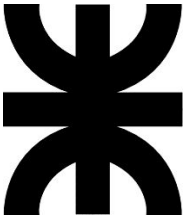

UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD

REGIONAL DE CÓRDOBA

INGENIERÍA DE SOFTWARE

LA CRISIS DEL SOFTWARE

Docentes:

- COVARO, Laura Inés
- ROBLES, Joaquín Leonel
- BELLI, Giuliana

Grupo N° 3:

- BACCA, Josue 72156
- BARAFANI, Francisco 72750
- GROBER, Luciana 72409
- MARIATTI, Marcos 72707
- VENTURI, Agustin 71943

Comisión: 4k4
Fecha de reentrega: 24/05/19

Índice

Resumen

En este informe técnico nos interiorizamos en el formato que posee el mismo con el fin de desarrollar un tema determinado, y entendiendo que es una herramienta importante para cualquier profesional a la hora de publicar información de manera formal y el cual debe ser comprendido fácilmente. En él desarrollaremos lo que se denomina "Crisis de software", que dió como resultado grandes problemas, principalmente de calidad que se hacía notable al momento de desarrollarlo.

Nos enfocamos en las causas de esta crisis para poder entender el porqué de las dificultades del desarrollo de software que no permitían que este crezca, se modifique, sea reutilizable, y, principalmente, que se adapte al entorno cambiante.

Una vez identificadas estas causas hablaremos sobre la disciplina que surgió como la posible solución a los problemas inherentes de la "Crisis del software", la "Ingeniería de software", que a través de nuevas técnicas y metodologías busca impedir que el anteriormente nombrado caiga en un estado en donde el esfuerzo de adaptación del sistema sea más alto que los beneficios que se esperan de este.

Introducción

“La crisis del Software” ,así como fue nombrada por primera vez en el año 1969 por el informático Friedrich L Bauer, hace referencia a los problemas que ha ido experimentando el software desde su comienzo hasta la actualidad generalmente como consecuencia de errores ocurridos en la planificación, la productividad y la estimación de los costos, afectando directamente la calidad del mismo.

En los comienzos el hardware y el software estaban a la par, siendo este último desarrollado de manera informal, sin especificaciones ni documentación. Debido a la carencia de técnicas y metodologías para el desarrollo de software, los proyectos comenzaron a complejizarse ya sea por el incremento de la demanda o el tamaño de los mismos dado que el crecimiento del hardware y la capacidad de cómputo nos lo permitían, otorgando como resultado proyectos entregados fuera de tiempo, fuera del presupuesto e incluso con productos que nunca llegaban a ser utilizados.

Esta crisis debía ser resuelta de manera urgente y es así como nace una posible solución: la ingeniería de software, una disciplina que se enfoca en todos los aspectos del desarrollo de software, incluyendo nuevas nuevas técnicas, metodologías y herramientas, con el fin de desarrollar un buen software que sea flexible, evolutivo y reutilizable.

Esta logró mejorar la calidad de software y la entrega del producto requerido en el tiempo acordado y bajo el presupuesto definido. Pero no siempre, por esto nos preguntamos: ¿Es posible evitar todos los problemas a la hora de desarrollar software? ¿A caso el software no siempre tiene problemas debido a su complejidad inherente?

Desarrollo

CAUSAS DE LA CRISIS

Para adentrarnos por completo en lo esencial de esta crisis es necesario hacer referencia al contexto donde estaba involucrado el desarrollo de software.
Comenzamos por la década de los 50, principios de los 60, donde la potencia de las máquinas computacionales era limitada en su gran mayoría, por lo que los programas a desarrollar eran medianamente simples si los comparamos con la actualidad, no tenían ninguna metodología o pasos a seguir para llegar al objetivo y solía usarse para su desarrollo un lenguaje de bajo nivel. Pero a finales de los 60, la potencia de las máquinas comenzó a aumentar exponencialmente, aparecieron lenguajes de programación de alto nivel y esto condujo a demandar software más complejo y de mayor tamaño. El problema fue que el avance de hardware no era equitativo con el avance del software, produciendo un gran déficit en cuanto a herramientas y técnicas para llegar a desarrollar el software que se requería. Así explicaba esta situación Edsger W. Dijkstra en su libro The Humble Programmer *"... el poder disponible de las máquinas creció en un factor de más de mil, la ambición de la sociedad para utilizar estas máquinas creció proporcionalmente, y era el pobre programador que encontró su trabajo en este campo explotado de tensión entre fines y medios*".

En ese entonces la importancia que se le daba a los procesos de desarrollo o al proyecto en sí, muchas veces era mínimo y en otras inexistente, solo se dedicaba un 25% del total del proyecto al análisis y desarrollo del producto que se quería lograr, incluyendo allí especificaciones, restricciones, pruebas, entre otras actividades, por ende el 75% restante se dedicaba a la corrección y el mantenimiento de los errores proveniente de errores de código, de especificaciones, etc. que hacian que el software se desgaste aún mas.
Los problemas se hacían notables a la hora de entregar el software ya que en las mayorias de las veces se hacía fuera de la fecha establecida, fuera del presupuesto acordado, sin las funcionalidades requeridas, dando como resultado un software de mala calidad.

Pero no solo el contexto en donde se desarrollaba el software y como se lo llevaba a cabo eran los únicos causantes de la crisis. Fred Brooks, en su libro "*No Silver Bulle*t" (1986)

definió dos tipos de complejidad a la hora de desarrollar software, la complejidad accidental y la complejidad esencial. La accidental se refiere a los problemas que creamos nosotros mismos y pueden ser corregidos, por ejemplo los errores de sintaxis o en las situaciones donde se seleccionan caminos de complejidad mayores que los necesariaos. Por el otro lado la complejidad esencial hace referencia a las situaciones donde todas las soluciones razonables a un problema son complicadas, al punto de ser imposibles, es decir que el problema posee una complejidad inherente y no puede ser simplificado o evitado. Entonces podemos afirmar que estos tipos de complejidades fueron importantes causantes de la crisis, y siguen repercutiendo en el desarrollo actual. Aunque la complejidad accidental puede ser evitada, la esencial es inevitable y seguirá existiendo mientras haya software.

LA CRISIS DEL SOFTWARE

En 1968 como consecuencia de lo que estaba ocurriendo con el desarrollo de software con poco desempeño y baja calidad, se denominó a esta etapa “La crisis del software” que no fue más que formalizar lo que ya había empezado a ocurrir a principios de la década del 60.

Durante la conferencia de la OTAN se definió la lista de los principales problemas que caracterizaban el software durante la crisis: (Extracto de *Improving Software Development Productivity* - Randall W. Jensen)

- No Confiables
- Entregados tarde.
- Costos prohibitivos en términos de modificación
- Imposibles de mantener.
- Funcionan en un nivel inadecuado.
- Exceden los costos del presupuesto.

La crisis de software se caracterizó principalmente por la calidad de los proyectos. Esta es definida por el alcance, tiempo y presupuesto, factores que no cumplian ninguno de los trabajos que se estaban desarrollando en el momento. Como consecuencia, terminaban siendo cancelados o, en la mayoría de los casos, entregados pero nunca usados. Además, quienes sí lo usaban, no lo usaban tal cual les fue entregado.

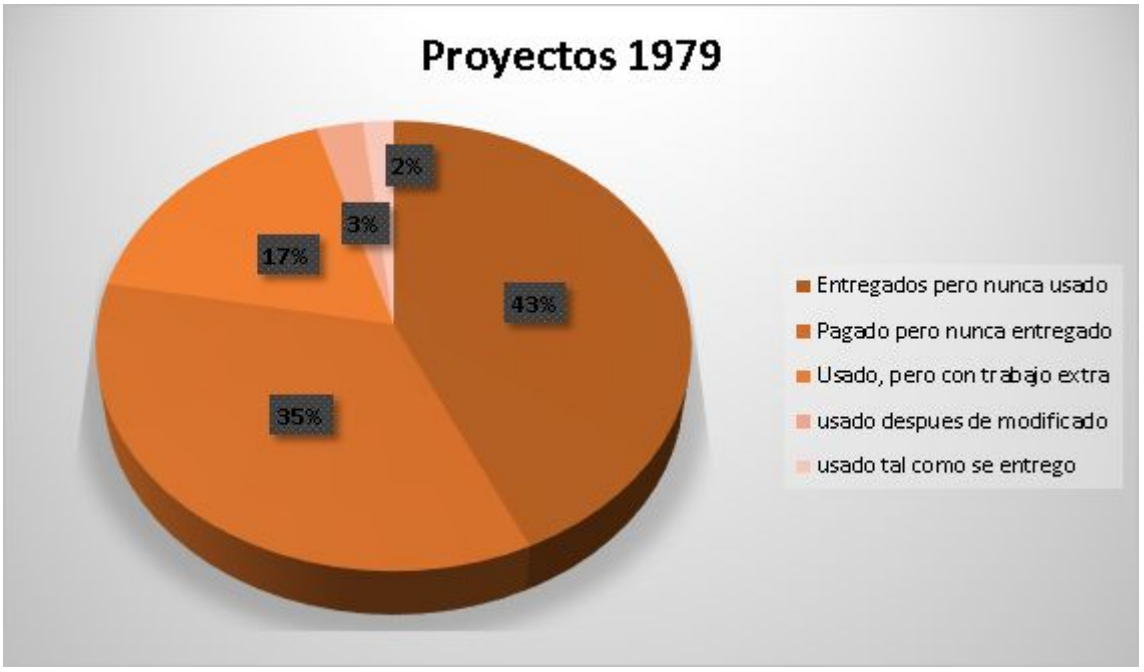

Gráfico 1 - proyectos 1979

En fin, el ámbito de software requería un cambio de enfoque significativo, no solo vinculado a funcionalidades inadecuadas sino teniendo en cuenta también problemas relacionados con cómo desarrollar software, como aumentar su calidad y mantenerla a medida que crece el tamaño de software existente y cómo estar preparado para satisfacer las nuevas demandas de software, entre otras.

SOLUCIÓN: LA INGENIERÍA DE SOFTWARE

Al ver que se estaba llegando a un punto crítico de no retorno que podría acabar con toda la industria de software se decidió buscar una posible solución, que se denominó “ingeniería de software”.

“La ingeniería de software es una disciplina que integra métodos, herramientas y procedimientos para el desarrollo de SW de computador” R.Pressman.
Ian Sommerville remarca que el foco de esta disciplina es siempre encontrar soluciones a problemas, pero que estas soluciones deben adaptarse a restricciones tanto organizacionales como financieras.

Podemos decir entonces que la Ingeniería del Software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software de calidad y cuyas principales actividades son la especificación, desarrollo, validación y la evolución del

software. Intenta dividir el proceso de desarrollo de software, definiendo pautas o reglas a seguir con el fin de reducir los errores y aumentar la calidad del software que se brinda, minimizando tiempo, costos de desarrollo, esfuerzo, entre otros.

Es la solución a la crisis ya que su principal objetivo es desarrollar un software de calidad, que es aquel que cumple con los requisitos funcionales y de rendimiento establecidos previamente y consta de unos estándares de desarrollo bien documentados. Además, incorpora al proceso nuevos modelos de desarrollo y modificación del ciclo de vida, nuevos paradigmas de programación, entre otros; que hacen que el desarrollo de software sea mucho más metódico y estructurado, disminuyendo así notablemente fallos y correcciones costosas.

Ian Sommerville también destaca dos razones principales por la que la ingeniería de software es tan importante:

- Cada vez con mayor frecuencia los individuos y la sociedad se apoyan en los avanzados sistemas de software. Por ende, se requiere producir económica y rápidamente sistemas confiables.
- A menudo resulta más barato a largo plazo usar métodos y técnicas de ingeniería de software para los sistemas, que sólo diseñar los programas como si fuera un proyecto de programación personal. Para muchos de los sistemas, la mayoría de los costos consisten en cambiar el software después de ponerlo en operación.

Estudios realizados en el año 1991 para cuantificar el impacto de estas medidas comparándolas contra las estadísticas del año 1979 arrojaron resultados muy buenos.

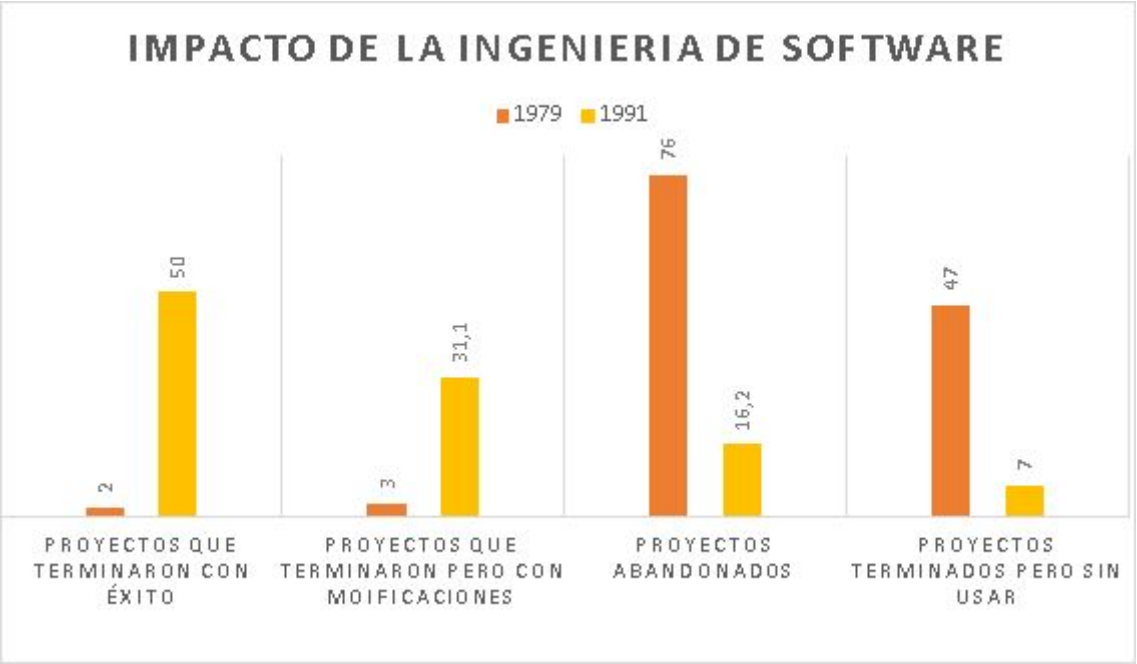

Gráfico 2 - impacto de la ingeniería de software.

Los resultados de los estudios no hicieron más que confirmar que los estándares y protocolos que definió la ingeniería en software era el mejor camino para el desarrollo de software, ya que tiene en cuenta todos los aspectos de producción, ya sea técnicas, de producción y de soporte.

LA INGENIERÍA DE SOFTWARE, HOY

La ingeniería de Software pudo acabar con la Crisis del Software, pero no quiere decir que no existan más problemas hoy en dia en cuanto al desarrollo. El software está presente en todos lados, desde las grandes organizaciones hasta nuestros bolsillos. La expansión de la demanda de software que llegó hasta en las organizaciones más pequeñas generó la necesidad de soluciones de software baratas, las cuales lideraron el crecimiento de metodologías más rápidas y simples. Estas desarrollan software ejecutable desde los requerimientos hasta la implementación de una forma más fácil y rápida. Actualmente los proyectos de software adoptan metodologías ágiles como Scrum o XP(Extreme Programming), las cuales intentan simplificar muchas áreas de la ingeniería, desde la especificación de requerimientos hasta el testing. Aun así los sistemas de software muy grandes siguen utilizando metodologías fuertemente documentadas, con muchos volúmenes en la documentación.

Conclusión

Gracias a que la crisis de software puso en evidencia los problemas graves del desarrollo de software es que nace la Ingeniería del Software. Esta es una disciplina que tiene en cuenta todos los aspectos de la producción de software, solucionando problemas en la mayoría de sus etapas pero principalmente en la de análisis y diseño. Además, la Ingeniería del Software ayuda a paliar los inconvenientes de los involucrados con respecto a técnicas y metodologías que requerían diferentes habilidades para ser utilizadas y así poder desarrollar el software complejo que se demandaba.

En nuestra opinión y basándonos en evidencias de estudios, concluímos en que la producción del software va mucho más allá que los conocimientos técnicos sobre tecnologías y habilidades de programación, diseño, análisis, etc. Estos aspectos proporcionan una base para el desarrollo de software pero como hemos estudiado, no son suficientes para los cada vez más complejos, grandes, diversos y sobre todo cambiantes sistemas de software.

Consideramos que la “Crisis del software” fue el momento en el tiempo en que esto fue comprendido y se propuso el objetivo de encontrar la solución, la “Ingeniería de software”. La importancia de esta disciplina es que el principal elemento para producir software son las personas, y para que estas puedan elaborar software a la medida del entorno cambiante en el que operan los sistemas, bajo distintas restricciones organizacionales y financieras es necesario una disciplina que asegure cumplir dicho objetivo.

Aun así, hoy en día siguen habiendo algunos problemas con el desarrollo de software en muchos proyectos, ya sea por cuestiones de costos, tiempo, competencia, y demás. Todo el esfuerzo de desarrollo se enfoca en el análisis, diseño, programación y prueba, es decir, en llegar a cumplir con el alcance del producto, y se dejan de lado las prácticas, métodos y gestión de equipo, que permiten que el software llegue a la calidad deseada y pueda crecer sin afectar su consistencia.

Creemos que el mejor camino para lograr estas metas es tratar de inculcar la ingeniería de software como una parte tan importante en el desarrollo como lo es la programación, las pruebas, etc; y tenerla en cuenta desde los primeros momentos de la producción, y no solo cuando se hacen evidentes las carencias del producto en cuanto a calidad y crecimiento.

Afirmamos que fue y es correcto definir a la Ingeniería de Software como la solución a la crisis del software ya que posee herramientas, metodologías y técnicas para realizar un correcto desarrollo de software que garantice la calidad del mismo,a través de los factores

de tiempo, presupuesto y alcance, asegurando un software abstracto, intangible y fácil de modificar.

Anexo

Para comprender el modelo de informe técnico se utilizaron las siguientes páginas:

- “Informe técnico”, Lucecita Bells https://www.academia.edu/5091078/INFORME_T%C3%89CNICO_LA_ESTRUCTURA_DEL_INFORME_T%C3%89CNICO_EST%C3%81_FORMADA_POR_La_parte_inicial
- “Cómo redactar un informe técnico”, Universidad Nacional Autónoma de Mexico http://www.ingenieria.unam.mx/~especializacion/egreso/Como_redactar_un_informr_tecnico.pdf
- “Basic Technical Report”, Maplesoft https://www.maplesoft.com/support/help/Maple/view.aspx?path=Task/BasicTechnicalReport
- pdf- modelo de informe técnico-secretaría de ciencia y tecnología.

Bibliografía

- “The Software Crisis - Georgia Tech - Software Development Process”, Udacity https://www.youtube.com/watch?v=0b5vp4Z2PKE
- “Historia del software. La crisis del software”, Algoritmo y Programa https://sites.google.com/site/algoritmoyprograma/4-2-historia-del-software-la-crisis-del-software
- “Crisis del Software”, Scrum manager https://www.scrummanager.net/bok/index.php?title=Crisis_del_software
- “What is software crisis? ”, Ashwin Dhakal https://www.quora.com/What-is-software-crisis
- “La Crisis del Software”, Universitat Politecnica de Valencia https://histinf.blogs.upv.es/2011/01/04/la-crisis-del-software/
- “Evidence of the Software Crisis - Georgia Tech - Software Development Process” https://www.youtube.com/watch?v=Cd3TrUK8axU
- “Crisis del Software”, Armando Acuña Martinez https://issuu.com/armaacum/docs/crisis_del_software
- Ingeniería de software- 7ma edición -Sommerville, Ian
- No Silver Bullet - Frederick Brooks http://worrydream.com/refs/Brooks-NoSilverBullet.pdf
- The Humble Programmer - Edsger W. Dijkstra https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html
- Jared King’s - The History of Software Engineering https://learn.saylor.org/mod/page/view.php?id=12353
- “Improving Software Development Productivity”, Randal W. Jensen http://ptgmedia.pearsoncmg.com/images/9780133562675/samplepages/9780133562675.pdf