Deuda técnica GX

La deuda técnica no tiene una definición específica, sino que refiere a un concepto, idea o filosofía, tampoco implica ciertas acciones en concreto ni un momento del tiempo dado y generalmente no puede aislarse del contexto donde surge.

Si se puede definir que la deuda técnico no son defectos en el software, ya sea a nivel de código, de funcionamiento o visuales, podemos decir también que la deuda técnica es visible solamente en la interna de los equipos de trabajo que se encargan de diseñar, desarrollar, testear o documentar un proyecto de software y nunca es visible al usuario final o consumidor de dicho software. Si es claro que la acumulación de deuda técnica aumenta el riesgo de tener defectos en el sistema.

Entonces por deuda técnica podemos entender que se trata de decisiones, definiciones o acciones que podrían tener por objetivo obtener un beneficio a corto plazo, pero que en un futuro de no ser "pagada" o corregida generarán mayores esfuerzos o costos a nivel de desarrollo, modificación, evolución o mantenimiento del sistema y claramente siempre estamos hablando de software o sistemas funcionales, siendo el principal generador de deuda técnica el apremio de tiempo. Además la generación de deuda técnica puede ser consciente o inconsciente, podemos saber de antemano que existe deuda técnica o que la estamos generando o que la misma solo sea expresada o visibilizada solamente en el futuro ya sea por la evolución del propio software, decisiones de negocio, políticas o cualquier otra cuestión que genere un cambio.

Esta pseudo definición de deuda técnica es a nivel general, lo cual también aplica al entorno Genexus aunque de forma particular.

Dentro de deuda técnica podemos encontrar dos grandes áreas, la primera implica factores de codificación principalmente o de calidad de codificación y la segunda factores de diseño, arquitectura, documentación, testing y demás elementos intervinientes en el desarrollo de software que no tratan el código en si como principal punto de partida

La primera, es la más fácil de detectar incluso existen herramientas de análisis automático de código los cuales nos ayudan a encontrar dichas cuestiones.

Por deuda técnica dentro de esta primera área o segmentación, entendemos aquellas decisiones que implican por ejemplo no respetar las buenas prácticas de programación (nomenclatura, reutilización, legibilidad, cometarios, correctitud), generar workarounds sobre ciertos problemas en vez de generar una solución directa.

A nivel de Genexus, por más que este catalogado como Low-Code, no necesariamente es un No-Code, Genexus aunque poco de todas formas posee código, por tanto puede ser susceptible a generar deuda técnica en este nivel, como por ejemplo.

1- No respetar las buenas practicas definidas en el entorno Genexus, como pueden ser:

a. No respetar la fácil integración y reutilización
b. No respetar la fácil comprensión por parte del programador
c. No respetar la nomenclatura en general y de tablas de forma heredada
d. Tener variables que no hagan referencia al atributo
e. Tener rules con reglas sin agrupación
f. Falta de descripción de los objetos de la KB
g. No usar enumerators en lugar de constantes

2- No hacer limpieza de la KB, sobre objetos en desuso
3- Desconocimiento de quien utiliza la herramienta
4- Presiones de tiempo

Aunque, muchos otros elementos que pueden ser categorizados como Deuda Técnica o generadores de deuda técnica a nivel de código son evitables al utilizar Genexus dada su naturaleza de Low-Code, también esta simplicidad o accesibilidad a la hora de generar piezas de código que ofrece, deriva en un uso de la plataforma de desarrollo por gente no capacitada, o pobremente capacitada en el desarrollo de software, lo cual ciertamente implica un mayor riesgo sobre los pocos ítems generadores de Deuda técnica a nivel de código por dicha falta de expertise de los usuarios.

Por ultimo en esta categoría, Genexus tiene cierta falencia en cuanto a lo que es integración y colaboración de código, la herramienta Genexus Server aún no está al nivel de otras como GitHub u otras que permiten la colaboración mediante request por ejemplo, por tanto esta "debilidad" deriva en mecanismos más rudimentarios de colaboración que pueden derivar en deuda técnica, y porque no esto mismo es una deuda técnica en si del propio Genexus.

Por otro lado por deuda técnica en el segundo conjunto o área la cual incluye diseño, arquitectura, documentación, testing, etc. nos encontramos con un mundo mucho más amplio y heterogéneo el cual viene indivisiblemente de la mano del contexto en el cual se está trabajando, las decisiones que generen deuda técnica en un contexto no tienen porque generarlo en otro.

La Deuda técnica en esta segmentación son las más difíciles de detectar, aquí ni Genexus ni ningún otro low-code esta exento, ya que hablamos de decisiones u acciones a nivel de diseño o de definición del software o cualquier otro tipo de decisión estratégica, omisión de acción o decisión sobre el proceso o producto en si mismo, tampoco hablamos de errores o defectos a la hora de diseñar o definir la arquitectura del sistema, siempre estamos dentro de la "burbuja" de decisiones que buscan una beneficio a corto plazo o no son intencionales pero que no afectan el funcionamiento del sistema en sí, y no son visibles para los usuarios. Claramente este tipo de deuda técnica siempre es visible a futuro cuando se está en la necesidad de evolucionar, modificar, o corregir lógica de negocio o el software en sí, y estas decisiones tomadas en el pasado nos impactan en un mayor costo, esfuerzo de trabajo o un impedimento para realizarlas.

Dentro de esta categoría podemos encontrar algunos de estos elementos como posibles generadores de Deuda técnica a nivel de Arquitectura.

1) Haber generado un producto mínimamente viable, y el mismo queda atascado sin poder evolucionar.
2) Workarounds que se vuelven permanentes
3) Querer reinventar la rueda, en vez de utilizar soluciones ya desarrolladas
4) Bloquearse en definir una arquitectura cuasi perfecta sin permitir el avance en el desarrollo (al no permitir dicho avance se pueden generar retrasos en la codificación incurriendo en presiones de tiempo, o un excesivo aumento de iteraciones para cumplir con lo definido)
5) Utilizar arquitecturas obsoletas
6) Falta de actualización
7) Carencias de conocimiento arquitectónico
8) Carencias o nula documentación
9) Degradación de la arquitectura, mediante la perdida de conceptos o ideas iniciales
10) Factores de ignorancia o parcialidad
11) Falta de testing o documentación de testeo
12) El paso del tiempo

Claramente Genexus no está exento de ninguno de los puntos anteriores

Entonces podemos decir que en general las posibles causas de generación de deuda técnica en ambas categorías se terminan agrupando en:

1) Presiones de tiempo o calendario como principal
2) Ignorancia de cómo hacer mejor las cosas
3) Constante presión al cambio o evolución
4) Descuidos tanto en diseño como en desarrollo
5) Centrarse en el beneficio a corto plazo
6) No actualización o desfase muy grande en la versión de Genexus utilizada

Analizando el metamodelo diseñado por Jeronimo Junior desde el punto de vista de la deuda técnica en el desarrollo con Genexus, podemos encontrarnos con ciertos elementos que podrían o deberían tener mayor desagregación.

Como se menciona anteriormente Genexus posee código propio o reglas las cuales a la hora de ser compiladas en el lenguaje de destino puedan acarrear deuda técnica, si solo nos quedamos con los ítems del metamodelo que refieren a “Internal Qualitty Attribute” o “Software Artifact” podemos caer en el error de englobar en un mismo ítem dicho código generado el cual suponemos libre de deuda técnica a nivel de código o por lo menos algo en lo cual no es aplicable este análisis y el código ingresado por el usuario en procedimientos, transacciones, etc., propio de Genexus el cual sí puede llevarnos a generar deuda técnica. Tal vez podríamos derivar de “Software Artifact” un elemento llamado “Automatic Generated” y otro “User Generated”

Aclaremos que cuando hablamos de que el código generado por Genexus suponemos libre de deuda técnica o algo no aplicable en este análisis, nos referimos a que se tiene

poca o nula injerencia o control en el código generado por Genexus a la hora de desarrollar en Genexus y que de detectarse allí deuda técnica o bien viene heredada del código que si crea el usuario o debería considerarse un “bug” del propio sistema Genexus y carecería de sentido práctico acceder a dicho código y modificarlo directamente y manualmente cada vez que se hace un build, en este caso se debería reportar dicho “bug” a Genexus mismo y esperar una corrección.

Por otro lado, lo que refiere al ítem “Technnical Desicions” sería de utilidad por un lado que sea referenciada como “Strategic Desicions” y a esta desagregarla en diferentes tipos por ejemplo “Technnical/Technological”, “Archictecture/Design” y “Other”, visto que por ejemplo desde el punto de vista “Technnical/Technological” al hablar de Genexus podemos englobar lo que refiere al cambio o no de versión a trabajar (Actualización de versión Genexus) así como desde el punto de vista “Archictecture/Design” la utilización o no de UserControls, Extension, Patterns, o todo aquel objeto externo que pueda ser incorporado por ejemplo del MarketPlace de Genexus. Por ultimo dejando la especificación “Others” para aquellas decisiones más estratégicas no incluidas en los otros dos ítems o la falta de decisiones en si misma.

Cuestiones que tienen alto riesgo de generación de deuda técnica en base a la mantenibilidad, corrección o creación por quien los crea y a su estrecha relación con la posibilidad de mantener el software actualizado en la última versión de Genexus.

Por otro lado en la especificación “Others” pueden entrar decisiones como por ejemplo la asignación de recursos humanos al desarrollo y como se segmentaran, enfocaran o trabajaran en conjunto si es que lo hacen las personas en el desarrollo en Genexus, recordar que la falta de herramientas completamente desarrolladas a nivel contributivas y colaborativas puede generar deuda técnica.

En cuanto a los posibles elementos generadores de duda técnica definidos anteriormente, podríamos ubicarlos dentro del metamodelo de la siguiente forma.

1)
   1- No respetar las buenas practicas definidas en el entorno Genexus,
      lo ubicaríamos en la especificación de “Sotware Artifact“ ->“User Generated”
   2- No hacer limpieza de la KB, sobre objetos en desuso
      Aquí se podría ver como que el no realizar la limpieza es un “Strategic Decisions”->”Others” pero en si quien acumularía todos esos elementos “basura” que complejizan el software y generen deuda técnica estarían dentro de “Sotware Artifact“->“User Generated”
   3- Desconocimiento de quien utiliza la herramienta
      De igual manera aquí, designar a alguien a desarrollar en Genexus al cual le faltan los conocimientos podría verse como que es una “Strategic Decisions”->”Others”, pero las piezas de software que generan esta persona las cuales pueden traer consigo DTs irían en “Sotware Artifact“ ->“User Generated”

4- Presiones de tiempo
   Este ítem es claro que corresponde a “Strategic Decisions”->”Others” pues en vez de modificar el cronograma o de demorar el lanzamiento, o explicar el retraso, o porque simplemente no se puede hacer ninguna de las anteriores se decide presionar a los desarrolladores

2)

1) Haber generado un producto mínimamente viable, y el mismo queda atascado sin poder evolucionar.
   “Strategic Decisions”->“Archictecture/Design”
2) Workarounds que se vuelven permanentes
   “Sotware Artifact“ ->“User Generated” y “Strategic Decisions”->”Others”
3) Querer reinventar la rueda, en vez de utilizar soluciones ya desarrolladas
   “Strategic Decisions”->“Archictecture/Design”
4) Bloquearse en definir una arquitectura cuasi perfecta sin permitir el avance en el desarrollo (al no permitir dicho avance se pueden generar retrasos en la codificación incurriendo en presiones de tiempo, o un excesivo aumento de iteraciones para cumplir con lo definido)
   “Strategic Decisions”->“Archictecture/Design” y “Strategic Decisions”->”Others”
5) Utilizar arquitecturas obsoletas
   “Strategic Decisions”->“Technnical/Technological”
6) Falta de actualización
   “Strategic Decisions”->“Technnical/Technological”

7) Carencias de conocimiento arquitectónico
   “Strategic Decisions”->”Others” al poner a cargo personas con esa carencia
8) Carencias o nula documentación
   “Strategic Decisions”->”Others” al decidir no documentar o que la documentación no sea relevante
9) Degradación de la arquitectura, mediante la perdida de conceptos o ideas iniciales
   “Strategic Decisions”->“Archictecture/Design”
10) Factores de ignorancia o parcialidad
    “Strategic Decisions”->”Others”
11) Falta de testing o documentación de testeo
    “Strategic Decisions”->”Others”
12) El paso del tiempo
    “Strategic Decisions”->”Others” el no hacer nada con el paso del tiempo es una decisión en si misma.

Sobre los ítems definidos por Jeronimo, si interpretamos como “Source Code” no al código generado por Genexus de forma automática sino al código ingresado por el usuario ya sea en procedimientos, transacciones, reglas, o incluso código java script incrustado en el html, considero que las siguientes categorías son aplicables a Genexus: Code Debt y Design Debt.

Por otro lado como ya comentamos y dado que estas categorías forman parte del proceso de desarrollo de software en si independiente de la tecnología, también se puede incurrir en deuda técnica sobre las mismas: Architecture Debt, Documentation Debt, Requierement Debt, Test Debt, Infraestrucure Debt, Usability Debt, Process Debt, Defect Debt, Versioning Debt.

En el caso de Build Debt si lo viéramos como Build and Deply Debt, encontramos ciertas dificultades a la hora de realizar por ejemplo desde un repositorio compartido, la extracción, compilación y consecuente deploy de las aplicaciones en Genexus, que de no poder ser automatizada, generan complejidad y por lo tanto un mayor costo de accionar, determinando en esta falta de automatización una deuda técnica.

Por último el tipo People Debt, se trata a parte porque en Genexus tal vez sea más visible esto como un tipo de deuda técnica, por un lado no solo abarca todos los aspectos en esta categoría de cualquier otra tecnología o metodología de desarrollo sino que por ejemplo si ponemos a programar en una tecnología a alguien que no sabe de esta, sería visiblemente notorio por los posibles errores o falta de avance en el desarrollo y la persona no generaría deuda técnica en sí (sino más bien quien decidió ponerla a programar en esa tecnología), en cambio la facilidad que presenta Genexus para obtener un producto, la no exigencia de expertise a la hora de su uso, pueden llevarnos a obtener un software funcional pero con una gran acumulación de deuda técnica.

En base a todo lo anterior, podemos tener en cuenta algunas cuestiones que ayuden a minimizar o mitigar la deuda técnica, pero como se comentó previamente la eliminación es más bien una cuestión utópica, ya que la evolución, el paso del tiempo y demás actores hacen que lo que hoy no es deuda técnica lo pueda ser en un futuro.

Como principal forma de mitigación, se debe empezar por reconocer la existencia y visibilizar la deuda técnica a los demás actores dentro del proceso de desarrollo, se debe también implementar mecanismos para reducir la deuda técnica de forma regular o atacar aquellos elementos de deuda técnica que ya son conocidos no postergándolos en el tiempo donde serían más costosos.

A nivel de Genexus, la manifestación se puede ver claramente en el no uso de buenas prácticas, la falta de documentación o la errónea nomenclatura, haciendo que el sistema sea de alguna forma dependiente del desarrollador de turno, dado que un cambio en dicha persona o integrar a mas desarrolladores implica un costo en tiempo y esfuerzo en el entendimiento del código (aunque sea poco) y una dificultad para identificar los objetos en una KB y a que refieren.

Por otro lado a nivel arquitectónico, claramente la falta de documentación del proceso de negocio, la falta clara de diseño arquitectónico generando objetos, transacciones y demás elementos de forma indiscriminada por los desarrolladores, la falta de coordinación o aislación de los mismos a la hora de desarrollar impidiendo la reutilización y la carrera constante de cumplir con requerimientos puntuales que llueven sin orden, sin priorización, sin una línea clara de trabajo, sin filtro, etc. terminan generando una total degradación del diseño definido si lo hubo en algún momento o de un empalme de piezas de código sin sentido, totalmente dependientes de quien la haya

desarrollado, inentendibles, costosamente modificables, costosamente depurables y con muy alta chances de generación de defectos.

Desde el punto de vista del desarrollador, todo lo anterior se hace sumamente notorio en cuanto a la sobre exigencia de tiempo que le toma realizar algún cambio, entender que se hizo y como, depurar cualquier defecto, evitar que una modificación impacte en otra parte del sistema sin darse cuenta y principalmente al drástico aumento de la complejidad, costo en tiempo de entendimiento y desarrollo o impedimento a la hora de incorporar, mantener, modificar o corregir funcionalidades.