NOMBRE ESTUDIANTE:	CÓDIGO DE ESTUDIANTE:
Cristian Mateo Alvarez Posada	**2220211010**

Realice una consulta de los siguientes puntos:

- ¿Qué es la crisis del software?
- ¿Por qué nace realizar un análisis del software antes de la etapa de codificación?
- Describa 5 ejemplos de crisis del software por falta de planeación
- ¿Como futuro profesional en el área de software, cuál es su papel para minimizar esta problemática tan difícil de cambiar en los desarrolladores?

Entregue un archivo en formato pdf. No olvide que las referencias deben ser académicas (ejemplo:google academic). Consulte papers en inglés.

¿Qué es la crisis del software?:

“La crisis del software es el resultado de la introducción de la tercera generación del hardware. Es un hecho de que el software que se construye no solamente no satisface los requerimientos ni las necesidades del cliente, sino que además excede los presupuestos y los horarios.”

“La crisis se caracterizó por los siguientes problemas:

- Funcionalidad incorrecta.
- **Desarrollo y mantenimiento inadecuados**.
- Insatisfacción de la demanda.
- Imprecisión en la **planificación del proyecto** y estimación de los costos.
- Baja calidad del software.
- **Dificultad de mantenimiento** de programas con un **diseño poco estructurado**, difícil y costoso.
- Por otra parte, se exige que el software sea eficaz y barato tanto en el desarrollo como en la compra.
- **Carencia de información sobre qué realizamos y cómo**.
- Insatisfacción de clientes y usuarios.
- Calidad sospechosa.

(Tema 1.2 Crisis del software - Ingeniería de software - Instituto Consorcio Clavijero, s. f.)

“In the 1960s, people started to realize that programming techniques had lagged behind the developments in software both in size and complexity. To many people, programming was still an art and had never become a craft. An additional problem was

that many programmers had not been formally educated in the field. They had learned by doing. On the organizational side, attempted solutions to problems often involved adding more and more programmers to the project, the so-called ‘million-monkey’ approach.

As a result, software was often delivered too late, programs did not behave as the user expected, programs were rarely adaptable to changed circumstances, and many errors were detected only after the software had been delivered to the customer.

This became known as the ‘software crisis’. This type of problem really became manifest in the 1960s. Under the auspices of NATO, two conferences were devoted to the topic in 1968 and 1969 (Naur and Randell, 1968), (Buxton and Randell, 1969). Here, the term ‘software engineering’ was coined in a somewhat provocative sense. Shouldn’t it be possible to build software in the way one builds bridges and houses, starting from a theoretical basis and using sound and proven design and construction techniques, as in other engineering fields?”

(van Vliet, 2007, p. 3)

¿Por qué nace realizar un análisis del software antes de la etapa de codificación?:

“When building a house, the builder does not start with piling up bricks. Rather, the requirements and possibilities of the client are analyzed first, taking into account such factors as family structure, hobbies, finances and the like. The architect takes these factors into consideration when designing a house. Only after the design has been agreed upon is the actual construction started. It is expedient to act in the same way when constructing software. First, the problem to be solved is analyzed and the requirements are described in a very precise way. Then a design is made based on these requirements. Finally, the construction process, i.e. the actual programming of the solution, is started. There are a distinguishable number of phases in the development of software.

Software engineering concerns methods and techniques to develop large software systems. The engineering metaphor is used to emphasize a systematic approach to develop systems that satisfy organizational requirements and constraints. This chapter gives a brief overview of the field and points at emerging trends that influence the way software is developed.”

(van Vliet, 2007, p. 10)

Describa 5 ejemplos de crisis del software por falta de planeación:

- Gracias a una planeación pobre, es posible que un software tenga que ser “desensamblado” para corregir una entrada de datos, un proceso o una salida que no fue tomada en consideración.

- Debido a que la planeación del software fue deficiente, la escalabilidad del software se vuelve cada vez más complicada.
- En vista de que la planeación del software fue errónea, el software resuelve un problema diferente al problema que el cliente quería resolver.
- Se pueden generar una serie de incongruencias si el software es de grandes magnitudes y no cuenta con una planificación que guíe y/o permita verificar que el software se está estructurando como debería.
- Al no haber presencia de una buena planeación se dificulta el trabajo al interior del grupo o equipo de trabajo encargado de llevar a cabo el resto de las fases de desarrollo del software.

¿Como futuro profesional en el área de software, cuál es su papel para minimizar esta problemática tan difícil de cambiar en los desarrolladores?

Mi papel para minimizar esta problemática, como futuro profesional del área de desarrollo de software, consistiría en reconocer que a pesar de existe un afán creciente por aprender nuevas tecnologías y/o lenguajes de programación, es de gran importancia aprender y reforzar conocimientos relacionados con la “aplicación de la ingeniería al software”, explorando las posibilidades que se pueden llegar a tener al aplicar dichas técnicas “ingenieriles” a la resolución de problemas de software, y también reconociendo la infinidad de errores que se pueden evitar gracias a una buena ejecución en las fases de desarrollo de software

Bibliografía:

Tema 1.2 Crisis del software—Ingeniería de software—Instituto Consorcio Clavijero. (s. f.). Recuperado 24 de marzo de 2022, de https://cursos.clavijero.edu.mx/cursos/179_is/modulo1/contenidos/tema1.2.html?opc=1

van Vliet, H. (2007). Software engineering: Principles and practice.