¿Qué es?

La deuda técnica refiere a un conjunto de acciones o decisiones al momento de desarrollar o mantener un sistema que pueden o no afectar resultados en el futuro.

Se pueden ver como esas acciones que se “hacen para salir del paso”: en principio se desarrollan con la idea de generar una solución fácil y rápida, y probablemente para así terminar una versión lo antes posible; pero al largo plazo puede generar riesgos en el sistema que no optimicen el funcionamiento en general, o inclusive (si no se detectan a tiempo) se pueden transformar en defectos si es percibido por el usuario.

Kruchten lo define como una metáfora, a mi entender lo hace de esta forma para notar que es una idea general que puede ocurrir de muchas formas (más adelante se desarrollará) y que las consecuencias pueden ser tanto positivas como negativas (según el momento en el que se trate y la forma luego de detectadas).

Es importante destacar que los problemas adjudicados a la deuda técnica no surgen ni se deben a acciones previas al desarrollo ni en ese momento, sino que ocurren en el futuro. Y la gravedad de estos no siempre es la misma frente a la misma acción sino que depende del contexto en que se están dando.

Existen dos tipos de deuda técnica:

- No intencional: no planeada, en general ocurre por falta de conocimiento
- Intencional: aquella que se es consciente de que se realiza la acción a cambio de ganar otra cosa (ej tiempo)

¿Dónde se encuentra?

La deuda técnica en general se asocia al producirse en código del sistema. Pero, siguiendo la filosofía de acciones que se toman como “atajos” en principio que luego en un futuro tienen sus consecuencias, la deuda técnica se puede encontrar también en:

- Diseño del sistema
- Creación y mantenimiento de la arquitectura
- Documentación
- Testeo
- Instalación

¿Cómo se manifiesta?

La gestión de la deuda técnica tiene varias etapas, y según el momento en la que se detecte, es la forma en que se manifiesta.

Por ejemplo en etapas tempranas, durante la instancia de codificación puede ser detectada como un error durante la revisión de código (en caso de que se realice). En etapas de testeo se la puede detectar cuando no se ejecutan todos los tests disponibles, o no se diseñan para cubrir todos los casos de prueba. También cuando el sistema se encuentra en producción cuando ocurre un error (en ese caso ya sería un defecto).

Hasta este punto, se describieron ejemplos donde la deuda técnica se manifiesta en forma negativa, pero Kruchten indica que también se la puede ver como un aspecto positivo porque podría presentar una instancia para testear un posible mercado o reevaluar ciertos riesgos.

¿Cómo la perciben los desarrolladores?

En general los desarrolladores detectan la presencia de la deuda técnica por los code smells (no todos producen deuda técnica, por ejemplo: no sería el caso cuando ocurre en códigos que no se manipulan con frecuencia): bloques de código que se saben que no cumplen principios de diseño y afectan la calidad del software.

También la pueden percibir al momento de desarrollar: escribiendo un código que no sigue un estándar establecido o no eficiente.

Dar varios ejemplos diferentes de deuda técnica.

En codificación: Nombrar de forma incorrecta variables, hardcodeado posibles valores parametrizables por el usuario, uso de código duplicado.

En diseño: No tener en cuenta casos borde o alternativos (por ejemplo que se pasen a usar más de un tipo de documento de identidad).

En documentación: Desarrollo escaso de la documentación de un sistema.

En testing: No correr todos los tests disponibles o no realizar los tests suficientes que cubran todos los flujos o resultados posibles.

Tipos de Deuda Técnica

Basado en la documentación brindada existen distintos tipos de deuda técnica. Las más destacadas son:

- Deuda en Arquitectura

  Causado por decisiones que perjudican en la calidad de la arquitectura del sistema, el mantenimiento y la escalabilidad.

  En el caso de Genexus, a partir de las transacciones creadas por el usuario la plataforma se encarga de crear y gestionar aspectos como la normalización de tablas, definición de los índices básicos y de la integridad referencial. De esta forma uno tiene cierta garantía en lo que respecta a la calidad. Esto no implica que no exista este tipo de deuda técnica ya que quizás se puede arrastrar desde las propias transacciones. Por ejemplo el seteo de atributos o campos duplicados.

- Deuda en el código

  Causado por código que no cumpla modularidad, reusabilidad o legibilidad.

  Si bien Genexus se define como una plataforma *low code*, es sabido que se debe programar en esta. Ejemplos como manejar código duplicado en lugar de implementar teniendo en cuenta la reusabilidad, código muerto o alto acoplamiento se pueden dar. Otro factor es el no cumplimiento de las buenas prácticas de desarrollo.

- Deuda en compilación

  Ocurre al momento de compilar el proyecto. Un caso que puede causar deuda técnica en Genexus de esta clase puede ser debido al uso de versiones anteriores con librerías o dependencias deprecadas que no dependen del desarrollador, no son fácil de manipular

- Deuda en documentación

  Errores relacionados a la documentación, ya sea relacionados a otros tipos de deuda técnica que requieran un seguimiento, como otras características que requieran documentación: requerimientos, relaciones, diseño, testeo, codificación.

- Deuda en requerimientos

  Ocurre cuando los requerimientos no coinciden con el producto final, o cuando estos no son los buscados por el cliente. Quizás este tipo de deuda técnica se relacione con Genexus al momento de querer cumplir requerimientos no funcionales (por ejemplo tiempo de respuesta) donde quizás no se puedan controlar tanto.

- Deuda en Testing

  Refieren a posibles “defectos” o errores desarrollados durante el testeo: falta de casos de prueba, no se cubre todo el código fuente, etc.

Gestión de la deuda técnica

Percibir y controlar la deuda técnica que se tiene en un sistema permite organizar las actividades dentro del proyecto en el que se está desarrollando, mantener los niveles de deuda técnica bajos y registrar tanto el trabajo realizado como los beneficios que conlleva.

Durante el proceso de gestión se pueden identificar distintas actividades o etapas: Identificación, Documentación o Representación, Comunicación, Medición, Priorización, Pago, Monitorización, Prevención.

Herramientas o estrategias usadas para la gestión de la deuda técnica

En cada una de las actividades de la gestión de la deuda técnica se usan distintas herramientas o estrategias para llevar adelante la tarea.

Actividad	Tecnologías y estrategias
Identificación	Inspección manual de código (59), Análisis de dependencias (33), checklist (23), SonarQube/SCALE (15), CheckStyle (10), FindBugs (9), Code Climate (2)
Documentación /representación	TD backlog (21), Artf. específicos para documentar TD (1), JIRA (24), Wiki (9), Trello (2), Excel (1), Youtrack (1)
Comunicación	Foros de discusión (14), Tópico de TD en reuniones de proyecto (29), Reuniones específicas de TD (22)
Medición	Medición manual (7), SonarQube (4), JIRA (8), Wiki (2), Herramienta propietaria (1)
Priorización	Análisis costo/beneficio (15), Tecnologías específicas para la toma de decisiones (6), Clasificación de issues (35)
Pago	Refactoring (47), Re-escritura de código (44), Re-diseño (35)
Monitorización	Manual (15), SonarQube (2), JIRA (8), Wiki (3), Definición de hecho (2)
Prevención	Guías (32), Estándares de codificación (40), Revisiones de Código (44), Reuniones de retrospectivas (36), Definición de hecho (24)
Otras	“Dejar la casa más limpia que como la encontré”

Tabla 3 – Tecnologías utilizadas para las actividades de gestión de la deuda técnica

En particular, en el contexto de Genexus no seria posible usar directamente herramientas como SonarQube, CheckStyle, FindBugs y Code Climate en la plataforma, pero quizás se pueden ejecutar sobre el código generado (en el caso de Java o .Net). Se debería profundizar si esto es posible y si esto maneja otras consecuencias.

Por otra parte, si bien Genexus maneja una plataforma *Wiki* (https://wiki.genexus.com) donde la propia empresa que ofrece el entorno de programación define y documenta funcionalidades, considero que la plataforma no tiene tanta especificación, desarrollo o ejemplos en las mismas.

En lo que respecta a las actividades de documentación, además del uso de plataformas como Trello, Jira o los propios editores de texto usuales, Genexus ofrece 2 opciones. *Knowledge Base Documentation* es un entorno donde se puede desarrollar una plantilla principal, cargar imágenes y documentos que pueden ser referenciados desde el primero y de esta forma hacer una descripción general del sistema con el que se está trabajando. Además, cada objeto también tiene un área de *documentación* donde uno puede documentar todo lo que considere necesario de este objeto, por lo que se podría usar para registrar características propias de este que generen deuda técnica. En lo personal, tanto la redacción como la revisión del área de documentación no son actividades que practique y desconozco que tanto se use esta opción dentro de la comunidad.

Luego, otro punto a tener en cuenta es el nivel de actividad que tienen los foros de discusión. Los foros pueden ser tanto públicos (plataformas en la web) como privados (comunicación interna en la empresa o el equipo de trabajo). El nivel de actividad de los foros internos pueden ser gestionados por un encargado del área. Pero los foros públicos dependen exclusivamente del nivel de participación y el “entusiasmo” de cada participante por permitir que la comunidad crezca. En lo que a mi respecta noto que la comunidad Genexus no se presenta tan activa en páginas web o foros (ej Stack Overflow) como sucede en otros lenguajes de programación.

Metamodelo de la deuda técnica

En términos generales todos los elementos del metamodelo planteado aparecen en el contexto de Genexus ya que estos son características de la deuda técnica como concepto. Por tanto, no veo que algunos sean inaplicables este lenguaje de programación en particular.

Aquellos elementos que estén más relacionados al equipo de trabajo, la metodología, los stakeholders o inclusive al mercado no tendrían tantas caracteristicas propias en Genexus al intentar compararlo con otros lenguajes de programación:

<u>Ocurrencia</u>
Si se maneja el concepto solamente desde el lado de la intencionalidad y no tanto del motivo (que en esos casos puede influenciar aspectos técnicos de la programación), la

ocurrencia tanto intencional como no intencional tienen el mismo desarrollo en distintos lenguajes de programación. Por ejemplo el desarrollo por desconocimiento o por presión de tiempo.

Factores que contribuyen

Los factores que pueden crear Deuda Técnica darse por:

- Características del proyecto: problemas en los requisitos, o decisión en el diseño
- Equipo: por desconocimiento o inexperiencia
- Decisiones tomadas por falta de tiempo por ejemplo para apurar los procesos

Influencias (positivas y negativas)

Pueden ser similares a los factores

- Relacionadas del equipo de trabajo.
- Características propias del mercado que limitan los requerimientos, limitando la escalabilidad si se diseña en particular para ese mercado.
- Gestión del proyecto que conlleva a tomar decisiones por falta de tiempo por ejemplo.

Gestión de la Deuda Técnica

Esta clase puede tener características propias de Genexus cuando se desea gestionar ítems propios al lenguaje, pero la gestión en sí seguramente sea indistinta del lenguaje que se esté manejando.

Otro punto a tener en cuenta puede ser la asignación de roles ya que con Genexus no se requieren tantos integrantes para llevar adelante un proyecto.

Como se desarrolló previamente, dentro de los distintos tipos de deuda técnica algunos tienen sus particularidades dentro del lenguaje. Pero también, otros elementos que pueden ser más interesantes en analizar debido a su cercanía al lenguaje pueden ser:

Decisiones técnicas

En este punto abarcan las que a mi entender son las se asocian primero: las que respecta a la programación, diseño o testeo

- no seguir estándares o buenas prácticas a la hora de programar
- mal uso de las recorridas (for each), ya sea por un filtrado ineficiente o no uso inadecuado de índices.
- pensar en diseño de las transacciones para los requerimientos y no dar lugar a un modelo escalable

Gestión de los aparatos de Software

- Artefactos propios de Genexus
  objetos como las transacciones (por el diseño de la arquitectura), y procedures (por el código), data providers (por el código no performante) o tipos de datos (no escalables)
- Además, la gestión de requisitos puede no ser tan profunda dado la facilidad de integrar al momento de mantener un software.

Algunos puntos a tener en cuenta a futuro para desarrollar en la Deuda Técnica:

- Desarrollar un poco más sobre casos de buenas prácticas de desarrollo, ejemplos más explícitos sobre poca modularidad o poca reusabilidad del código
- Existencia y manejo de distintas versiones de Genexus
    - consecuencias de migrar a más recientes
    - usar versiones sin mantenimiento
    - comunicación con la comunidad
- Considerar el hecho de empezar a trabajar en Gx habiendo hecho solo un curso del lenguaje, sin conocimientos en otras áreas como Ingeniería de Software o base de datos.
- Poca participación de la comunidad
- No existencia de repositorios
    - lejos de usar algo como Git
    - Genexus server no resuelve del todo
    - También está el Gx Market
- Cómo puede afectar que Gx genere el código final
- Investigar sobre el testing en Gx
- No trakear defectos
- Pensar características que influye positivamente