Fowler, 1999).

DRY, KISS, YAGNI. Estos acrónimos que describen algunas prácticas de programación (San Miguel, 2014). DRY – *Don’t Repeat Yourself* (No te repitas), se refiere a evitar el copiado y pegado de código fuente que realiza lógica semejante. Esto genera duplicaciones de código e impacta negativamente en la mantenibilidad del sistema. KISS – *Keep It Simple, Stupid!* (manténlo simple, estúpido!) se refiere a quedarse con opciones y utilización de técnicas que produzcan diseños simples de entender. Por último, YAGNI – *You Aren’t Gonna Need It* (no vas a necesitarlo) se refiere a no producir diseños que traten de capturar posibles requerimientos futuros, pero sin la certeza de que serán necesarios.

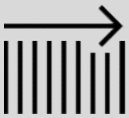
 **Seguimiento**	 **Estabilización**	 **Reducción**
Actividades centrales		
Identificación	Identificación	Identificación
	Medición	Medición
	Priorización	Priorización
		Repago
Monitoreo	Monitoreo	Monitoreo
Actividades de soporte		
Documentación	*Documentación*	*Documentación*
Comunicación	*Comunicación*	*Comunicación*
	Trazabilidad	*Trazabilidad*
	Prevención	*Prevención*

Figura 4.2: Perfiles de gestión de deuda técnica.

4.2. Perfiles de gestión

Siguiendo con lo comentado en la Sección 3.2, de acuerdo a la rigurosidad con que se realizará la gestión de la deuda técnica sobre los artefactos o sistemas que conforman una unidad de negocio, el autor propone tres perfiles de gestión con crecientes niveles de intensidad, nombrados de acuerdo al objetivo de gestión de deuda técnica que se desea perseguir: (1) Seguimiento, (2) Estabilización y (3) Reducción. Cada perfil esta compuesto por una combinación de las actividades de gestión identificadas en la Sección 4.1. En la Figura 4.2 se muestran estos perfiles.

La separación en los tres perfiles descritos arriba no pretende fijar ni pre-establecer una “mejor práctica” para la gestión de la deuda técnica. Solo pretende abarcar un conjunto de situaciones comunes que pueden llegar a encontrarse al momento de tomar decisiones para abordar la gestión de la deuda técnica. Queda a criterio del lector poder personalizar las actividades de gestión utilizadas que aporten el mayor valor al establecer su propia práctica.

4.2.1. Perfil de Seguimiento

En el perfil de seguimiento, el objetivo es observar la deuda técnica que se va generando en el transcurso del tiempo. Es una estrategia de gestión conveniente para usar en etapas iniciales de una nueva unidad de negocio, donde los sistemas irán evolucionando en formas que son muy difíciles de predecir y planificar de antemano. Estos sistemas son modificados frecuentemente, y deben adaptarse con agilidad a los cambios para responder rápidamente a nuevas oportunidades y amenazas. La deuda técnica generada en esta etapa es (idealmente) intencional y es usada con fines estratégicos para alcanzar objetivos de negocio.

Las principales actividades de gestión involucradas son la identificación de los sistemas, sus componentes y artefactos directamente relacionados (como documentos de diseño, documentos de arquitectura, esquemas de bases de datos, etc.) que poseen deuda técnica. Pueden utilizarse herramientas automatizadas, pero también puede ser factible llevar un registro de problemas recurrentes escuchando atentamente las discusiones de diseño en reuniones con personal técnico y otros interesados. También es conveniente estar atento a síntomas, como una alta tasa de defectos, gran cantidad de llamados a la mesa de ayuda, etc.

Si bien realizar una medición para estimar el costo o esfuerzo necesarios para reducir la deuda sea una tarea demasiado detallada en este estadio, es conveniente realizar una valoración de los problemas encontrados para su posterior análisis. Durante el período de tiempo que dure el seguimiento es aconsejable ir monitoreando la evolución de los elementos identificados, a fin de ir evaluando la necesidad de pasar a un perfil de Estabilización o bien de Reducción de deuda técnica, o tal vez de dejar de gestionarla explícitamente en caso de que se quiera focalizar el seguimiento en otros sistemas más críticos. Es conveniente ir documentando los hallazgos en la valoración y el monitoreo al realizar dicho seguimiento de la deuda, para luego poder comunicar a los interesados haciendo visibles potenciales riesgos.

4.2.2. Perfil de Estabilización

En el perfil de estabilización, el objetivo es colocar una cota superior a la deuda técnica para mantenerla en niveles tolerables. Es una opción de gestión adecuada en sistemas de información que sirven de apoyo para las operaciones de unidades negocios ya establecidas. Estos sistemas están operando en producción desde cierto tiempo, ya tienen un grado de madurez y están en una etapa de mantenimiento evolutivo, donde los nuevos requerimientos son frecuentes pero predecibles puesto que están alineados a objetivos ya definidos. Inclusive, este perfil es adecuado para la gestión de la deuda en sistemas legacy, que son muy críticos o están demasiado cerca del final de su ciclo de vida como para introducir modificaciones grandes. La deuda técnica existente en estos sistemas es tolerable pero futuras modificaciones podrían llegar a aumentarla si no se gestiona adecuadamente.

Al igual que en el perfil de Seguimiento, se identifican los sistemas con deuda técnica y se los analiza para descubrir los items con deuda, para luego proceder a estimar el costo y esfuerzo de repararlos. Tanto la identificación y medición son normalmente asistidas por herramientas especializadas, aunque puede haber elementos de deuda que no sean detectados por las mismas, con lo que deban ser complementadas por estimaciones manuales.

Una vez desagregada la deuda técnica en items, se los prioriza con algún criterio a efectos de diferenciar los problemas más críticos de los menos críticos, o diferenciar los más urgentes de los menos urgentes. Una buena táctica de priorización de los items es seleccionar los más críticos, y entre ellos, los que menos costo o esfuerzo requieren por parte del equipo de trabajo.

En forma periódica, a medida que se está desarrollando un nuevo requerimiento, debe medirse la deuda técnica para compararla contra el umbral de estabilización definido. Si éste es superado (tal vez con un margen de tolerancia pre-acordado) deben iniciarse acciones para contrarrestarlo a efectos de que converja hacia el nivel requerido.

El umbral o cota superior fijada para la deuda técnica debería ser capaz de ser comprendida por el CIO u otros interesados, en términos de esfuerzo

aproximado. Por ej. podría definirse como el esfuerzo que el equipo de trabajo puede realizar en el período de tiempo de 3/4 de Sprint o iteración de desarrollo, con un margen de error del 25 %, así si el Sprint es de 2 semanas (10 días laborales), entonces solucionar la totalidad de la deuda técnica no debería superar los 7,5 días de trabajo del equipo, y con el margen incluido debería estar entre los 5 y 10 días.

En este perfil, la documentación de los hallazgos de deuda suele automatizarse ya que las herramientas generan reportes detallados, aunque sigue siendo necesario documentar los hallazgos que no son detectados automáticamente. Asimismo, en este perfil cobra vital importancia el hacer visible la deuda hacia los interesados, tarea que se facilita al poder proveer evidencia concreta tras las mediciones efectuadas, y poder concientizar o advertir sobre riesgos potenciales.

Para establecer una traza, el análisis y medición arrojan información sobre las dependencias y relaciones entre componentes y otros artefactos usados por el sistema. También deben tenerse en cuenta decisiones pasadas y futuros requerimientos. Esta información permite tomar decisiones sobre la gestión de la deuda en otros artefactos, que puede tomarse en cuenta para complementar la priorización y las consiguientes acciones correctivas.

Por último, la estabilización de la deuda se logra de dos formas posibles: o bien los nuevos requerimientos agregan deuda y debe eliminarse deuda de la base de código existente, o bien los nuevos requerimientos no agregan deuda en cantidades significativas. Esta última opción es la más madura, aunque para ello hay que introducir prácticas de prevención efectivas.

4.2.3. Perfil de Reducción

En el perfil de Reducción, el objetivo es reducir la deuda técnica a niveles tolerables. Es necesario enfocarse en la reducción de la deuda técnica cuando comienza a evidenciarse una caída en la productividad del equipo de trabajo debido a que el sistema sufre de problemas de calidad (por ejemplo: mantenibilidad, confiabilidad, seguridad, portabilidad, performance, etc.), razón por la cual el equipo de trabajo se ve impedido de completar nuevos

requerimientos en los plazos acordados.

Los problemas de una excesiva deuda técnica suelen aparecer en unidades de negocio que han superado la etapa de incubación e innovación y han comenzado una etapa de consolidación, o se intensifican las presiones de entrega por cambios en el mercado (para una lista más exhaustiva, diríjase a la Sección 2.2).

Al igual que en el perfil de Estabilización, la deuda técnica se identifica y se mide para estimar el costo y esfuerzo de reparación, así como también se prioriza para subsanar primero los problemas más críticos.

Sin embargo, el objetivo no es mantener la deuda técnica en el nivel actual sino disminuirla, lo que presupone que se dedicará tiempo y esfuerzo del equipo de trabajo para repagar la deuda (solucionar problemas), en vez de desarrollar nuevos requerimientos. En esta situación es que tiene más sentido no solo evaluar el costo de repagar sino también el beneficio esperado. La priorización debe ser tal que el tiempo invertido en el pago de esa deuda (parte del capital) implique una reducción en el interés de la misma (el tiempo extra invertido por no haber pagado antes).

En forma análoga a la Estabilización, se debe establecer un umbral máximo (más bajo que el nivel de deuda actual), y deben monitorearse periódicamente los niveles de deuda para comprobar que efectivamente el esfuerzo invertido conlleva a una reducción en la misma. Una vez que la deuda llega a niveles tolerables, o bien se ha eliminado la deuda de los artefactos que impedían el progreso, puede cambiarse el perfil de gestión hacia el de Estabilización.

La documentación y comunicación de los hallazgos encontrados y a repararse deben formalizarse con los interesados del sistema, esto es debido a que debe detenerse momentáneamente el desarrollo de nuevos requerimientos para subsanar los problemas encontrados. Por ejemplo si se está usando una metodología ágil como Scrum, debe ingresarse la deuda técnica en el backlog del producto, acordar su priorización, estimar el esfuerzo y en cual sprint será pagada.

La trazabilidad de los artefactos del sistema con sus dependencias y re-

laciones con otros artefactos proveen información valiosa respecto al alcance de las modificaciones a realizar para pagar la deuda, ya que permite dimensionar y estimar mejor el esfuerzo necesario. Permite analizar en profundidad si algún artefacto es parte del problema que impide elevar la productividad para desarrollar nueva funcionalidad.

Por último, la gestión suele complementarse adoptando prácticas de prevención, evitando introducir nueva deuda técnica.

4.3. Roles y responsabilidades

La gestión de la deuda técnica es realizada por personas con distintos roles. Por un lado está el equipo de trabajo, que tiene el rol y última responsabilidad operativa por generarla o bien remediarla. Sin embargo, existen otros dos roles que son necesarios y complementarios.

Por un lado, al momento de la medición, deben definirse las reglas o violaciones de calidad que serán tenidos en cuenta para computar el capital de la deuda técnica. Según como se establezca esta configuración de reglas, la estimación puede arrojar resultados diferentes. Por ej. si para un sistema la seguridad no es prioritaria, tal vez se quiera configurar la medición (o la herramienta de medición) para que la muestre con una severidad baja, en vez de prioritaria. Asimismo, las reglas de estilo del código fuente (por ej. usar tabs o espacios) pueden no ser tenidas en cuenta en la estimación. Este rol debe realizarlo personal técnico calificado y con experiencia en deuda técnica, diseño y arquitectura. Un buen candidato para el rol es un arquitecto de la organización o del equipo.

Por otro lado, al momento de la priorización y de establecer los umbrales de Estabilización o Reducción, quien tome la decisión final debe tener experiencia tanto en el dominio del problema en que opera el sistema como en los aspectos del negocio. En opinión del autor, son los aspectos de negocios los que deben dirigir la evolución de la deuda técnica a efectos de poder tanto progresar como alcanzar los objetivos buscados. En este caso, un perfil ideal sería alguna persona que tenga interés en el cumplimiento de objetivos de ne-

gocio, como un responsable de proyecto (Líder de Proyecto, *Project Manager*), y en última instancia el CIO.

Cuando estos dos roles son ejercidos por personas distintas, se establece una suerte de control por oposición sobre una variable en tensión. En efecto, el personal técnico tomará responsabilidad por la calidad de la estructura interna del sistema, mientras que un responsable de producto o proyecto tomará responsabilidad por el cumplimiento de los objetivos de negocio: el comportamiento natural de los primeros es hacer que la deuda técnica se reduzca, mientras que los segundos desearán lo opuesto.

4.4. Ciclo de vida de mejora continua

Se abordará la gestión de la deuda técnica junto con sus perfiles de gestión, con un enfoque de mejora continua, lo que posibilita efectuar los cambios y ajustes necesarios de una manera orgánica e incremental. La justificación de esta elección se basa en que la presencia de deuda técnica es de una naturaleza prácticamente ineludible, quienes consideren gestionarla muy probablemente ya hayan percibido sus consecuencias y, como es esperable, no puedan detener la operación del negocio en el día a día a efectos de remediar estos problemas. Como resultado, una estrategia de solución gradual suele ser una alternativa viable al momento de exponer la problemática para concientizar a los interesados.

Como modelo de mejora continua se adoptará el *Círculo de Deming*, también conocido como PDCA (del inglés *Plan-Do-Check-Act*),[1] ilustrado en la Figura 4.3. El Diccionario AMA de Negocios y Gestión define PDCA como “una técnica de supervición de cuatro etapas para la mejora continua de procedimientos de trabajo. El *plan* comprende definir metas, el *do* comprende la práctica, *check* comprende el control de retroalimentación, y *act* sugiere modificaciones a la luz de la experiencia” (Kurian, 2013).

Se adoptará el modelo PDCA por su simplicidad conceptual, no obstante existen otros enfoques a la mejora continua. Un ejemplo es el modelo

[1] En castellano:Planificar-Hacer-Chequear-Actuar.

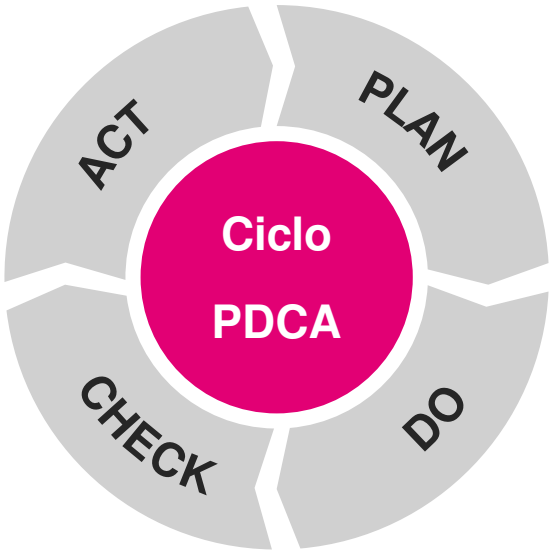


Figura 4.3: Las cuatro fases del modelo PDCA de mejora continua.

de siete fases de COBIT 5 (ISACA, 2012b, 2012a), el cual es una extensión del modelo PDCA y tiene la particularidad de estar desagregado en tres ejes enfatizando la importancia no solo de las actividades para alcanzar el objetivo particular (Ciclo de vida de mejora continua), sino también las inherentes al comportamiento, relación y responsabilidades entre los interesados (Habilitación del cambio) así como la definición y seguimiento de objetivos de control que evidencien que el plan se esté llevando a cabo exitosamente (Gestión del programa).

Notese que el alcance de la presente discusión sobre la mejora continua, no hace referencia a las etapas donde se concientiza a la Dirección de la empresa para establecer el programa, ni la licitación o compra de las herramientas que más se ajusten a las necesidades técnicas.

4.4.1. Planificar

En la etapa de planificación se evalúa la situación actual, se establecen los objetivos a alcanzar durante el ciclo de mejora, y se determinan las brechas que hay que acercar para lograr esos objetivos. Asimismo, se acuerda y establece la forma en que se va a medir el éxito de la gestión en lo relativo al cumplimiento de los objetivos. A continuación se describen las actividades

que habría que realizar para planificar el ciclo.

Establecer la visión estratégica. Como se describió en el Capítulo 3, es necesario acordar con los interesados (CIO, gerentes, directores, etc.) la visión estratégica de la deuda técnica de los sistemas de información dentro de la empresa, analizando distintos modelos y herramientas de ayuda a la toma de decisiones, en conjunción con información del contexto del negocio (por ej. encuestas de satisfacción, competidores, análisis FODA, análisis de 5 fuerzas, etc.). El resultado será un listado de unidades de negocio junto con una idea del perfil de deuda técnica (Sección 3.1) que se debería esperar en cada uno de ellos.

Identificación de elementos. Particularmente en la primer iteración del ciclo de mejora, conviene hacer una valoración, para todas las unidades de negocio, de todos los elementos o artefactos que posean deuda técnica, típicamente consultando con los equipos de trabajo. En esta instancia no se tiene mayor información sobre la cantidad de deuda, Capital u otras mediciones. Sin embargo, de poder utilizar herramientas que analicen el código fuente de los sistemas, posiblemente también arroje no solo los items o artefactos con deuda, sino también una priorización y el Capital (en términos monetarios o de esfuerzo).

Establecer los perfiles de gestión. Una vez identificados y valorizados los elementos con deuda técnica, y en base al perfil de deuda identificado luego de analizar la visión estratégica, por cada unidad de negocios se establece un perfil de gestión (Sección 4.2). Asimismo, se definirán los umbrales o cotas superiores de deuda técnica permitidos.

Establecer la duración del ciclo. La duración del ciclo de mejora es una variable importante. Mientras que una duración muy extensa puede hacer que la gente pierda el impulso o el interés se disperse, una duración muy corta podría no ser suficiente para mostrar mejoras significativas. El autor recomienda ciclos que duren unos pocos meses. Si la metodología de desarrollo es tal que

se realizan entregas en iteraciones de tiempo fijo (*time-boxed*), por ej. cada 2 semanas, conviene redondear el ciclo a un múltiplo de dichas iteraciones.

Establecer indicadores. Los indicadores miden el desempeño de la gestión hacia la consecución de los objetivos planteados.

Además de los relacionados a la deuda técnica, cada unidad de negocios deberá proponer los indicadores que mejor se ajusten a sus objetivos. Por ej. si un sistema crítico para una unidad de negocios tiene problemas de calidad (usuarios reportan una gran cantidad de defectos), entonces deben agregarse indicadores de calidad externos (como la cantidad de defectos reportados en el ciclo) a efectos de poder evaluar *a posteriori* si esos defectos son efectivamente consecuencia de la deuda técnica, o bien es atribuible a otras fuentes (ej. requerimientos mal relevados, programadores no entrenados, etc.).

Si se adopta algún método específico de gestión de deuda técnica, o herramientas disponibles en el mercado, es aconsejable aprovechar los indicadores que los acompañan. Por ejemplo, SQALE (un método de gestión) aporta sus propios indicadores e índices en base a las métricas que recolecta (Letouzey, 2016).

A continuación se sugieren algunos indicadores que pueden ser de utilidad de acuerdo al perfil de gestión seleccionado.

Valoración cualitativa. En el caso del perfil de Seguimiento se podría proponer una escala cualitativa para cada uno de los sistemas de esa unidad de negocios, con valoraciones de 1 (no tiene) a 5 (la deuda es intolerable) con respecto a la cantidad de deuda técnica que el equipo cree que tiene. Luego el indicador sería el promedio de esas valoraciones, o (si se quiere ser conservativo) se puede tomar valor el máximo obtenido.

Deuda técnica de Sistema. Se calcula un indicador diferente para cada uno de los sistemas S_i que abarca el ciclo de gestión actual. Específicamente, se calcula el Capital de la deuda técnica para el sistema S_i, denominado DT_i, y mide el costo (en términos monetarios o de esfuerzo) de las

modificaciones necesarias sobre un sistema determinado para alcanzar una calidad ideal. El cálculo de la deuda técnica normalmente es proporcionado por herramientas automatizadas (Capítulo 5). Asimismo, deberían adicionarse problemas que no son detectados por estas herramientas.

Densidad de deuda técnica Determina un ratio entre la Deuda técnica de Sistema, y el Costo de reescribir el sistema (CR), resultando en la fórmula DT_i/CR_i. El Costo de reescribir el sistema se estima contando las LDC y asignando un esfuerzo o costo por línea (ej. 20 minutos/LDC). Altos valores de este indicador sugiere la evaluación de rehacer ese sistema desde cero en vez de mantenerlo. Puede aplicarse a componentes reusables también.

Distancia de brecha. Para el caso de los perfiles de Estabilización y Reducción, donde se define una cota superior de deuda técnica tolerable, se puede incluir un indicador que de una idea de la "distancia" que falta para cerrar la brecha entre los sistemas y umbral. Suponiendo que la unidad de negocio consta de un conjunto de N sistemas, donde cada sistema S_i tiene una deuda técnica DT_i, se toma el sistema con mayor deuda, obteniendo un indicador $Max(DT_i) - Umbral$, que será positivo si se ha superado el umbral y negativo si está en niveles tolerables.

Porcentaje de sistemas sobre el umbral. Como indicador relacionado al anterior, para perfiles de Estabilización o Reducción, se puede establecer qué porcentaje de los N sistemas que componen la unidad de negocio poseen deuda técnica sobrepasando el umbral permitido. Un valor más bajo indica una concentración de deuda técnica en unos pocos sistemas. Se define una función que marca el "exceso" como $E(i) = 1$ si $DT_i - Umbral > 0$ y $E(i) = 0$ en caso contrario, luego el indicador es $\frac{100}{N} \sum_{i=1}^{N} E(i)$.

4.4.2. Hacer

La ejecución del plan se lleva a cabo en el día a día, en el contexto del proceso normal de desarrollo de software. Tanto las actividades que compo-

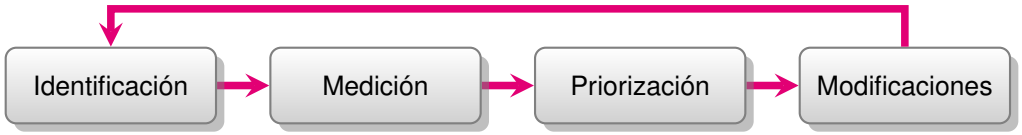


Figura 4.4: Actividades periódicas durante el proceso de desarrollo.

nen la gestión de la deuda técnica (Sección 4.1) como las de soporte, ahora son actividades extras que tiene que realizar el equipo de trabajo.

En el caso de que se tengan que realizar repagos (o remediaciones) puesto que es necesario reducir la deuda en algún artefacto, el tiempo para hacerlo debe estar contemplado en la planificación de las entregas. Cuando requieran la aprobación del cliente o usuario, deben provenir del backlog con su consecuente estimación de esfuerzo de remediación.

En forma periódica el equipo de trabajo tendrá a disposición herramientas para gestionar la deuda técnica que va generando o remediando durante su día laboral. Idealmente las herramientas pueden ejecutarse directamente desde el IDE (*Integrated Development Environment*) o cuando hay modificaciones en el repositorio de código fuente (si se adopta la práctica de integración continua), y cumplen la función de identificar, medir y priorizar los problemas detectados, como se ilustra en la Figura 4.4. En base a esta nueva información un desarrollador realiza las modificaciones pertinentes. Esta realimentación continua es la que permite también tomar medidas de prevención.

4.4.3. Chequear

A medida que va pasando el tiempo en el ciclo de mejora, se deben ir chequeando los indicadores definidos para comprobar que la gestión está dando los resultados esperados.

Para el caso particular del perfil de Seguimiento, el objetivo es obtener una valoración de la evolución de la deuda técnica con el propósito de poder emitir una alerta temprana a los distintos interesados.

En el caso de los perfiles de Estabilización y Reducción, se debe controlar la evolución de los indicadores en cuanto al nivel de deuda técnica detectado, ya sea con herramientas automáticas o bien en formas manuales (como

reuniones de trabajo, revisiones de diseño, etc.).

Asimismo, conviene cotejar los indicadores de deuda técnica con otros relacionados al proceso de desarrollo, con la intención de determinar si existen correlaciones entre, por ej., número de defectos reportados, productividad, cumplimiento de los hitos de entrega, satisfacción de usuarios, etc.

4.4.4. Actuar

En base a los análisis realizados durante el chequeo, y las conclusiones a las que se han llegado, se establecen las correcciones necesarias para mejorar el proceso.

Podría ser que se necesiten mejoras en el proceso de gestión de deuda técnica. No es aconsejable incorporar actividades que luego se realicen rara vez, ya que introduciría deuda técnica en el mismo proceso de gestión de deuda técnica! Lo ideal es que la gestión sea lo más liviana y llevadera que sea posible.

Podría ser que se descubran factores que impacten en una mejora del proceso de desarrollo en general, o que se determine que existen problemas de capacitación del equipo en áreas específicas (típicas: seguridad, arquitecturas, uso de nuevas tecnologías).

En definitiva, las acciones correctivas que pueden tomarse dependen de los resultados obtenidos. Más allá del éxito del ciclo de mejora presente, es fundamental monitorear la deuda técnica remanente en los sistemas, para poder darles visibilidad mediante la comunicación hacia los interesados, y así poder establecer los objetivos del próximo ciclo de mejora de la deuda técnica.

Capítulo 5

Métodos y herramientas

En el presente Capítulo se abordará el tercer y último objetivo planteado en la Sección 1.1, esto es, describir algunos de los métodos y herramientas de gestión de la deuda técnica disponibles en el mercado para sistemas desarrollados con tecnología Java, con la intención de mostrar el estado de arte actual.

En el mercado existen diferentes métodos para estimar y gestionar la deuda técnica, aunque el más conocido es SQALE, tanto por su nivel de adopción como porque su especificación es pública. Se dedicará gran parte de este Capítulo a su descripción. Otros métodos están fuertemente vinculados a herramientas y servicios comerciales por lo que no se tratarán con tanta profundidad.

Por último, la gestión de la deuda técnica requiere en gran medida de herramientas que automaticen su análisis y seguimiento. Se describirán algunas que, por ser de gran calidad, gratuitas y de código abierto, proporcionan una caja de herramientas que es accesible a cualquier organización.

5.1. Método SQALE

El método SQALE adopta su nombre como un acrónimo de *Software Quality Assessment based on Lifecycle Expectations* (Letouzey, 2016), en español Valuación de Calidad basada en Expectativas de ciclo de Vida. Además de definir los cálculos necesarios para estimar la deuda técnica, aporta algu-

(a) Estructura del Modelo de Calidad

(b) Características

Figura 5.1: SQALE: Modelo de Calidad y Características. Extraída de Letouzey (2016).

nos indicadores que sirven como apoyo para guiar la mejora de la calidad del código fuente.

SQALE trata la calidad del código fuente como un requerimiento no funcional, y adopta un modelo de calidad que formula y organiza los requerimientos no funcionales en una forma jerárquica con tres niveles: Características, Sub características y Requerimientos de Código Fuente, de acuerdo a la Figura 5.1a. Esta organización en Características y Sub características es análoga a la provista por el estándar ISO 25010:2011 (ISO/IEC, 2011), con la diferencia que las propiedades a evaluar no son exactamente las mismas. La Figura 5.1b muestra el nivel de Características.

Las Características representan el nivel superior del modelo de calidad. Estas nueve propiedades son relevantes al código fuente e impactan en diversas actividades del ciclo de vida. Asimismo, las Características se muestran como una pirámide indicando que una capa inferior actúa de cimiento de una capa superior, esto es, cronológicamente, es necesario que el código sea testeable (Testeabilidad) para que luego sea confiable (Confiabilidad).

Aunque la estructura piramidal ofrezca una forma sencilla para entender el modelo, y Letouzey permita adaptarlo eliminando capas de acuerdo a lo que la organización necesite, las Características no pueden cambiar de posición relativa dentro de la pirámide. Así, si la organización necesita tanto seguridad y portabilidad del código fuente, necesariamente habría que adaptarlo en el orden especificado, aún cuando bien podría hacerse en el orden inverso, una

Figura 5.2: Características y Sub Características en SQALE. Extraída de Letouzey (2016).

restricción innecesaria de acuerdo a la opinión de este autor.

El segundo nivel es el de Sub características. Cada Característica comprende una o más Sub características (Figura 5.2), esto se hace como desagregación de distintos conceptos bajo una misma característica. Como muestra la Figura, la testeabilidad esta compuesta por la testeabilidad a nivel de unidad y de integración.

Finalmente el nivel de Requerimientos contiene todos los elementos relacionados a la calidad del código fuente, que se asociarán a una sub característica. Por ej. un requerimiento que relacione la complejidad ciclomática del código (McCabe, 1976) con una subcaracterística: Se puede decir que una alta complejidad impacta en la testeabilidad de los tests unitarios (testeabilidad a nivel de unidad), y también a la Entendibilidad del código (*Understandability*). Aquí, se puede definir un Requerimiento que “el código no contenga secciones de código con una complejidad ciclomática mayor a cierto número”.

5.1.1. Modelo de calidad y Deuda Técnica

El modelo de calidad definido arriba enuncia un conjunto de requerimientos agrupados en Sub características y estos a su vez en Características. Todo tema de calidad que genere deuda técnica debería estar incluido en el

modelo como un Requerimiento, de modo tal que un Requerimiento representa una regla que debe ser obedecida a efectos de no generar deuda.

Asimismo, SQALE representa los resultados de los cálculos en términos de costos (monetarios o esfuerzo). Para esto se utilizan los conceptos de funciones de *remediación* y funciones de *no remediación*, enunciados a continuación.

El propósito de la función de remediación es mapear los resultados encontrados al chequear el Requerimiento en términos de costos de corrección. Estos costos están calculados desde el punto de vista del equipo de trabajo. O sea, el costo de remediación tiene un impacto negativo en el plan del proyecto asumiendo que los problemas encontrados se solucionarán.

En contraste, una función de no remediación mapea una no conformidad de un Requerimiento con el costo de mantener la no conformidad y entregar el software sin las debidas correcciones. El costo se calcula desde el punto de vista del impacto en el negocio (en vez de desde el punto de vista del equipo de desarrollo), y debería considerar todos los costos directos e indirectos generados por la no conformidad, incluyendo: costos de remediación más altos debido a que deben solucionarse en etapas posteriores de desarrollo, costos de mantenimiento más caros, el costo de recursos adicionales (ej. espacio en disco, memoria), el costo del personal extra necesario para operar el software, etc. Se aconseja que en la estimación de estas funciones se convoque a todos los interesados necesarios debido al alcance de los mismos.

5.1.2. Capital e Interés

El conjunto de funciones de remediación constituyen el modelo de estimación del Capital de la deuda técnica, esto es, la suma de todas las no conformidades a los Requerimientos es el Capital, llamado SQI (*SQALE Quality Index*).

Asimismo, el conjunto de funciones de no remediación representa el modelo de estimación para el Interés de la deuda técnica. La suma de las funciones de no remediación arrojan el SBII (*SQALE Business Impact Index*) o Índice de Impacto de Negocio SQALE. Aquí cabe mencionar que SQALE trata

a este índice como el Interés de la deuda técnica, aunque según las definiciones encontradas en la literatura (Sección 2.5) el interés se refiere comunmente al costo extra en que incurrirá el proveedor del software, mientras que SQALE adopta una visión más abarcadora al considerar el impacto al negocio en general, lo que redunda en una métrica más valiosa para la toma de decisiones para el cliente y proveedor en conjunto, una situación que podría observarse y aprovecharse en el caso de sistemas de desarrollo interno a la organización (aunque no tanto en relaciones de *outsourcing*).

5.1.3. Indicadores

SQALE especifica tres indicadores que resumen y sintetizan en una forma sencilla la deuda técnica para una rápida toma de decisiones: el Rating, la Pirámide y el Mapa de Deuda.

Rating SQALE

El Rating SQALE define una escala que otorga categorías según el nivel de deuda técnica que se encuentre en el software, desde la A (muy bueno) hasta la E (muy malo). Para ello se determina el ratio entre la deuda técnica encontrada (SQI) y el costo de reescribir el código desde cero. Esta última métrica se puede estimar basándose en contar las LDC y asignarle un costo en esfuerzo (por ej. 30 minutos por LDC) o monetario (por ej. $100 por LDC). Un ejemplo es el rating del Cuadro 5.1, donde se han definido cinco niveles con rangos.

Pirámide

La Pirámide SQALE muestra todos los índices agrupados por Característica. El índice SQI puede ser desagregado por Categoría en otros índices, útiles para identificar en qué Características hay que hacer énfasis (por ej. si el índice de Security es alto en muchas aplicaciones, tal vez sea indicio de que el equipo de trabajo necesite capacitación en este área).

Rating	Hasta	Color
A	1 %	
B	2 %	
C	4 %	
D	8 %	
E	∞	

Cuadro 5.1: Escala de Ratings SQALE. Extraído de (Letouzey, 2016).

La Figura 5.3 ejemplifica una pirámide. Se representa como una matriz cuadrada de doble entrada, ordenada con el mismo criterio cronológico usado para las Características. El valor de cada índice se va repitiendo a lo largo de toda la fila, comenzando por la columna que corresponde a la diagonal. Abajo, se suman los índices a modo de resumen, con la última sumatoria (abajo a la derecha) representando el SQI.

Mapa de Deuda

El Mapa de Deuda de SQALE (Figura 5.4) ubica un número de artefactos (archivos, componentes, aplicaciones, etc.) con relación a su deuda técnica (SQI en el eje X) y su interés o Impacto en el Negocio (SBII en el eje Y). Este gráfico asiste a la estrategia de remediación, ya que ayuda a visualizar fácilmente los artefactos que más impacto en el negocio producen. Por ej. podría decidirse reducir la deuda técnica de los artefactos con mayor impacto al negocio pero que requieran menos tiempo de desarrollo, es decir, poner énfasis en el cuadrante superior izquierdo del gráfico (Letouzey y Ilkiewicz, 2012).

5.2. Otros métodos

Además de SQALE, otros métodos de acceso público, que comprendan una especificación formal que pueda usarse para implementarse en una herramienta no son conocidos por el autor. Existen herramientas comercia-

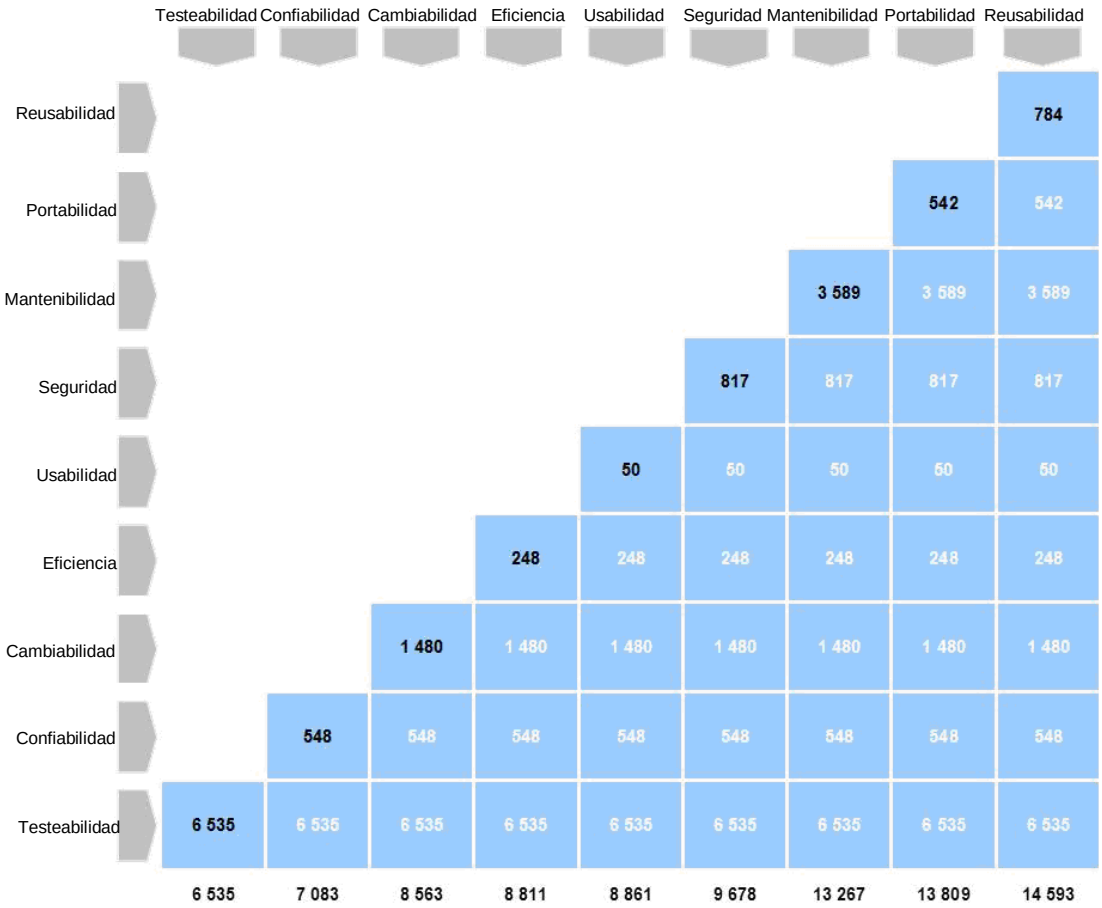


Figura 5.3: Pirámide SQALE. Extraída de Letouzey (2016).

les y servicios que implementan su método propietario. Aunque se conocen algunos detalles de estos métodos, otros detalles permanecen ocultos.

5.2.1. CAST AIP

La herramienta *CAST Application Intelligence Platform*[1] utiliza un método con una manera similar de calcular la deuda técnica a SQALE (Curtis y cols., 2012). La herramienta calcula un indicador TQI (*Total Quality Index*) en base a cinco métricas: (1) Robustez, (2) Performance, (3) Seguridad, (4) Transferenciabilidad y (5) Cambiabilidad (Sappidi y cols., 2012). Al momento de escritura de este documento, el autor no ha podido tener acceso a una versión de prueba de la herramienta para su evaluación.

[1] `http://www.castsoftware.com/products/application-intelligence-platform-system-level-software-analysis`

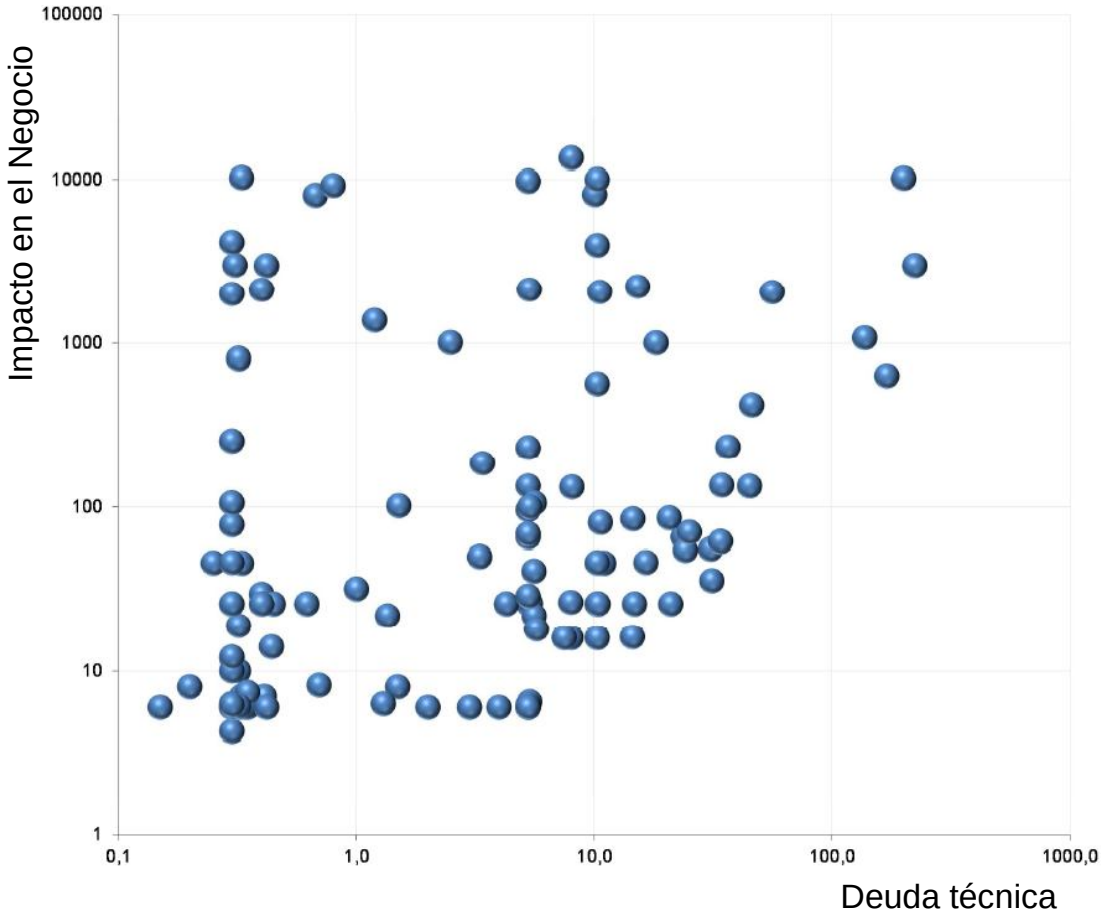


Figura 5.4: Mapa de Deuda SQALE. Extraída de Letouzey (2016).

5.2.2. SIG/TÜViT

Otro método de análisis de deuda técnica es el de SIG/TÜViT[2]. La empresa TÜViT ofrece un servicio en base a un método de evaluación de la calidad del código fuente desarrollado en conjunción con SIG[3] (Baggen, Correia, Schill, y Visser, 2012). El cliente debe firmar un acuerdo de confidencialidad, y en base a herramientas semi-automáticas, la empresa ofrece recomendaciones de mejora. Al momento de escritura de este documento, el autor no ha podido tener acceso a un informe de mejoras. Sin embargo, se sabe que el método se basa en los cálculos de deuda técnica e interés de Nugroho y cols. (2011).

El método califica el sistema con un rating de 1 a 5 con respecto a su

[2] https://www.tuvit.de/en/products/maintainability-1215.htm

[3] https://www.sig.eu/

mantenibilidad, donde 1 es el peor puntaje y 5 el mejor.

Por último, el método está basado en una base de datos propietaria que debe ser "calibrada" anualmente para mantener la calidad de sus resultados, y esta base de datos nunca ha sido de acceso público, razón por la cual este servicio ha sido criticado.

5.3. Herramientas

En lo relativo a las herramientas disponibles en el mercado que asisten en la gestión de la deuda técnica en sistemas de información desarrollados en Java, existe una amplia variedad, cubriendo diversos propósitos.

Las herramientas cumplen un rol fundamental, sin lo cual la gestión de la deuda técnica sería inviable, puesto que permiten automatizar gran parte del análisis efectuado sobre el código fuente, de manera que resulte efectivo en costo y realizado en una forma rápida y eficiente, lo que permite obtener un feedback casi en forma instantánea sobre la evolución de los atributos internos del sistema.

Se concentrará la atención en herramientas de acceso público y gratuito. Esto último es relevante porque evidencia la existencia de un mercado abierto y maduro, y quizás más importante: *evidencian un costo de entrada muy bajo para comenzar a gestionar la deuda técnica*.

5.3.1. Sonarqube

Sonarqube[4] es una de las herramientas más completas del mercado, ya que actúa como un servicio web que integra otras herramientas más especializadas en un aspecto de la calidad interna en particular. Sonarqube es capaz de analizar código fuente de los lenguajes de programación más populares y tiene una arquitectura extensible a la cual se puede aumentar su funcionalidad por medio de *plugins*. A criterio del autor, lo más interesante es que provee una visión integral de la deuda técnica de un software, actuando

[4] `http://www.sonarqube.org/`

de tablero de comando mostrando distintos indicadores, por ej. los mostrados en la Figura 5.5:

Porcentaje de cobertura. Muestra qué porcentaje del código fuente fue ejercitado con tests.

Porcentaje de código duplicado. El porcentaje de código fuente duplicado dentro del mismo proyecto.

Defectos y vulnerabilidades. La cantidad de errores y vulnerabilidades de seguridad detectados.

Cambios en últimos 30 días. Permite visualizar diferencias en los indicadores con una ventana de 30 días.

Línea de tiempo. Muestra la evolución de cada indicador con un gráfico para una interpretación más intuitiva.

Las medidas tomadas en cada aspecto gestionado por la herramienta también son resumidas en un informe (Figura 5.6). Asimismo, el análisis del código fuente puede ser disparado tanto desde el IDE como desde Maven y servidores de integración continua.

5.3.2. Cobertura y JaCoCo

Cobertura[5] y JaCoCo[6] son herramientas que permiten conocer el porcentaje de código fuente que es ejercitado por tests, lo que es un indicador de la "testeabilidad" del sistema, o cuan fácil de testear es la funcionalidad del mismo. Como resultado, un bajo nivel de cobertura puede indicar cuan propenso es el sistema a introducir defectos a medida que va evolucionando. Pueden ser llamados directamente por Sonarqube o en forma independiente desde herramientas de construcción.

[5] `http://cobertura.github.io/cobertura/`

[6] `http://www.eclemma.org/jacoco/`

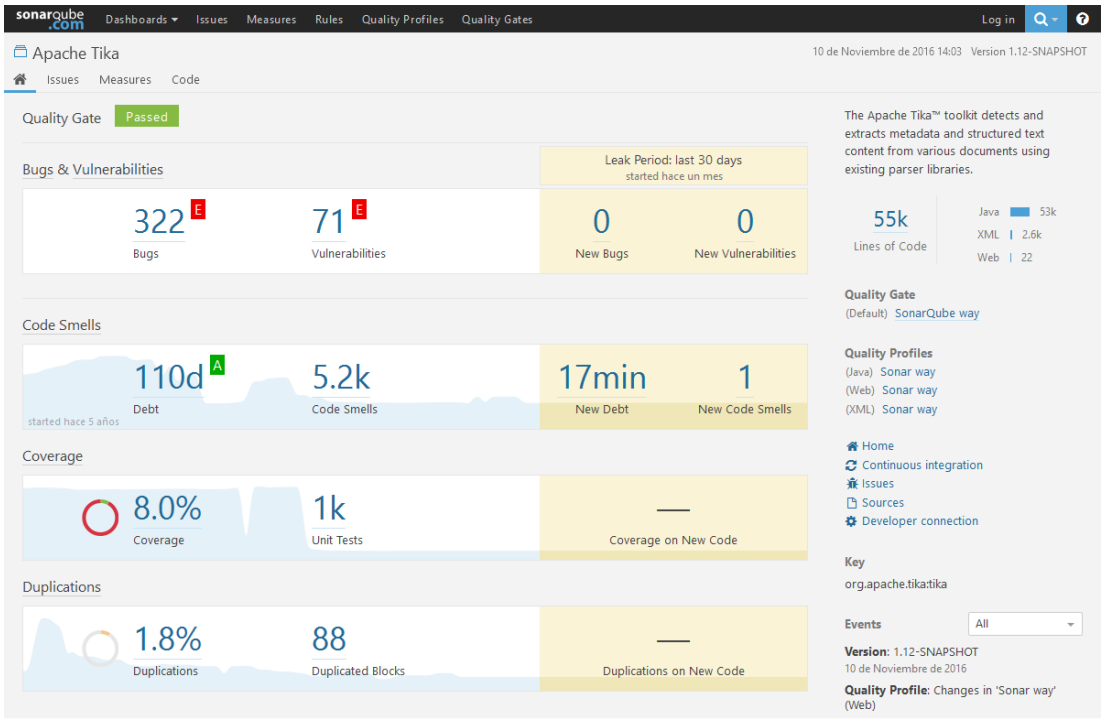


Figura 5.5: Herramienta Sonarqube, mostrando indicadores para el proyecto Apache Tika.

5.3.3. PMD

La herramienta PMD[7] es un analizador de código que detecta defectos y *code smells* para varios lenguajes de programación. Opera sobre el código fuente del sistema, chequeando si existen violaciones a reglas pre-definidas de buenas prácticas, o defectos conocidos.

Asimismo, estas reglas son configurables en sus umbrales a efectos de que no generen alertas innecesarias en caso de que el equipo de trabajo considere que alguna de ellas no aplique a su contexto particular. Está integrada a Sonarqube, aunque puede llamarse en forma independiente tanto desde el IDE como de otras herramientas de construcción como Maven.

[7] `https://pmd.github.io/`

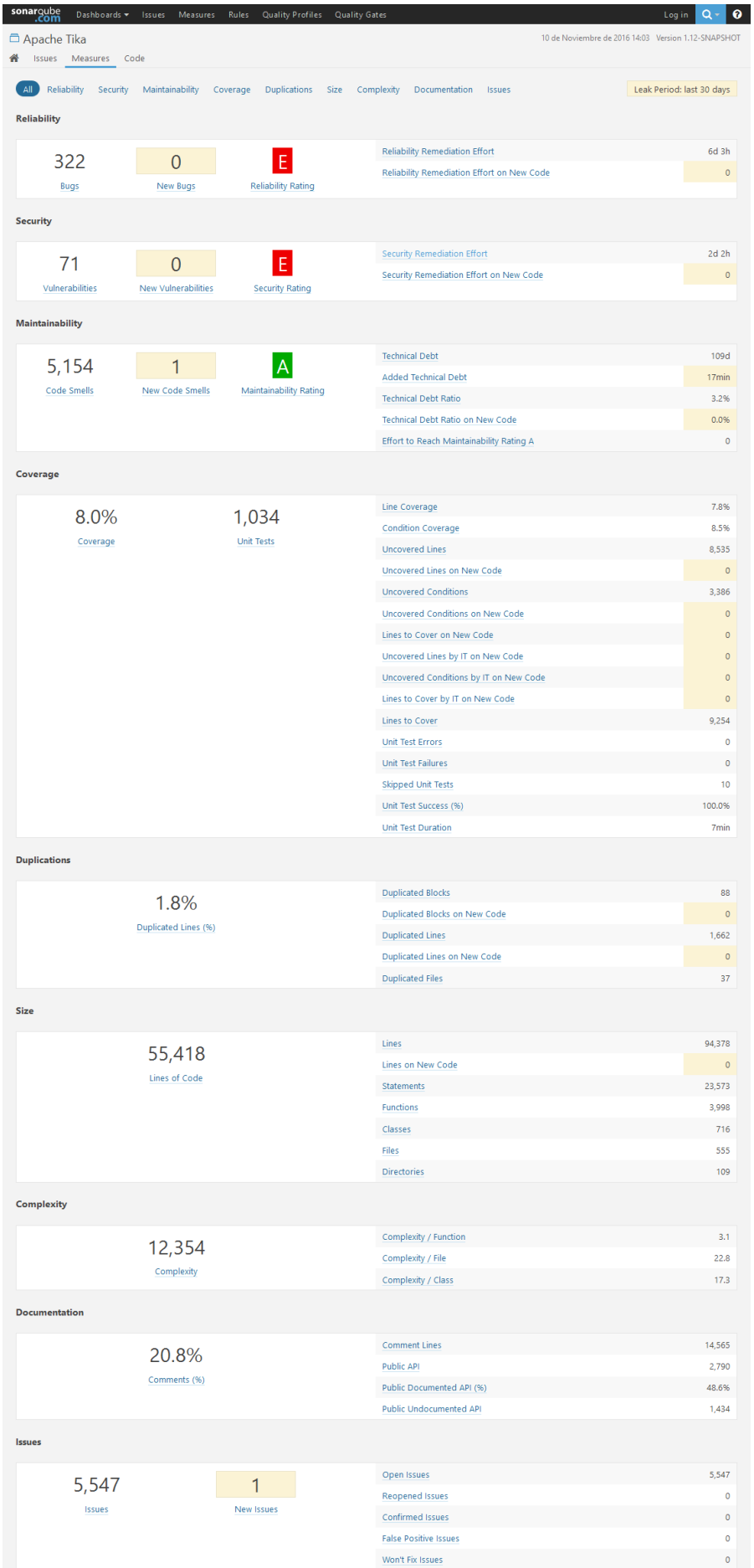


Figura 5.6: Sonarqube, mostrando mediciones para el proyecto Apache Tika.

5.3.4. Findbugs

Findbugs[8] es otro analizador de código configurable con reglas, pero a diferencia de PMD que realiza su análisis sobre el código fuente, Findbugs opera sobre el *bytecode*, esto es, el código ya compilado, por lo que suele complementar eficazmente la detección de defectos. También está integrada a Sonarqube y puede invocarse de forma independiente.

5.3.5. Checkstyle

Checkstyle[9] es una herramienta que chequea si el código fuente adhiere a un estándar de codificación. Por ej. verifica si los espacios de indentación entre sentencias anidadas son los correctos, si las llaves que abren ({) y cierran (}) bloques de código están en un renglón diferente o en el mismo renglón que la sentencia `while`, etc. La adherencia a un estándar de codificación es importante porque permite al equipo de trabajo leer el código fuente sin sentirse agobiado por el formateo del mismo, además que contribuye a apropiar colectivamente el trabajo (en contrapartida con apropiarse solo del trabajo que hizo un miembro del equipo individual).

Está integrada a Sonarqube y también se puede ejecutar independientemente. Asimismo, existen otras herramientas que formatean el código, llamadas *beautifiers* que corrigen automáticamente el código fuente para conformar con un estándar definido.

5.3.6. Maven

Maven[10] es una herramienta de construcción y gestión de proyectos de propósito general, basada en la ejecución de *plugins* para realizar distintas tareas, como construir el proyecto de software, ejecutar pruebas o generar automáticamente un sitio web con información del proyecto. Existen *plugins* para cada una de las herramientas nombradas en este capítulo, por lo que su

[8] `http://findbugs.sourceforge.net/`

[9] `http://checkstyle.sourceforge.net/`

[10] `http://maven.apache.org/`

ejecución puede dispararse a voluntad del desarrollador o en el contexto de un servidor de integración continua.

Una de las características más importantes de Maven es que explícitamente permite gestionar las dependencias del sistema con los componentes de software que se utilizan para su construcción, ejecución y testing. Esto es una funcionalidad muy útil ya que la gestión de dependencias automatiza la trazabilidad entre elementos de software, simplificando la tarea de realizar un análisis de impacto. Por ej. si un componente reusado por varios sistemas presenta una vulnerabilidad de seguridad, desde Sonarqube se pueden buscar todos los sistemas que usan esa versión particular del componente, con lo que puede evaluarse el impacto del cambio.

Capítulo 6

Conclusiones

A lo largo de esta obra se ha descrito el fenómeno de la deuda técnica y su gestión, tanto en términos generales como específicamente en los sistemas de información desarrollados en Java.

A continuación se presentan conclusiones, recomendaciones y reflexiones finales en relación a los aspectos investigados y aportes realizados.

6.1. Concientización

La deuda técnica es un **fenómeno real**, pero no es un fenómeno nuevo, ya en la década del ‘70 las leyes de Lehman hablaban del efecto “declinante” en la calidad que producen los cambios continuos en sistemas, que incrementan su complejidad a menos que se realicen esfuerzos que la compensen. Un síntoma de la presencia de deuda técnica, aunque en su lenguaje técnico y por lo tanto enfocado solo a aquellas audiencias, lo que impedía el acceso del concepto al personal jerárquico no especializado en desarrollo de software. No solo ha existido desde hace varias décadas, sino que posiblemente sus síntomas se hayan incrementado con el paso del tiempo, puesto que la complejidad de los sistemas y sus integraciones no ha disminuido.

La deuda técnica impacta directamente sobre los atributos internos de los sistemas, e indirectamente sobre sus atributos externos, por lo que su visibilidad directa es percibida por el equipo de trabajo, aunque rara vez comunicada. Aún es común encontrar cierto nivel de inmadurez entre los res-

ponsables de la gestión de TI que no la reconocen, o no la exponen de cara a los interesados de la empresa por temor a que sus gestiones se vean cuestionadas.

El primer paso para resolver el problema es que los equipos de trabajo sean capaces de reconocerla, de esta manera puede ser gestionada de forma acorde a los objetivos estratégicos de la empresa. La deuda técnica que se genera en forma inadvertida debe ser minimizada, una forma de lograrlo es capacitando al equipo. La que se genera en forma intencional es la que apalanca los objetivos de negocio, por lo que puede (y debe) gestionarse como cualquier otro recurso.

Como muestran las encuestas relevadas en esta obra, **existe una falta de concientización** por parte de los interesados de la organización, por lo tanto esta es una actividad esencial para comenzar a darle visibilidad y así poder adquirir los recursos para gestionarla efectivamente. A este fin, la metáfora financiera del Capital y el Interés representan un punto de partida para explicar los conceptos hacia nuevas audiencias, y que explican el fenómeno en términos menos técnicos.

6.2. Utilidad del interés

La metáfora de la deuda técnica, compuesta por el Capital (el esfuerzo que es necesario realizar hasta alcanzar una calidad ideal) y el Interés (el esfuerzo extra que es necesario realizar por no haber alcanzado esa calidad ideal), sirve perfectamente para introducir a una audiencia en el tema. Las herramientas que calculan Deuda Técnica, en realidad están estimando el Capital, puesto que es lo único que pueden estimar con cierta confianza.

El Interés depende de los requerimientos que necesitan desarrollarse y su priorización. Las modificaciones que es necesario realizar al sistema para desarrollar uno de estos requerimientos pueden implicar una modificación en un sector o módulo del sistema con alta deuda técnica, lo que llevaría acarreado un alto interés, pero también puede impactar un módulo con una baja deuda, lo que acarrearía un bajo interés. Entonces, para sacar una proyec-

ción a largo plazo y poder realizar una decisión del tipo “pagar capital” vs. “no pagar capital pero pagar intereses”, estos requerimientos deben cumplir: (1) estar estimados en esfuerzo, (2) estar priorizados con prioridades estables, y (3) las dos condiciones anteriores deben valer durante la extensión del cronograma en que se desee predecir el interés.

Como las anteriores condiciones son muy difíciles de cumplir por un largo periodo de tiempo (¿años?, ¿meses?), la estimación del interés se reduce a una cifra que es difícil de sostener, por lo tanto, en la práctica, el interés de la deuda es de menos utilidad que el capital.

La regla general recomendada por el autor es: en sistemas que están en constante evolución, poner atención en el capital (seguirlo, estabilizarlo o reducirlo) e ignorar el interés. No obstante, existen dos excepciones a esta regla: (1) si el software está cerca del fin de su ciclo de vida (por ej. sistemas *legacy*), en cuyo caso se puede evaluar solo el pagar intereses para no arriesgar la desestabilización de funcionalidad existente, y (2) cuando el sistema solo evoluciona en modo correctivo (solución de defectos, vulnerabilidades de seguridad) haciendo entregas en períodos de tiempo prefijados, en cuyo caso puede predecirse el interés en forma más confiable (por ej. actualizaciones de parches de *Windows Update*).

6.3. Visión estratégica

Para dar una gestión apropiada, es necesario ubicarse en un nivel macro y plantearse una visión del aporte de la gestión de la deuda técnica a los objetivos estratégicos de la empresa. Para ello, el CIO debe estar empapado en el contexto, mercado y situación particular por la que esté pasando la organización para poder guiar la racionalización de recursos a ser asignados a este propósito.

Encuadrar la empresa dentro de los aspectos de gestión de TI que tienen un impacto en la deuda técnica ayuda de dos formas: por un lado, brinda una idea del perfil de deuda técnica que puede esperarse en la empresa o unidades de negocio particulares. Por otro lado, estos aspectos ayudan a definir

un perfil de gestión de la deuda técnica para que ésta converja hacia niveles más adecuados.

Aún si la gestión de la deuda técnica no es un problema mayor dentro de la empresa, evaluar modelos de gestión como los relevados en el Capítulo 3 proveen información fundamental sobre la forma que adoptará una gestión de TI en términos generales.

6.4. Gestión

Todas las etapas del proceso de desarrollo de software son permeables a la deuda técnica. Ninguna es inmune, y la gestión de la deuda técnica no es la excepción. El **proceso de gestión de la deuda técnica debe ser lo más simple posible**. La razón es doble: por un lado no debe afectar significativamente la capacidad de producción de la función de TI, cuyo objetivo principal es la generación de valor mediante la provisión de funcionalidades de software, y por otro lado, no es deseable que el propio proceso de gestión de deuda técnica acumule deuda técnica (dejando actividades sin realizar).

Este trabajo aporta un sencillo marco teórico para gestión de deuda técnica, basado en tres perfiles de gestión: Seguimiento, Estabilización y Reducción. Cada uno de ellos es de utilidad para alcanzar diferentes objetivos de negocio. Estos perfiles simplifican la toma de decisiones a nivel gerencial, ya que vinculan el análisis de Visión estratégica enunciado arriba con la práctica diaria, identificando y agrupando un conjunto de actividades para alcanzar esos objetivos. Se incentiva al lector a evaluar y ejecutar la mínima cantidad de actividades de gestión que aporten positivamente hacia los niveles deseados.

Asimismo, el marco teórico de perfiles de gestión establece diferentes estrategias que se diferencian por el impacto concreto en la toma de decisiones: cuando se hace Seguimiento solo se observa la evolución de la deuda, en Estabilización se requiere intervenir para mantenerla en los niveles actuales, mientras que en el de Reducción no solo se requiere intervenir, sino que se necesita acordar con los interesados para planificar y priorizar el repago o remediación.

6.5. El rol de la automatización

La gestión de la deuda técnica en sistemas de información sería inviable si no fuese por las herramientas. Servicios como Sonarqube asisten en la automatización de las actividades de Identificación, Medición, Priorización, Monitoreo, Documentación y Trazabilidad de la deuda técnica, esto es, todas las actividades identificadas para gestionarla a excepción del Repago y Prevención, que requieren el esfuerzo explícito del equipo de trabajo para modificar el sistema.

El autor recomienda iniciar la gestión de la deuda técnica con las herramientas ya instaladas: no solo simplificará enormemente el trabajo diario, además las herramientas específicas para el lenguaje Java (las descriptas en esta obra) son de muy alta calidad, ampliamente adoptadas por la comunidad de desarrolladores, gratuitas y de código abierto. Por el bajo costo, escasos requerimientos de *hardware* y simplicidad en su uso, la curva de aprendizaje es realmente corta y la inversión muy baja. En experiencia del autor, 10 días de trabajo realizado por una persona son más que suficientes para instalar y configurar estas herramientas.

En definitiva, se recomienda descansar en la información arrojada por las herramientas para realizar gran parte de la gestión. Sin embargo, no toda la deuda técnica es detectable automáticamente, el caso más importante es el de la deuda sobre la arquitectura de sistemas.

6.6. Arquitectura

La arquitectura de software es una de las fuentes de deuda técnica más problemática, por diversos motivos.

Por un lado, la arquitectura es uno de los aspectos internos del software que tienen un gran impacto en la estructura y comportamiento del sistema, por lo que cambios realizados sobre ella tienden a repercutir sobre gran parte de la base de código del mismo. En otras palabras, modificar la arquitectura requiere un esfuerzo considerable.

Además, el proceso de diseño de la arquitectura es una de las actividades que se suele realizar prematuramente en relación al cronograma de desarrollo, por lo que no siempre se diseña conociendo todos los requerimientos funcionales y no funcionales que debe soportar. A medida que el sistema evoluciona, la arquitectura ideal comienza a desalinearse con la definida inicialmente.

Por último, pero no menos importante, las herramientas que hoy se pueden encontrar en el mercado no tienen la capacidad de detectar las divergencias entre la arquitectura actual y la ideal, por lo que se requiere invertir esfuerzo del equipo para realizar este análisis. Es normal tener que acordar con los interesados del sistema la planificación de refactoreos para alinear la arquitectura hacia una estructura más acorde a las necesidades actuales (es decir, adoptar un perfil de Reducción) por el alto riesgo que conlleva.

Se recomienda comenzar a considerar la arquitectura en la gestión de la deuda técnica una vez que el equipo de trabajo se encuentre cómodo con el proceso de gestión establecido, ya se encuentre capacitado y con cierto grado de experiencia en el uso de las herramientas. Debe considerarse que la arquitectura implica más trabajo de gestión manual puesto que no es automatizable.

6.7. Un proceso de mejora emergente

El proceso de gestión de deuda técnica arroja evidencia concreta sobre las deficiencias de los sistemas. Un número concreto en términos de esfuerzo o costo como el Capital de la deuda y otros indicadores de gestión propuestos, junto con los otros indicadores que ya tenga definido el CIO para gestionar la función de TI, aportan una visión realista sobre la operación diaria del sector.

Con una buena gestión, las deficiencias de los sistemas solo son tolerables si la deuda ha sido adquirida en forma intencional, caso contrario la deuda técnica se transforma en un síntoma de otros problemas, los cuales deben analizarse sus causas. En este sentido los indicadores arrojados por el proceso de gestión de deuda técnica pueden aportar datos que permitan

sacar conclusiones.

Por ej. si existen muchas violaciones que tienen que ver con vulnerabilidades de seguridad, puede ser indicio que el equipo de trabajo carezca de la capacitación adecuada, asimismo el equipo de SQA no está poniendo a prueba temas específicos de seguridad, lo que termina impactando sobre el proceso de desarrollo de software.

Como ejemplo adicional, si existen problemas recurrentes de arquitectura, tal vez sea necesario enfocar futuras inversiones en este área. En el caso de una PyME, cuyo proceso de desarrollo carezca de proceso de definición de arquitectura de software, los hallazgos encontrados pueden servir como evidencia de que es necesario instalar dicho proceso, o bien establecer una práctica de Arquitectura Empresarial.

En otras palabras, la información de deuda técnica ofrece evidencia concreta y contundente que puede analizarse y presentarse hacia los niveles directivos como fundamento para promover mejoras en el proceso de desarrollo, o bien en la gestión de TI en general. Como tal, es un proceso que habilita y ayuda a las empresas a mejorar sus prácticas, y elevar sus niveles de madurez. El proceso de gestión de deuda técnica funciona en forma emergente para soportar otros procesos de la gestión de TI, esto es, aporta un enfoque *bottom-up* (de abajo hacia arriba) donde la evidencia y equipos de trabajo proveen la información necesaria para encausar la gestión de TI en el cumplimiento de los objetivos y estrategias de la empresa.

Apéndice A

Estimación de la deuda técnica

Como se describió en el Capítulo 2, la deuda técnica esta compuesta por el Capital y el Interés, de la misma forma en que existe en una deuda financiera. Pero a diferencia de una deuda financiera, donde la deuda, capital o interés pueden deducirse a través de una ecuación matemática, la deuda técnica no puede cuantificarse tan fácilmente. Sin embargo, aún así es necesario tener alguna forma de estimar su magnitud, ya sea cualitativa o cuantitativamente, a efectos de poder gestionarla adecuadamente.

En este Apéndice se introducirá al lector sobre técnicas de estimación del Capital (la brecha entre la calidad actual versus la "calidad ideal"), el Interés (el esfuerzo extra debido a no invertir en solucionar los problemas de calidad), y una forma en que puede determinarse un "punto de equilibrio", esto es, una estimación del momento en el tiempo en el cual se alcanza el mismo costo para la decisión de "pagar deuda versus pagar intereses".

A.1. Calidad ideal

Entendiendo que para estimar la deuda técnica se necesita determinar la magnitud de la brecha que existe entre la calidad actual y la calidad ideal, lo primero por establecer son las características que debe tener el software para ser considerado con una calidad "ideal", de forma tal de poder inferir su calidad "actual".

Es claro que la calidad ideal es un concepto que no es un absoluto, sino

que representa los niveles óptimos de calidad que se encuadren dentro de los requisitos de negocio de la organización, o inclusive de un proyecto o software en particular. Así, una organización que desarrolle software de control de equipamiento médico necesitará obtener altos niveles de confiabilidad y certificar su proceso de desarrollo y el producto con normas internacionales, mientras que la misma organización seguramente no requiera los mismos niveles de confiabilidad para su software de control de presentismo para su personal.

En la práctica, se pueden establecer umbrales o *thresholds* en KPIs (*Key Performance Indicators*) o métricas cuyo cumplimiento indica un grado (alto o) ideal de calidad, mientras que el traspaso de ese umbral evidencia una brecha entre la calidad actual y la ideal.

Cuando no se llega al umbral deseado, es importante para una correcta gestión que sea posible estimar el esfuerzo, costo o tiempo extra necesario para alcanzarlo. Incluso cuando ocurre que se llega al umbral deseado, es interesante poder determinar la magnitud en el cual se excede del umbral ya que los recursos asignados a tal efecto pueden redistribuirse en otras actividades más prioritarias.

Por ejemplo, si quiere medirse la testeabilidad del software, se puede medir el porcentaje de cobertura de tests sobre el código fuente: un porcentaje de 100 % indica que todo el código fuente tiene tests asociados que ponen a prueba cada una de sus funcionalidades, mientras que un 0 % indica que no hay tests asociados a ninguna sección de código. Existen herramientas como Cobertura (Cobertura, s.f.) que calculan esta métrica en forma automatizada. En este sentido, para un proyecto podría especificarse que el porcentaje de cobertura mínimo admitido es del 70 %, definiendo así una característica de calidad interna que el software debe cumplir como parte de los temas que establecen la calidad ideal.

Así, la calidad ideal del software quedará establecida por un conjunto de métricas o KPIs que deben cumplirse. El no cumplimiento de una de esas métricas genera una “violación” de esa métrica, esto es, una brecha que formará parte del Capital.

Además, es sensato asumir que no todas las violaciones a las métri-

Severidad	Descripción
BLOQUEANTE	La presencia de este problema implica un riesgo inadmisible.
CRÍTICA	La presencia de este problema implica un muy alto riesgo.
MAYOR	La presencia de este problema implica un alto riesgo.
MENOR	La presencia de este problema implica un riesgo moderado.
INFORMACIÓN	Este problema implica un bajo riesgo.

Cuadro A.1: Ejemplo de escala cualitativa de severidades.

cas tienen la misma importancia o "severidad". Como ejemplo, la detección de un *memory leak* es un tema más importante de resolver que la correcta indentación del código fuente. Es común clasificar las severidades en forma cualitativa. El Cuadro A.1 ejemplifica una clasificación posible.

Por último, otra cuestión a tener en cuenta es la "prioridad" (o urgencia) de la violación encontrada, lo cual es dependiente del contexto actual. Como para cualquier gestión, es preciso poder diferenciar claramente lo importante (severidad) de lo urgente (prioridad).

A.2. Medición versus Estimación

En el mundo financiero, una deuda producida por un préstamo, el capital y el interés (que depende de una tasa de interés) pueden ser calculadas en forma exacta a partir de fórmulas matemáticas. Sin embargo, en la metáfora de la deuda técnica tal fórmula no es aplicable para obtener una medida o magnitud concreta y única. La razón es que la deuda técnica se deduce a partir de contar todas las brechas de calidad *encontradas* entre la calidad actual y la ideal. Esta es la razón por la que se puede estimar, pero no medir.

El problema es que el proceso es imperfecto ya que las brechas deben ser buscadas, detectadas para luego ser registradas y contadas. Puesto que el proceso de detección requiere de un análisis detallado del código fuente, y éste puede comprender millones de LDCs, en la manera en que sea posible este proceso debe automatizarse, caso contrario tendría un costo inviable.

Asimismo, existe deuda técnica cuya detección es más confiable si proviene del análisis realizado por un experto que por medios automáticos, tal es el caso de la deuda de arquitectura o de diseño, o la deuda de documentación. En general, las herramientas de automatización son muy buenas detectando cosas que "son" pero no son buenas detectando las cosas que "deberían ser": una persona puede discernir más confiablemente las intenciones originales de un desarrollador al escribir un algoritmo, mientras que una herramienta es más confiable para detectar problemas en el resultado final.

Lo anterior impacta en la precisión de la estimación, puesto que la detección puede producir "falsos positivos" (cuando se detecta un problema que en realidad no lo es) y "falsos negativos" (cuando un problema real no fue detectado).

A.3. Estimación del Capital

Curtis, Sappidi, y Szynkarski (2012); Letouzey (2016). De acuerdo a la definición de estos autores (Sección 2.5) es habitual encontrar en la práctica que la calidad ideal se ha definido como un conjunto de reglas discretas que representan violaciones a alguna característica particular, donde cada regla tiene asociada un determinado esfuerzo de reparación. Así, la calidad es ideal cuando no se presenta ninguna de esas violaciones. Formalmente se puede definir un conjunto $V = \{v_1, v_2, ..., v_M\}$ con M violaciones consideradas, donde cada violación v_i tiene asociado un esfuerzo de reparación de e_i horas-hombre.

Para estimar el Capital (o la cantidad de Deuda Técnica) se supondrá que se ha analizado el software y se han detectado diferentes tipos de violaciones, donde posiblemente la misma violación ha aparecido en distintas partes del software. Se supondrá que cada violación v_i se ha encontrado un número $n_i \geq 0$ de veces.

Si además se conoce el valor por hora (p_i) del desarrollador que debe solucionar una violación v_i, se puede estimar el capital en términos de costo

($) con la siguiente fórmula:

$$Capital = \sum_{i=1}^{M} n_i \times e_i \times p_i \tag{A.1}$$

Si se supone que el precio por hora p de trabajo es único para todas las violaciones (por ei. si se usa el promedio entre todos los desarrolladores), la estimación puede simplificarse a:

$$Capital = p \times \sum_{i=1}^{M} n_i \times e_i$$

A su vez, puede desagregarse el Capital de acuerdo a la severidad de las violaciones encontradas. En el caso del método de Curtis y cols. (2012) se definen tres severidades (alta, media y baja) pero podrían ser más. Por ej. usando las severidades del Cuadro A.1:

$$\begin{aligned} Capital = {} & Capital(BLOQUEANTE) + Capital(CRÍTICA) \\ & + Capital(MAYOR) + Capital(MENOR) \\ & + Capital(INFORMACIÓN) \end{aligned}$$

donde $Capital(S)$ es la suma de la Ecuación A.1 para las violaciones de severidad S.

Asimismo, el método de Curtis y cols. (2012) también agrega en el cálculo de $Capital(S)$ el porcentaje del capital que se desee pagar, sin embargo, el autor de esta obra piensa que es incorrecto: lo que se desee pagar no forma parte del cálculo del capital sino que es una decisión posterior a conocer la estimación del mismo.

Otras formas de desagregar el Capital es agrupando las violaciones según la característica de calidad que afectan, por ejemplo, si se detectan lugares donde es posible ser atacado con técnicas de *SQL injection*, o *Cross Site Scripting* (Wichers, 2013), cada una de estas violaciones pueden ser agrupadas bajo la categoría “Seguridad”. Otra categoría podría ser “Cambiabilidad”, comprendida por violaciones indicando código duplicado y funciones

con alta complejidad ciclomática (McCabe, 1976). Asimismo, cada categoría puede ser cruzada por la severidad de las violaciones, generando así una matriz que luego permita tomar decisiones con respecto a qué categoría y/o severidades de violaciones priorizar para una adecuada reparación.

Nugroho, Visser, y Kuipers (2011). Los autores llaman al capital el Esfuerzo de Reparación (RE), esto es el costo de reparar hasta alcanzar la calidad ideal, de acuerdo a la siguiente ecuación:

$$RE = RF \times RV \times RA$$

donde RF es la Fracción de Retrabajo, un estimado del porcentaje de LDC que necesitan ser cambiadas para mejorar la calidad (de acuerdo a una tabla con porcentajes pre-calculados por el método). RV es el Valor de Reconstrucción, una estimación del esfuerzo necesarios si se quiere reconstruir el sistema (que depende del tamaño en LDC). RA es el Ajuste por Refactoreo, incorporado como un porcentaje de descuento calculado por juicio de expertos (ej. $RA = 10\,\%$ según sus autores).

A.4. Estimación del Interés

La estimación del interés de la deuda técnica es una métrica menos confiable que el Capital, ya que el esfuerzo extra que es necesario realizar para modificar un sistema o módulo por causa de no haber reparado los problemas de calidad hasta alcanzar el nivel ideal, depende de varios factores.

En principio, para hacer una proyección a futuro del interés, es necesario estimar el esfuerzo de desarrollar los nuevos requerimientos con la deuda pagada y sin la deuda pagada, para luego calcular su diferencia. Este enfoque resulta en trabajo extra para el equipo de trabajo.

Además, los requerimientos deben estar priorizados, y estas priorizaciones deben permanecer estables, a efectos de preparar un cronograma o planificación con el orden de desarrollo de los requerimientos. Esto es importante al momento de decidir hacer el repago de la deuda o pagar los intereses,

puesto que la estimación del interés pierde confiabilidad conforme el cronograma coloque muy a futuro un requerimiento en cuestión. Lo anterior, sin perjuicio de que (en una etapa de mantenimiento del sistema) los requerimientos no suelen conocerse, incluso estimarse y priorizarse muy a futuro (meses o años hacia adelante).

Algunos de los métodos publicados para calcular el interés se esbozarán a continuación.

Nugroho, Visser, y Kuipers (2011). Proponen una fórmula para calcular el interés, basándose en su método para calcular la deuda técnica, que otorga a un sistema un rating de 1 a 5 estrellas (1=peor, 5=calidad ideal). El interés lo calcula como la diferencia entre el esfuerzo de mantenimiento (ME) entre un nivel de 5 estrellas y el nivel de calidad (estrellas) del sistema actual:

$$ME = \frac{MF \times RV}{QF}$$

donde MF es el esfuerzo de mantenimiento del sistema en forma anual medido en porcentaje de LDC que se estima que van a cambiar. Su modelo sugiere que es aproximadamente 15 % en cambios en el código (aunque esto depende del muestreo particular que hayan realizado). RV es el Valor de Reconstrucción, o esfuerzo necesario para reconstruir el sistema, que depende del conteo de LDC. Por último, $QF = 2^{(NivelCalidad-3)/2}$ es un factor que depende del nivel de calidad actual y aumenta a medida que éste aumenta (en cantidad de estrellas), o sea de 1 a 5, QF es $0{,}5$, $0{,}7$, $1{,}0$, $1{,}4$ ó $2{,}0$.

Falessi y Reichel (2015). En otro método de calcular el interés, se propone una herramienta académica que lo estima en base al análisis del repositorio de código fuente (Git) para detectar los cambios realizados históricamente, y una herramienta de seguimiento de defectos (Redmine) para entender cómo los cambios en el código se relacionan al defecto. La herramienta, llamada MIND, mide el interés en términos de cuantos defectos adicionales ocurrieron, o ocurrirán, debido a violaciones de reglas de calidad (extraídas de Sonarqube).

Singh, Snipes, y Kraft (2014). Por último, usando en un enfoque de análisis del trabajo de una persona al modificar un sistema, se propone un *plugin* para el IDE Visual Studio, que registra la secuencia de acciones que realiza el desarrollador para modificar el código fuente, incluyendo cuando navega en la estructura del proyecto y actividades de edición, para luego salvar un log de estas acciones. Así, se cuantifica el esfuerzo que realiza el desarrollador para comprender el sistema mientras completa las modificaciones requeridas. En este contexto, el interés es la diferencia de tiempo entre comprender el código con la estructura actual y con una estructura ideal. Las estimaciones pueden estar viciadas con desviaciones de tiempo provenientes de actividades extras del desarrollador mientras está editando (ej. atendiendo el teléfono).

A.5. Estimación del punto de equilibrio

En referencia a la teoría económica, es interesante poder determinar si existe un momento en el tiempo en el cual pagar el capital de la deuda técnica requiere menos esfuerzo que lidiar con el esfuerzo extra producto de los intereses.

Nugroho, Visser, y Kuipers (2011). En base a las estimaciones de capital (RE) e interés (ME) presentadas previamente en este Apéndice, si se compara el RE pagado en su totalidad en el período actual, y el acumulado de ME de todos los períodos a futuro (escala en meses o años), se puede obtener el período en el cual se cruzan ambos valores.

Chatzigeorgiou, Ampatzoglou, Ampatzoglou, y Amanatidis (2015). Sugieren un framework para calcular en qué versión del producto se igualará el capital con la suma de los intereses acumulados. Como es esperable, el autor alerta sobre la calidad y validez de esta estimación conforme se aventura a realizarse sobre el futuro distante.

El modelo realiza un análisis histórico de las últimas versiones del producto, lo que sugiere que cuanto más versiones o *releases* se hayan entre-

gado, más fiable será la estimación. Se utiliza un algoritmo que arroja una "distancia" entre un diseño y un diseño óptimo, que se considera el capital a pagar. La diferencia entre los esfuerzos de agregar funcionalidad sobre el diseño actual y el óptimo constituye el interés. La versión en la que se llega al punto de equilibrio se calcula como el cociente entre el capital y el interés.

Referencias

Alves, N. S., Ribeiro, L. F., Caires, V., Mendes, T. S., y Spinola, R. O. (2014). Towards an ontology of terms on technical debt. En *Managing technical debt (mtd), 2014 sixth international workshop on* (pp. 1–7).

Ambler, S. (2015). *2015 Q1 Agile State of the Art Survey Results* (Inf. Téc.). Scott Ambler + Associated. Descargado de `http://www.ambysoft.com/surveys/agile2015Q1.html` (Accedido: 08-Noviembre-2016)

Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., y Avgeriou, P. (2015). The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology*, *64*, 52–73.

Automated Source Code Maintainability Measure (ASCMM) V1.0 (Inf. Téc.). (2016, January). OMG. Descargado de `http://www.omg.org/spec/ASCMM/1.0/`

Avgeriou, P., Kruchten, P., Nord, R. L., Ozkaya, I., y Seaman, C. (2016). Reducing friction in software development. *IEEE Software*, *33*(1), 66–73.

Baggen, R., Correia, J. P., Schill, K., y Visser, J. (2012). Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, *20*(2), 287–307.

Beck, K., y Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional.

Blank, S. (2015, Junio). *Lean Innovation Management - Making Corporate Innovation Work.* Descargado de `https://steveblank.com/2015/06/26/lean-innovation-management-making-corporate-innovation-work/` (Accedido: 20-Setiembre-2016)

Booch, G. (2006). The accidental architecture. *IEEE software*, *23*(3), 9.

Carr, N. (2003). IT doesn't matter. *Harvard business review*, *81*(5), 41.

Chatzigeorgiou, A., Ampatzoglou, A., Ampatzoglou, A., y Amanatidis, T. (2015). Estimating the breaking point for technical debt. En *Managing technical debt (mtd), 2015 ieee 7th international workshop on* (pp. 53–56).

Cobertura. (s.f.). *Cobertura: A code coverage utility for Java.* Descargado de `http://cobertura.github.io/cobertura/` (Accedido: 08-Junio-2016)

Coley, S. (2009). *Enduring Ideas: The three horizons of growth.* Descargado de `http://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/enduring-ideas-the-three-horizons-of-growth` (Accedido: 19-Setiembre-2016)

Conway, M. E. (1968). How do committees invent. *Datamation*, *14*(4), 28–31.

Cunningham, W. (1992). The WyCash Portfolio Management System. En *Addendum to the proceedings on object-oriented programming systems, languages, and applications (addendum)* (pp. 29–30). New York, NY, USA: ACM. Descargado de `http://c2.com/doc/oopsla92.html`

Curtis, B., Dickenson, B., y Kinsey, C. (2015). *CISQ Recommendation Guide - Effective Software Quality Metrics for ADM Service Level Agreements* (Inf. Téc.). CISQ. Descargado de `http://it-cisq.org/wp-content/uploads/2016/04/CISQ-Recommendation-Guide-Effective-Software-Quality-Metrics-for-ADM-Service-Level-Agreements.pdf`

Curtis, B., Sappidi, J., y Szynkarski, A. (2012). Estimating the principal of an application's technical debt. *IEEE software*(6), 34–42.

DiRomualdo, A., y Gurbaxani, V. (1998). Strategic intent for IT outsourcing. *MIT Sloan Management Review*, *39*(4), 67.

El Sawy, O. A. (2003). The IS Core IX: The 3 Faces of IS identity: connection, immersion, and fusion. *Communications of the Association for Information Systems*, *12*(1), 39.

Environmental Protection Agency. (2015, Setiembre). *EPA, California Notify Volkswagen of Clean Air Act Violations / Carmaker allegedly used software that circumvents emissions testing for certain air pollutants.* Descargado de `https://yosemite.epa.gov/opa/admpress.nsf/a883dc3da7094f97852572a00065d7d8/`

dfc8e33b5ab162b985257ec40057813b!OpenDocument (Accedido: 23-Mayo-2016)

Ernst, N. A., Bellomo, S., Ozkaya, I., Nord, R. L., y Gorton, I. (2015). Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt. En *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp. 50–60). New York, NY, USA: ACM.

Falessi, D., y Reichel, A. (2015). Towards an open-source tool for measuring and visualizing the interest of technical debt. En *Managing technical debt (mtd), 2015 ieee 7th international workshop on* (pp. 1–8).

Fernández-Sánchez, C., Garbajosa, J., y Yague, A. (2015). A framework to aid in decision making for technical debt management. En *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on* (pp. 69–76).

Fowler, M. (1999). *Refactoring: improving the design of existing code*. Pearson Education India.

Fowler, M. (2006, Mayo). Code Ownerwhip. *Bliki [Blog]. Available from: http://martinfowler.com/bliki/CodeOwnership.html*.

Fowler, M. (2009, Octubre). Technical debt quadrant. *Bliki [Blog]. Available from: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html*.

Freijedo, C. (2015). *Material complementario - parte 1.* Lección Materia Gestión Estratégica de las TICs.

Gaudin, O. (2015, Julio). *Water Leak Changes the Game for Technical Debt Management.* Descargado de http://www.sonarqube.org/water-leak-changes-the-game-for-technical-debt-management/ (Accedido: 09-Mayo-2016)

Henderson, B. (1970). *The Product Portfolio.* Descargado de https://www.bcgperspectives.com/content/classics/strategy_the_product_portfolio/ (Accedido: 31-Octubre-2016)

ISACA, C. (2012a). *COBIT 5: Implementación*. ISACA.

ISACA, C. (2012b). COBIT 5: Un Marco de Negocio para el Gobierno y la Gestión de las TI de la Empresa. *Estados Unidos: ISACA*.

ISO, I. (2011). IEEE: 42010: 2011 systems and software engineering, archi-

tecture description. *International Standard*.

ISO/IEC. (2011). *ISO/IEC 25010 - systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - system and software quality models* (Inf. Téc.). Descargado de `http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733`

Jones, C., y Bonsignour, O. (2011). *The economics of software quality*. Addison-Wesley Professional.

Kruchten, P. (2011, Julio). *What colours is your backlog?* Descargado de `https://pkruchten.files.wordpress.com/2013/12/kruchten-colours-yow-sydney.pdf` (Accedido: 23-Mayo-2016)

Kurian, G. T. (2013). *The AMA dictionary of business and management*. AMACOM Div American Mgmt Assn.

Lacity, M. C., Willcocks, L. P., y Feeny, D. F. (1996). The value of selective IT sourcing. *MIT Sloan Management Review*, *37*(3), 13. Descargado de `http://sloanreview.mit.edu/article/the-value-of-selective-it-sourcing/`

Lehman, M. M. (1996). Laws of software evolution revisited. En *Software process technology* (pp. 108–124). Springer.

Letouzey, J.-L. (2016, Marzo). The SQALE Method for Managing Technical Debt - Definition Document [Manual de software informático]. Descargado de `http://www.sqale.org/wp-content/uploads/2016/04/SQALE-Method-EN-V1-1.docx` (Accedido: 27-Junio-2016)

Letouzey, J.-L., y Ilkiewicz, M. (2012). Managing technical debt with the sqale method. *IEEE software*, *29*(6), 44–51.

Li, Z., Avgeriou, P., y Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, *101*, 193–220.

Lim, E., Taksande, N., y Seaman, C. (2012). A balancing act: what software practitioners have to say about technical debt. *Software, ieee*, *29*(6), 22–27.

McCabe, T. J. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*(4), 308–320.

McConnell, S. (2004). *Code Complete, Second Edition*. Redmond, WA, USA: Microsoft Press.

McConnell, S. (2007, Noviembre). *Technical Debt.* Descargado de http://www.construx.com/10x_Software_Development/Technical_Debt/ (Accedido: 09-Mayo-2016)

McFarlan, F. W., McKenney, J. L., y Pyburn, P. (1983). *The information archipelago-plotting a course*. Reprint Service, Harvard business review.

Nicolau-Juliá, D., Expósito-Langa, M., y Tomás-Miquel, J.-V. (2015). Exploración y explotación de conocimiento en el ámbito empresarial. validación de escalas en un sector industrial de bajo perfil tecnológico. *Investigaciones Europeas de Dirección y Economía de la Empresa*, *21*(3), 139–147.

Nugroho, A., Visser, J., y Kuipers, T. (2011). An Empirical Model of Technical Debt and Interest. En *Proceedings of the 2Nd Workshop on Managing Technical Debt* (pp. 1–8). New York, NY, USA: ACM.

OPSSI. (2016, Abril). *Reporte anual sobre el Sector de Software y Servicios Informáticos de la República Argentina - Reporte año 2015* (Inf. Téc.). Cámara de Empresas de Software y Servicios Informáticos de la República Argentina. Descargado de http://www.cessi.org.ar/descarga-institucionales-2007/documento2-130347cd83ae771a9f3db3da5407269a

Project Management Institute. (2013). *A Guide to the Project Management Body of Knowledge (PMBOK Guide).* Project Management Institute. Descargado de https://books.google.com.ar/books?id=FpatMQEACAAJ

Rao, A. (2016, Abril). *Creating a Debt-aware Culture.* Descargado de https://www.infoq.com/presentations/debt-aware-culture (Accedido: 08-Noviembre-2016)

Robertson, D. (2012, Agosto). *2009 Robertson Enterprise Architecture Speech.* Online. Descargado de https://www.youtube.com/watch?v=aZha3iL-TJA (Accedido: 05-Octubre-2016)

Ross, J., Weill, P., y Robertson, D. (2006). *Enterprise architecture as strategy : creating a foundation for business execution*. Cambridge, MA: Harvard Business School Press. Boston, Mass.

San Miguel, J. (2014, Febrero). KISS, YAGNI & DRY, 3 Principles to Simplify Your Life as Developer. *iTexico Blog*. Online. Descargado de `http://martinfowler.com/bliki/CodeOwnership.html` (Accedido: 08-Noviembre-2016)

Sappidi, J., Curtis, B., y Szynkarski, A. (2012). *CAST Report on Application Software Health - 2011/12 (Summary of Key Findings)* (Inf. Téc.). CAST Research Labs. Descargado de `http://www.castsoftware.com/research-labs/technical-debt-estimation`

Sharma, T., y Samarthyam, G. (2015, Agosto). Limitations of Technical Debt Quantification: Do you Rely on these Numbers? *InfoQ*. Descargado de `https://www.infoq.com/articles/tech-debt-quantification` (Accedido: 18-Agosto-2015)

Singh, V., Snipes, W., y Kraft, N. A. (2014). A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension. En *Managing technical debt (mtd), 2014 sixth international workshop on* (pp. 27–30).

Sterling, C. (2010). *Managing Software Debt: Building for Inevitable Change*. Addison-Wesley Professional.

Taylor, H. (2006, noviembre). Critical Risks in Outsourced IT Projects: The Intractable and the Unforeseen. *Commun. ACM*, *49*(11), 74–79.

The Open Group. (2011). TOGAF version 9.1 (Manual de software informático n.º G116). Online. Descargado de `http://pubs.opengroup.org/architecture/togaf9-doc/arch/`

TIOBE Software. (s.f.). *TIOBE Programming Community Index Definition.* Descargado de `http://www.tiobe.com/tiobe-index/programming-languages-definition/` (Accedido: 08-Noviembre-2016)

Weill, P., y Ross, J. (2005). A Matrixed Approach to Designing IT governance. *MIT Sloan Management Review*, *46*(2), 26–34.

Whitten, D. (2010). Adaptability in IT Sourcing: The impact of switching costs. En *International workshop on global sourcing of information technology and business processes* (pp. 202–216).

Wichers, D. (2013). OWASP Top-10 2013. *OWASP Foundation, February*.

Wikipedia. (2016a). *DevOps — Wikipedia, the free encyclopedia.* Descargado de `https://en.wikipedia.org/wiki/DevOps` ([Online; accedido 18-Octubre-2016])

Wikipedia. (2016b). *Difusión de innovaciones — Wikipedia, the free encyclopedia.* Descargado de `https://es.wikipedia.org/wiki/Difusi%C3%B3n_de_innovaciones` ([Online; accedido 16-Setiembre-2016])

Woodard, C. J., Ramasubbu, N., Tschang, F. T., y Sambamurthy, V. (2012). Design capital and design moves: the logic of digital business strategy. *Forthcoming, MIS Quarterly, Special Issue on Digital Business Strategy: Toward a Next Generation of Insights*.

Zachman, J. A. (1996). The framework for enterprise architecture: background, description and utility. *Zachman International*, 1–5.

Zazworka, N., Shaw, M. A., Shull, F., y Seaman, C. (2011). Investigating the impact of design debt on software quality. En *Proceedings of the 2nd workshop on managing technical debt* (pp. 17–23).

Declaración Jurada de origen de los contenidos

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente en la Escuela de Estudios de Posgrado y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

Gabriel Alfredo Belingueres
DNI 24137123
FIRMADO