* Diapositiva 1: Introducción (Alexis)

NO SILVER BULLETS - NO HAY BALAS DE PLATA. Artículo científico sobre ingeniería de software presentado en 1986 por Fred Brooks, ingeniero de software y científico de la computación.

* Diapositiva 2: Significado e Hipótesis (Alexis)

El análisis de Brooks gira en torno a las dificultades esenciales presentes en el desarrollo de software, presentando también históricos intentos de solución y alternativas factibles. El nombre del ensayo es una alusión a una hipotética “mágica solución”, rápida y efectiva con la que pueden afrontarse los monstruos del software. Brooks expresa que no existe a simple vista tal “bala de plata” ////// que garantice mejoras, en órdenes de magnitud, como la hay en dificultades accidentales. (ver ley de moore)

* Diapositiva 3: Dificultades (Alexis)

Hay dos tipos de dificultades en el desarrollo de software. Las dificultades esenciales son aquellas inherentes a la naturaleza del software: relacionadas con la complejidad del problema que se desea resolver. No pueden eliminarse. Y las dificultades accidentales, que sus causas se presentan durante el desarrollo pero sobre cuestiones referidas a hardware, performance, flujos de procesamiento, etc. ///Avanzan a un paso acelerado, en una industria bien establecida.

* Diapositiva 4: Complejidad (Alexis)

El software no repite partes semejantes, las unifica en un único elemento. Intrínsecamente, incrementa el número de elementos y sus interacciones. Esto aumenta la complejidad de forma usualmente no lineal. De aquí deriva la dificultad en la comunicación, la poca confiabilidad, estructuras poco adaptables al cambio, falta de integridad conceptual, retrasos en entregas, etc.: todos problemas esenciales del desarrollo de software.

* Diapositiva 5: Conformidad (Marcos)

Consiste en la dificultad que tiene el software en adaptarse a otros sistemas, interfaces y el entorno. En cualquier caso, deberá ser el software el que se deba adaptar al contexto y nunca al revés. El rediseño de interfaces provistas es muy costoso. Todo esto aumenta la complejidad del problema ya que no solo se debe dar solución al mismo con sus arbitrariedades, sino que conformar con todo elemento con el que interactúe.

* Diapositiva 6: Modificabilidad (Marcos)

En un proyecto, es la funcionalidad la que a menudo cambia, y el software  
equivale a la funcionalidad. Las manufacturas no cambian casi nunca una vez fabricadas. El software, en cambio, sufre cambios estando en funcionamiento. Estos no pueden evitarse: requerimientos, elementos con los que interactúa, hasta el entorno. Pueden requerirse modificaciones al producto en todo momento.

* Diapositiva 7: Invisibilidad (Marcos)

El software es incapaz de ser percibido. Es intangible y no puede visualizarse. No hay representaciones geométricas, ya que no está restringido en el espacio. Por lo tanto, no es fácil plasmarlo en diagramas, mucho menos a uno, en el cual apreciar sus detalles. No siempre puede alcanzarse un modelo jerárquico, sino más bien múltiples interrelaciones. La carencia de herramientas conceptuales aumenta la complejidad y dificulta la comunicación en los grupos de trabajo.

* Diapositiva 8: Lenguajes de Alto Nivel (Pia)

La evolución hacia los lenguajes de alto nivel fue el avance más efectivo a la productividad, confiabilidad y simplicidad del software en sus comienzos. Elimina gran parte de la complejidad accidental, abstrayendo preocupaciones técnicas del programador, siendo no inherentes al problema. En la actualidad ha aumentado el esfuerzo intelectual del usuario para utilizar la herramienta.

* Diapositiva 9: Time-Sharing (Pia)

Concepto casi obsoleto en la actualidad. Su mejor equivalente es el aumento de productividad de grupos de trabajo, con correcto uso de los recursos e inmediatez en sus actividades. Debe restringirse al máximo el tiempo muerto que pueda nublar la visión global de la complejidad del problema. Los tiempos de respuesta son tan mínimos actualmente que no deberían esperarse beneficios.

* Diapositiva 10: Sistemas de Programación Unificados (IDEs) (Pia)

Los IDE tienen amplio uso en la actualidad y han mejorado perceptiblemente la productividad. Esto se debe a que atacan las dificultades accidentales que resultan de usar programas individuales en conjunto, entregando bibliotecas integradas, formatos unificados de archivos y fácil desarrollo de herramientas y frameworks.

* Diapositiva 11: Ada, Lenguajes de Alto Nivel y POO (Tomas Saint Avit)

Existen lenguajes que impulsan metodologías de desarrollo, compuestas de un conjunto de buenas prácticas, como lo hizo originalmente Ada. El aplicar estas metodologías permite un salto en el la calidad del software desarrollado, objetivo más allá de los avances gramaticales de lenguajes.

La POO se caracteriza por su capacidad de llevar los modelos generados en el diseño a código con una trazabilidad muy clara y mejor abstracción de los objetos.

No reducen la complejidad esencial del problema mismo, pero provee mayor libertad para la expresión del diseño.

* Diapositiva 12: Inteligencia Artificial y Sistemas Expertos (Tomas Saint Avit)

La inteligencia artificial tiene dos definiciones. Puede definirse como intentos de resolver problemas en su totalidad, atacando dificultades esenciales, aunque lo que en verdad se ataca son problemas accidentales. Son herramientas: reconocimiento de voz y reconocimiento de imágenes.

Luego están los sistemas expertos. Estos pretenden transmitir conocimientos a quien los utiliza, desarrollados a partir de recolectar experiencia de expertos. Su dificultad depende de encontrar a dichos expertos, y definir cómo aplicar sus conocimientos: optimización, sugerencias, reglas de inferencia, asesoría de pruebas, etc.

* Diapositiva 13: Programación Automática (Tomas Saint Avit)

La programación automática es el avance más esperado, pero está ligado a una confusión general. Debe especificarse el método de solución y no el problema mismo. Por lo tanto, sólo es aplicable a casos con propiedades convenientes, como número reducido de parámetros, múltiples alternativas para hallar solución y reglas claras para su selección. Ej. modelos matemáticos. No es generalizable para cubrir la dificultad esencial en todos los contextos.

* Diapositiva 14: Programación Gráfica y Verificación de Programas (Tefa)

Presentar gráficamente los componentes del software no contribuye a la simplicidad. Son muchos, interconectados, a veces sin jerarquía y abstractos. Analizar diagramas en conjunto es imposible por el tamaño de la visualización requerida. Se dificulta hallar una visión global e implementar en su conjunto.

La verificación de programas desde el diseño conlleva mucho trabajo y no garantiza eliminar la tarea de testing, sólo reduce la carga (ya que siempre van a aparecer errores). Además, establece compleción de requerimientos definidos, no completitud de requerimientos obtenidos, que es donde recae la dificultad esencial.

* Diapositiva 15: Herramientas y Estaciones de Trabajo (Tefa)

Mejores ambientes de programación ya han resuelto las dificultades accidentales. Los mejores avances pueden observarse en los repositorios, mejorando la cooperación. Más allá de esto, provee a la comodidad y confiabilidad del programador, pero no en mucho más que evitar errores sintácticos en código. De igual manera, mejores estaciones de trabajo no resuelven el problema. El programador no está limitado por los tiempos de respuesta, sino por el tiempo para pensar. Los beneficios son marginales.

* Diapositiva 16: Buy vs Build (Tefa)

El costo de cualquier software en esencia es el gasto en desarrollo, el costo del trabajo intelectual del equipo es mucho mayor al precio que tendrá replicar algo ya construido.

Desarrollar un producto permite cumplir los requisitos de manera personalizada. Sin embargo, adquirir software nos aísla de las etapas inherentes al desarrollo (diseñar, construir, testear, documentar, etc) que involucran tiempo y esfuerzo, mientras nos brinda soluciones rápidas y estables a costo fijo.

* Diapositiva 17: Buy vs Build (continuación) (Tefa)

No siempre la adquisición de software construido satisfacerá exactamente las necesidades del cliente. No obstante, las personas son capaces de adaptar su forma de trabajo a sus herramientas o de encontrar soluciones con sus herramientas a pesar de sus limitaciones.

Es tendencia actual que se opte por herramientas ya desarrolladas, considerando el costo-beneficio de cada opción.

* Diapositiva 18: Refinamiento de Requerimientos y Prototipado Rápido (Vicente)

La buena definición de los requerimientos es esencial para el éxito del proyecto. El cliente no conoce exactamente lo que quiere y puede cambiar constantemente sus expectativas. Debe desarrollarse un método iterativo para especificar requerimientos en conjunto con el cliente. No puede construirse un producto de software sin retroalimentación. Los prototipos o simulaciones, aunque con alta abstracción y pocas restricciones, permiten al cliente visualizar lo especificado y comprobar si cubre sus necesidades.

* Diapositiva 19: Desarrollo Incremental: Crecer, No Construir (Vicente)

El software no debe desarrollarse como un conjunto de componentes, sino como funcionalidades incorporadas orgánicamente. Este desarrollo creciente se logra con iteratividad, trabajando desde lo general hasta el detalle. La complejidad no se aborda en su totalidad desde el comienzo. Permite realizar prototipos y facilita la trazabilidad. Ayuda a la moral de los equipos con cada logro.

* Diapositiva 20: Great designers (Vicente)

Los buenos diseños no provienen específicamente de buenos métodos de diseño y mejor formación académica, sino de grandes diseñadores. El desarrollo de software, al fin y al cabo, es una actividad creativa y requiere inspiración. Existen grandes diferencias probadas entre el promedio y los buenos diseñadores. No alcanzan el éxito sólo los proyectos elaborados por comités: muchos nacen de pocas grandes mentes.

* Diapositiva 21: Great designers (continuación) (Vicente)

Los buenos diseñadores deben ser tratados como los buenos administradores (managers), quienes tienen igual efecto en los resultados finales. Deben ser recompensados adecuadamente y criados para alcanzar su máximo potencial. Una vez identificados, debe favorecerse su educación y liderazgo y estimularlos con pares de su nivel, siguiendo cuidadosamente su desarrollo personal.

* Diapositiva 22: Fin

Extra:

Accidental complexity is caused by our tools and methods, such as having to write and test additional code in a language because we can't express our ideas directly, and things like that. New methods and techniques can reduce accidental complexity. I can write programs faster and better than I could twenty-five years ago, because I have better languages and tools.

Essential complexity comes from the fact that what we try to do with programming is inherently complicated, and that there is an irreducible complexity. "Essential", in this context, means "relating to the essence of the thing" rather than "very necessary".

LINKS

* <https://www.cs.colorado.edu/~kena/classes/5828/s12/lectures/02-nosilverbullet.pdf>
* <https://blog.acolyer.org/2016/09/06/no-silver-bullet-essence-and-accident-in-software-engineering/>
* <http://www.grin.com/en/e-book/321505/summary-of-no-silver-bullet-essence-and-accident-in-software-engineering>
* <https://www.goodreads.com/topic/show/464084-no-silver-bullet-essence-and-accident-in-software-engineering>
* <https://angelomartinezblog.wordpress.com/2016/07/17/no-silver-bullet/>
* http://paperswelove.org/raleigh-durham/docs/no\_silver\_bullet/no\_silver\_bullet.html#automatic-programming