# Una vista de los siglos 20 y 21 de la Ingeniería de Software

Barry Bohem

# La visión Hegeliana

- El filósofo Hegel propuso la teoría en que la comprensión humana sigue el siguiente camino:
  - tesis (esta es la razón por la que las cosas sucedan manera en que lo hacen)
  - antítesis (que muestra que la tesis falla en algunos aspectos importantes)
  - síntesis (es una superación que capta la lo mejor de ambos tesis y antítesis evitando al mismo tiempo sus defectos).

La idea es aplicar esta teoría para comprender la evolución de la Ingeniería del Software en el tiempo desde los 50's en adelante.

# 1950's Tesis: "La Ingeniería de Software es Como la Ingeniería de Hardware"

- En los 50's los ingenieros de software utilizaban las mismas prácticas y preceptos que los ingenieros de otras disciplinas.

- "Medir dos veces y cortar una vez"
- "Estamos pagando 600 dólares la hora para este equipo y $2 la hora de mano de obra, y quiero que actúes en consecuencia."
- "comprobaciones de escritorio, para ejecutar manualmente los programas antes de ejecutarlos en una computadora.

# 1950’s – Ejemplo caso de éxito

- - SAGE (Sistema Automatizado de Entorno Terrestre)
- - Proceso de desarrollo de tipo cascada.
- - el secreto del éxito fue el hecho de que todos los ingenieros hubieran sido entrenados para organizar los esfuerzos a lo largo de todas las disciplinas de la ingeniería.
- - Otro indicio de la orientación del Hardware fue la creación en los 50's de instituciones de profesionales del software ej IEEE y ACM cuyos nombres contienen alusiones al hardware y a la ingeniería.

# 1960's Antítesis: La Elaboración de Software

- Por los 60's comenzaron a verse diferencias significativas entre el software y el hardware

- - software era mucho más fácil modificar
- - no requiería costosas líneas de producción para hacer copias del producto.
- - una vez cargado, no requiere reconfigurar cada instancia de hardware

# 1960’s – Facilidad de Cambio

- Esta facilidad de cambio del software, llevó a adoptar una modalidad "code and fix" en contraposición a "medir dos veces, cortar una“. => Código Spaghetti
- Muchas aplicaciones de software comenzaron a demandar más personas para el soft que para el hard, incluso personas de otras disciplinas distintas de la ingeniería.

# 1960’s – Software vs. Hardware

- El software comenzó a "no encajar" con los aspectos del hardware.
  - – - Es invisible, no pesa nada, pero cuesta mucho.
  - – - Es difícil saber si el proyecto va bien o se está atrasando.
  - – - Si agregamos más gente para tratar de cumplir con el Schedule se retrasa aún más (Brooks).
  - – - Se necesita mucha más especificación que para el hardware.

# 1960’s – No todo fue code and fix

- Algunos contraejemplos del code and fix en los 60's:
- - OS-360 de IBM (caro, tarde, y difícil de usar, pero confiable y completo)
- - NASA Mercurio, Géminis y Apolo (nave espacial tripulada y el software de control en tierra)

# 1960’s tendencias

- Otras tendencias en los 60s:

- - Mejores infraestructura en OS de mainframe, y maduración de lenguajes de alto nivel como Fortran y COBOL.
- - El establecimiento de ciencias de la computación y la informática departamentos de universidades, con mayor énfasis en la software.
- - El inicio del desarrollo de software con fines de lucro y las compañías de productos.

# 1970's Síntesis y Antítesis: La formalidad y los procesos en cascada

- - Proceso de codificación más cuidadoso, precedido de un diseño, proviniente de requerimientos (en respuesta al code and fix de los 60s).

# 1970’s - Tendencias

- - Programación estructurada (A raíz de la famosa carta Carta de Dijkstra acerca de la peligrosidad de la instrucción GOTO)
- - Con dos ramas: Métodos formales (comprobación matemática de correctitud) vs. Programación "Top-Down".
- - Principios de modularidad (principios de acoplamiento y cohesión)
- - Information Hiding (Parnas)
- - TADs
- - Diseño estructurado

- Aproximaciones cuantitativas.
- Se desarrollaron algunas métricas para ayudar a identificar módulos propensos a defectos.

# 1970’s – Modelo en cascada

- El modelo waterfall se interpretó como un proceso sequencial, en el cual cada fase no inicia hasta que se termina la anterior (tal como lo indicaban los procesos militares).
- Algunos gerentes impacientes aceleraban la fase de codificación teniendo un mínimo de requerimientos y de diseño, porque "tendrían mucho tiempo que invertir en debugging".
- En contraposición Royce propuso un modelo en cascada sintetizando los procesos dirigidos por requerimientos con el proceso de manufactura de los 60s
- En donde se agregan iteraciones en fases sucesivas, prototipando antes de continuar con el desarrollo general.
- Luego tras una verificación y validación de los artefactos, se proceder a la proxima fase.

- Los procesos formales y los modelos en cascada, requerían mucha documentación y eran caros y lentos de usar.
- Por lo cual presentaban problemas de escalabilidad y de usabilidad por los programadores menos expertos.

# 1980’s: Síntesis Productividad y Escalabilidad

- El surgimiento de los métodos cuantitativos de los 70s ayudaron a identificar mejor dónde invertir más esfuerzos.
- Ejemplos:
- - muchas tareas de testing se podrían haber realizado mucho más eficientemente en etapas tempranas.
- - se podrían reducir costos utilizando herramientas de soft, generando procesos, evitando o automatizando tareas, reusando, etc.

# 1980’s - Tendencias

- Estándares
- Hubieron varias instituciones que analizaron los problemas de la IS, y generaron algunos estándares Ej ISO9001.
- Pero sin abandonar definitivamente el modelo en cascada.
- En algunos casos el adoptar estos estándares tuvieron buenos resultados reduciendo el doble trabajo.

# 1980’s - Herramientas de Software:

- - analizadores de coverage del codigo
- - generadores de casos de test
- - herramientas de test unitario
- - herramientas de trazabilidad
- - otras

# 1980’s – procesos de software

- Procesos de Software:
- - Procesos de programación
- - Procesos de requerimientos
- - Procesos de arquitecturas
- - Procesos de administración de cambios
- - Otros

# 1980's Evitar/ automatizar tareas:

- Mejoras potenciales de productividad
- - Lenguajes de nivel más alto
- - Programacón orientada a objetos
- - Programación Visual (ej. manejador de ventanas común)

- Tareas evitables/automatizables vs. las escenciales que requieren la intervención humana. (No Silver Bullet)
- Las posibles técnicas que atacan las tareas escenciales incluyen
- buenos diseñadores
- prototipado rapido,
- desarrollo que evoluciona (growing vs. building)
- reuso (ej. POO)

# 1990’s Antítesis: Concurrencia vs. Procesos Secuenciales

- En los 90 se continuó fuertemente con el desarrollo orientado a objetos.
- - Desarrollo del UML
- - Patrones de diseño
- - Lenguajes de descripción de arquitecturas
- - Emerge de la WWW.

# 1990’s - Tendencias

- - Time to market
- Con la expansión de la WWW, y la necesidad de competir en el mercado comenzó a sentirse significativamente la
- lentitud de los modelos en cascada.
- - Diseños interactivos: Tendencia Bottom-up en vez de top down.

- Concurrencia:
- - Al realizar los procesos concurrentemente se necesitaron técnicas para el manejo concurrente de las distintas tareas de ingeniería de modo que se puedan sincronizar y establilizar.
- Ejemplos de organizaciones que adoptaron estas técnicas AT&T, Rational RUP, Microsoft.
- - Otro ejemplo fue el desarrollo Open Source que tuvo importante auge en los 90s (Ej. Torvald's Linux 1991)

# 2000's Antítesis y Síntesis Parcial: Agilidad y Valor

- Durante los 2000's hay un rápido cambio en
- las tecnologías de información (Herramientas en la nube, Redes Sociales),
- En las organizaciones (startups, adquisiciones)

- El esquema waterfall es obviamente obsoleto

# 2000’s – Métodos Ágiles

- Métodos Ágiles

- Desarrollo Adaptivo
- Crystal
- Desarrollo de Sistemas Dinámicos
- Extreme Programming XP
- Desarrollo orientado a Feature
- Scrum
-
- Manifiesto Ágil (cuatro reglas)
    - Interacciones individuales sobre procesos y herramientas
    - Software funcionando sobre documentación
    - Colaboración del cliente sobre negociación del contrato
    - Respuesta al cambio sobre seguir el plan.

# 2000’s – Ágiles (no sirven siempre)

- Los métodos ágiles mostraron buenos resultados en la práctica cuando los proyectos no son muy grandes.
- Los intentos de escalar métodos ágiles que consistieron en hacer equipos de equipos demostraron en la práctica que
- -cada equipo desarrollaba su propia arquitectura, haciendo difícil la integración.
- -cada equipo se concentraba en lo más fácil primero, careciendo de seguridad y escalabilidad global

# 2000’s - Tendencias

- Tendencias 2000s:
- - Sacar productos al mercado lo más rápido posible (métodos ágiles) y al menor costo.
- - Incremento en el énfasis de la usabilidad
- - Software crítico, de confianza.
- - Uso creciente de componentes de 3ros "out of the shelf" COTS
- Problemas:
- - no poder acceder al código fuente
- - interoperabilidad
- - costos de integración
- - Desarrollo Orientado a Modelos (Grandes organizaciones definen interfaces y protocolos, los COTS se ven forzados a integrar esas interfaces.)

# 2010 y más: Antítesis y Síntesis parcial: Globalización y Sistemas de Sistemas

- - Conectividad Global
- - Sistemas masivos de sistemas
- - Incremento en la autonomía del software y la combinación con la biología

- …

# Conclusiones

- 1950's
- + No olvidar de las ciencias. Deben incluírse otras disciplinas.
- + Pensar antes de actuar.
- - No usar procesos secuencialmente rigurosos.
- 
- 1960's
- + Pensar fuera de los límites. La ingeniería repetitiva nunca hubiera creado Arpanet o el Mouse y las ventanas.
- + Respetar las diferencias del software. Como es invisible hay que encontrar una buena manera para hacerlo visible.
- - Evitar la programación con ganchos.

# Conclusiones

- 1970's
- + Eliminar errores temprano. (Y si es posible prevenirlos en el futuro atacando la causa)
- + Determinar el proposito del sistema.
- - Evitar desarrollo Top-down.
- 
- 1980’s
- + Hay muchas maneras de incrementar la productividad (staffing, herramientas, procesos, reuso)
- + Lo que es bueno para los productos es bueno para los procesos
- - No hay soluciones mágicas o silver bullets

# Conclusiones

- 1990’s
- + El tiempo es dinero. Cuanto antes el soft esté listo, antes empieza a redituar.
- + Hacer el soft útil para la gente.
- - Ser rápido, pero no apresurarse de modo de tener los requerimientos subespecificados.

- 2000's
- + Si el cambio es rápido la adaptability triunfa.
- + Considerar y satisficer los requerimientos de todos los the stakeholders.
- - Evitar el abuso de los slogans. YAGNI (you aren’t going to need it) no es siempre así.

# Conclusiones

- 2010’s
- + Mantenerse dentro del alcance. Algunos sistemas de sistemas son simplemente muy grandes y complejos.
- + Tener una estrategia de salida. Si las cosas salen mal hay una alternativa.
- - No creer todo lo que se lee.