Para el proyecto denominado **STEAM-RADAR** uno de los atributos de calidad a evaluar será la eficiencia, para términos de este criterio, únicamente evaluaremos la eficiencia en términos de ejecución computacional, esto debido a que la eficiencia se puede ver desde el punto de vista de la usabilidad, de la seguridad, etc. Una vez aclarado dicho punto, procedemos a definir este atributo.

Según McCall, la eficiencia es "La cantidad de recursos informáticos y código requeridos por un programa para realizar una función". Honestamente dicha descripción queda muy corta, así que, la complementaremos con la definición que nos proporciona Pressman:

"La eficiencia de un sistema se puede medir a través del uso de recursos como tiempo de procesador, memoria, acceso a la red, instalaciones del sistema, espacio en disco, etc. Es un concepto relativo y multivariado, en el sentido de que un sistema puede ser identificado como más eficiente que otro en términos de algún parámetro, como el uso del procesador, pero no hay un solo escala en la que especificar una eficiencia óptima"

Como **factor de diseño**, la eficiencia es una propiedad difícil de manejar, ya que implica proyecciones a partir del diseño con el fin de estimar los efectos de las elecciones sobre las eventuales necesidades de recursos. Sin embargo, dado que estos son de considerable importancia en términos de la implementación del sistema eventual, a menudo es necesario hacer al menos crudo predicciones de necesidades.

Recordemos que según el modelo de McCall a cada factor le corresponden ciertos criterios y dados esos criterios, a cada una le corresponden ciertas métricas, las cuales son maneras objetivas, cuantificables y mesurables de asegurar o negar la existencia de cierto atributo de calidad. Se puede apreciar de manera más clara en la siguiente imagen anexo:

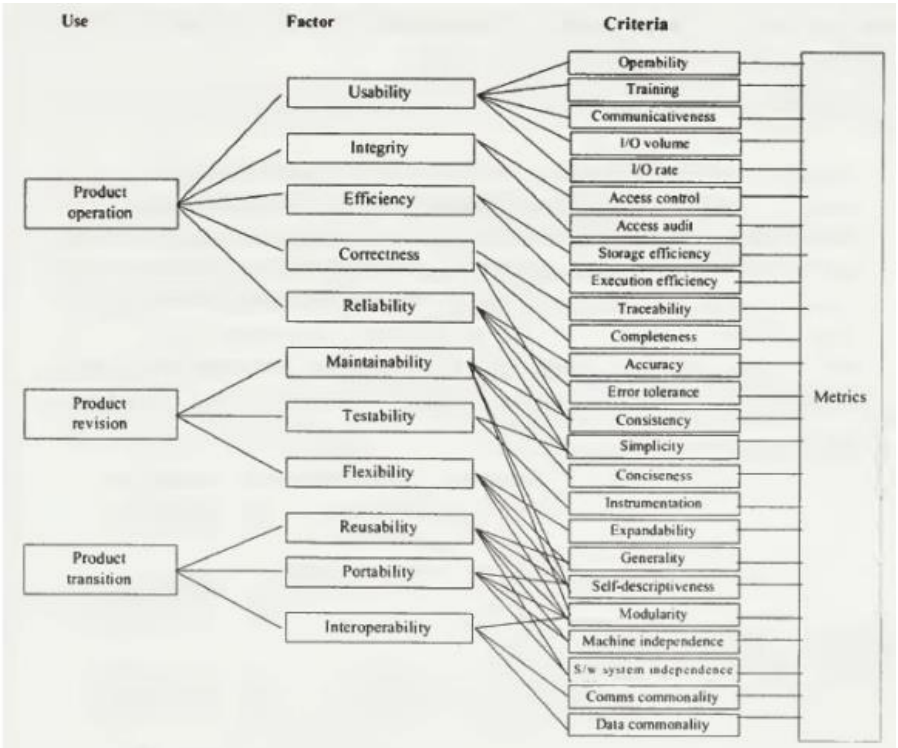

Figure 9.2: McCall software quality model

Podemos apreciar que, según McCall, sus 2 criterios de calidad son:

- **Eficiencia de ejecución**: Aquellos atributos del software que proveen un tiempo mínimo de procesamiento.
- **Eficiencia del almacenamiento**: Aquellos Atributos del software que promueven requisitos mínimos de almacenamiento durante la operación.

El modelo de McCall fue creado con el propósito de proporcionar una orientación para la adquisición de software, reduciendo la brecha entre usuarios y desarrolladores al enfocarse en un número de factores de calidad que reflejen las prioridades de ambas partes, igualmente para clasificar los atributos de calidad del software, los cuales son:

- Factores de calidad del producto
- Factores de calidad del proceso
- Factores de calidad del sistema

A continuación, mostraremos un caso real donde se haya aplicado dicho atributo en el diseño, como recordaremos la primera aparición del modelo de calidad de McCall surgió de la necesidad de proveer calidad, de manera mesurable y clara, a ciertos sistemas militares. Y que mejor manera de hablar de la calidad de McCall que usando sus propias métricas de calidad utilizadas en ese proyecto. Por supuesto tenemos que tomar en cuenta que ciertos conceptos mencionados pueden no estar presentes en el contexto actual, pero si sus equivalentes.

Notas:

- La fuente original se encuentra en Ingles, por lo tanto, las traducciones no son del todo precisas.
- Se omitió la descripción de los apartados con la leyenda (solo implementación), debido a que actualmente nos centramos en la eficiencia desde la perspectiva del diseño.

Eficiencia de ejecución

Requisitos de calidad de funcionamiento asignados al diseño (fase de diseño en el sistema nivel).

Los requisitos de rendimiento del sistema deben desglosarse y asignados adecuadamente a los módulos durante el diseño. Esta métrica simplemente identifica si los requisitos de rendimiento han sido (1) o no (0) asignados durante el diseño.

Medida de eficiencia de procesamiento iterativo (diseño e implementación)

La métrica a nivel de módulo es la suma de las puntuaciones de los siguientes elementos aplicables divididas por el número de elementos. A nivel de sistema, es una puntuación media para todos los módulos.

- **Cálculos no dependientes del bucle que se mantienen fuera del bucle.** Deben evitarse prácticas como la evaluación de constantes en un bucle. Esta medida se basa en el número de instrucciones que no dependen de un bucle que se encuentra en todos los bucles de un módulo. Esto se medirá a partir de una representación detallada del diseño durante el diseño y desde el código durante la implementación.
- **Compilador/lenguaje ensamblador de optimización del rendimiento utilizado (implementación solamente).**
- **Expresiones compuestas definidas una vez (solo implementación).**
- **Número de superposiciones.** El uso de superposiciones requiere una sobrecarga en cuanto al tiempo de procesamiento. Esta medida, la inversa del número de superposiciones, refleja que, en lo alto, se puede aplicar durante el diseño cuando el esquema de superposición se define y durante la implementación.
- **Libre de empaquetado/desempaquetado de bits/bytes en bucles.** Se trata de una medida binaria que indica la sobrecarga implicada en bit/byte de embalaje y desembalaje. Colocar estas actividades dentro de bucles debería debe evitarse si es posible.
- **Libre de código ejecutable no funcional (solo implementación).**
- **Declaraciones de decisión codificadas de manera eficiente (solo implementación).**
- **Vinculación de módulos (solo implementación, requiere ejecución).**
- **Enlaces del sistema operativo (solo implementación, requiere ejecución).**

Medida de eficiencia en el uso de datos (fases de diseño e implementación aplicadas)

La métrica a nivel de módulo es la suma de las puntuaciones de los siguientes elementos aplicables divididos por el número de elementos. La métrica del sistema es el valor promedio de todas las métricas del módulo valores.

- **Datos agrupados para un procesamiento eficiente.** Los datos utilizados por cualquier módulo deben organizarse en la base de datos, búferes, matrices, etc., de manera que se facilite el tratamiento. La organización de los datos durante el diseño y la implementación es que se examinará para proporcionar esta medida binaria.
- **Las variables se inicializan cuando se declaran (solo implementación)**
- **No hay expresiones de modo mixto (solo implementación).**
- **Elección común de unidades/tipos.** Por las razones análogas a las expuestas anteriormente (3), el presente convenio Seguido. La medida es la inversa del número de operaciones que tienen unidades o tipos de datos poco comunes.
- **Datos indexados o referenciados para un procesamiento eficiente.** No solo la organización de los datos, (1) anterior, sino el esquema de vinculación entre los elementos de datos afecta el procesamiento de manera eficiente. Se trata de una medida binaria de si la indexación utilizada para los datos fue elegida para facilitar el procesamiento.

Eficiencia de almacenamiento

Medida de eficiencia de almacenamiento (fases de diseño e implementación en el módulo) nivel primero y luego nivel de sistema).

La métrica a nivel de módulo es la suma de las puntuaciones de los siguientes elementos aplicables divididas por el número de elementos aplicables. La métrica a nivel de sistema es el valor promedio de todos los valores de métrica del módulo.

- **Requisitos de almacenamiento asignados al diseño (solo fase de diseño).** Las necesidades de almacenamiento para el sistema deben asignarse al módulo individual durante el diseño. Esta medida es una medida binaria de si dicha asignación se efectúa explícitamente (1) o no (0).
- **Instalaciones de almacenamiento virtual utilizadas.** El uso de técnicas de almacenamiento virtual o de radio búsqueda mejora la eficiencia de almacenamiento de un sistema. Esta es una medida binaria de si Estas técnicas se planifican y utilizan (1) o no (0).
- **Datos comunes definidos una sola vez (solo implementación).**
- **Segmentación del programa.** Los esquemas de segmentación eficientes minimizan la longitud máxima del segmento para minimizar el requisito de almacenamiento. Esta medida se basa en La longitud máxima del segmento. Se aplicará durante el diseño cuando Las estimaciones están disponibles y durante la implementación.
- **Segmentación de datos.** La cantidad de datos a los que hace referencia un módulo en forma de matrices, búferes de entrada, o datos globales, a menudo es pequeño en comparación con el tamaño de las áreas de almacenamiento requeridas. Esto representa una ineficacia uso eficiente del almacenamiento. La medida se basa en la cantidad de datos utilizados divididos por la cantidad total de datos disponibles para un módulo.
- **Gestión dinámica de la memoria utilizada.** Se trata de una medida binaria que pone de relieve las ventajas de utilizar Técnicas de gestión de memoria dinámica para minimizar la cantidad de almacenamiento requerido durante la ejecución. Esto se planifica durante el diseño y se utiliza durante la implementación.
- **Empaquetado de datos utilizado (solo implementación).**
- **Libre de código no funcional (solo implementación).**
- **No hay código duplicado.** El código duplicado debe evitarse por la misma razón que el punto anterior. Esta medida, que se aplicará durante el diseño y la implementación, es en función de la cantidad de código duplicado.
- **Compilador/lenguaje ensambladora de optimización del almacenamiento utilizado (implementación solamente).**
- **Libre de elementos de datos redundantes (solo implementación).**

A manera de anexo se muestran los artefactos utilizados para la evaluación de las métricas anteriores, así como las fórmulas matemáticas utilizadas:

Table 6.2-1 Software Quality Metrics (Continued)

CRITERION/ SUBCRITERION	METRIC	FACTOR(S): EFFICIENCY					
		REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE
EXECUTION EFFICIENCY/ REQUIREMENTS	EE. 1 PERFORMANCE REQUIREMENTS ALLOCATED TO DESIGN			☐			
	SYSTEM METRIC VALUE = Same as line above				☐		
ITERATIVE PROCESSING	EE. 2 ITERATIVE PROCESSING EFFICIENCY MEASURE: (by module)						
	(1) Non-loop dependent computations kept out of loop. $\left(1 - \frac{\#\text{ nonloop dependent statements in loop}}{\text{total \# loop statements}}\right)$				☐		☐
	(2) Performance optimizing compiler/assembly language used.					☐	
	(3) Compound expressions defined once. $\left(1 - \frac{\#\text{ compound expression defined more than once}}{\#\text{ compound expressions}}\right)$						☐
	(4) Number of overlays. $\left(\frac{1}{\#\text{ of overlays}}\right)$				☐		☐
	(5) Free of bit/byte packing/unpacking in loops.			☐		☐	
	(6) Free of nonfunctional executable code. $\left(1 - \frac{\#\text{ nonfunctional executable code}}{\text{total executable statements}}\right)$						☐
	(7) Decision statements efficiently coded. $\left(1 - \frac{\#\text{ inefficient decision statements}}{\text{Total \# decision statements}}\right)$						☐

Table 6.2-1 Software Quality Metrics (Continued)

CRITERION/ SUBCRITERION	METRIC	FACTOR(S): EFFICIENCY					
		REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE	YES/NO 1 OR 0	VALUE
STORAGE EFFICIENCY	SE. 1 STORAGE EFFICIENCY MEASURE: (by module)						
	(1) Storage requirements allocated to design.			☐			
	(2) Virtual storage facilities used.			☐		☐	
	(3) Common data defined only once. $\left(1 - \frac{\#\text{ variables defined more than once}}{\text{total \# variables}}\right)$						☐
	(4) Program segmentation. $\left(1 - \frac{\text{maximum segment length}}{\text{total program length}}\right)$				☐		☐
	(5) Data segmentation. $\left(1 - \frac{\text{Amount of unused data}}{\text{total amount of data}}\right)$				☐		☐
	(6) Dynamic memory management utilized.			☐		☐	
	(7) Data packing used.					☐	
	(8) Free of nonfunctional code. $\left(1 - \frac{\#\text{ nonfunctional statements}}{\text{total \# statements}}\right)$						☐
	(9) no duplicate codes. $\left(1 - \frac{\#\text{ duplicate statements}}{\text{total \# statements}}\right)$				☐		☐
	(10) Storage optimizing compiler/assembly language used.					☐	
	(11) Free of redundant data elements. $\left(1 - \frac{\#\text{ redundant data elements}}{\#\text{ data elements}}\right)$						☐
	MODULE METRIC VALUE = $\frac{\text{total score from applicable elements}}{\#\text{ applicable elements}}$				☐		☐
	SYSTEM METRIC VALUE = $\frac{\text{sum of storage efficiency measures for each}}{\text{total \# modules}}$				☐		☐

A quedado claro que gran parte de la eficiencia recae en la implementación, sin embargo, desde el diseño podemos "implementar" ciertos mecanismos que permitan que nuestro algoritmo, independientemente de la plataforma, sea más eficiente. Un método es no forzar el **"Predictor de saltos".**

Explicación:

Una instrucción de salto es una instrucción en un programa o código que se utiliza para alterar el flujo normal de ejecución. Cuando se encuentra una instrucción de salto, el programa se desvía de forma no lineal hacia una dirección de memoria específica donde se encuentra otra instrucción que será ejecutada a continuación.

El predictor de saltos es un mecanismo que permite predecir el resultado de una instrucción de salto en un procesador, con el objetivo de optimizar los tiempos de ejecución. Esto se consigue mediante la recopilación de estadísticas que miden la frecuencia y la dirección del salto para cada ubicación en el código. La información se almacena en una tabla para su posterior uso y mejorar los tiempos de ejecución.

El objetivo del predictor de saltos es predecir con anticipación el resultado de una instrucción «salto» en un procesador antes incluso de que se ejecute para optimizar los tiempos de ejecución.

En resumen:

Todo lo anterior suena muy confuso, pero en términos más sencillos se refiere a que mientras menos condicionales, llamadas de funciones e instrucciones GOTO existan, el código será más rápido ya que no tendrá que frenar para cambiar de sentido. Existen ciertas situaciones donde es imposible no usar tantas condicionales, sin embargo, se debe evitar siempre que sea posible. Lo anterior promueve mayor simplicidad del flujo general del programa, lo que se traduce en menos instrucciones que se deben ejecutar.

Un ejemplo tangible dentro del proyecto es el siguiente:

Cuando estamos "minando" la experiencia, existen 2 casos de experiencia, la "simple" y la "compuesta". Los 2 anteriores casos poseen un procesamiento particular.

La solución lógica es iterar sobre la tabla y decir si es caso simple: "minar caso simple", sino "minar caso compuesto". El problema aquí es que por cada iteración existirá una parada, las cuales solo entorpecen el rendimiento y el flujo continuo.

La respuesta que nosotros planteamos es la siguiente, iterar sobre la tabla y determinar de qué tipo son (Pero no se procesan). Después mandar el "conjunto" de tipo "simple" a una función dedicada al tipo "simple", lo mismo con el otro caso, lo anterior nos permite que, en el proceso pesado, ósea "el minado de datos" no existan paradas, ya que como todos los datos son del mismo caso no se requiere detenerse hasta el final del ciclo. Lo que proporciona mayor fluidez.

Métricas para el proyecto:

Para la definición de métricas relacionadas con la fiabilidad nos basaremos en las métricas proporcionadas por McCall (adaptadas al contexto de nuestro programa), así como otras cuestiones que consideramos pertinentes, el método de evaluación es simple si se cumple (Cabalmente sin excepciones), se le asigna un puntaje de 1, en caso contrario se le asigna una puntuación de 0.

Eficiencia de ejecución

Criterio	**Métrica**	**¿Se cumple?**
Eficiencia de ejecución	Mientras se ejecuta el programa, la utilización total del CPU, es igual o menor al 40% de su capacidad total	
Eficiencia de ejecución	El tiempo de minado de un “link de usuario” no debe superar los 8 segundos.	
Eficiencia de ejecución	El tiempo de minado de un “perfil completo” no debe superar el minuto con 30 segundos.	
Eficiencia de ejecución	La actualización de cookies de inicio de sesión es menor a 30 segundos. (Iniciar Sesión).	
Eficiencia de ejecución	El tiempo de intercambio entre perfil y perfil no supera los 8 segundos. (Copiar y pegar otra URL)	
Eficiencia de ejecución	La actualización de cookies solo se realiza cuando es necesario (Las cookies expiraron, se desea ingresar con otro usuario)	
Eficiencia de ejecución	El tiempo de arranque del programa no supera los 10 segundos.	
Eficiencia de ejecución	Todo el código en el proyecto es funcional.	
Eficiencia de ejecución	La complejidad algorítmica de cada método debe ser igual o menor a Big-O^2.	
Eficiencia de ejecución	Los tipos de datos deben ser homogéneos (Cuando corresponda)	
Eficiencia de ejecución	El tiempo de transición entre la pagina de inicio y la página de búsqueda es igual o menor a 8 segundos.	

Eficiencia de almacenamiento

Criterio	Métrica	¿Se cumple?
Eficiencia de almacenamiento	El tiempo de lectura de los archivos locales no supera los 2 segundos.	
Eficiencia de almacenamiento	El tamaño de los archivos locales no supera el KB de peso.	
Eficiencia de almacenamiento	La conexión a la base de datos tarda menos de 4 segundos.	
Eficiencia de almacenamiento	El tiempo de subida de un documento (Objeto), tarda menos de 2 segundos	

Eficiencia de almacenamiento	Mientras se ejecuta el programa, la utilización total de la memoria RAM, es igual o menor al 40% de su capacidad total	

Nota: Todas las medidas numéricas anteriormente mencionadas fueron tomadas en base al tiempo real en que una persona promedio le toma realizar ciertas acciones manualmente (las que correspondan). La idea en general es que el tiempo de realizar dichas acciones sea estrictamente menor al que le tarda a un ser humano realizar dicha tarea.

Referencias:

EmployingMcCallsmethodandtheISO-9126toasseshequalityofISbyusers.pdf

10 métricas de rendimiento de aplicaciones y cómo medirlas | Computer Weekly

c++ - ¿Por qué se procesa más rápido un array ordenado que uno desordenado? - Stack Overflow en español

Predictor de saltos (Branch prediction) - Qué es, definición y concepto - Muy Tecnológicos (muytecnologicos.com)

https://apps.dtic.mil/sti/pdfs/ADA049014.pdf

https://www.researchgate.net/profile/Ming-Chang-Lee-2/publication/263939913_Software_quality_factors_and_software_quality_metrics_to_enhance_software_quality_assurance_BJAST/data/00b4953c64d6e6be8b000000/Software-quality-factors-and-software-quality-metrics-to-enhance-software-quality-assurance-BJAST.pdf