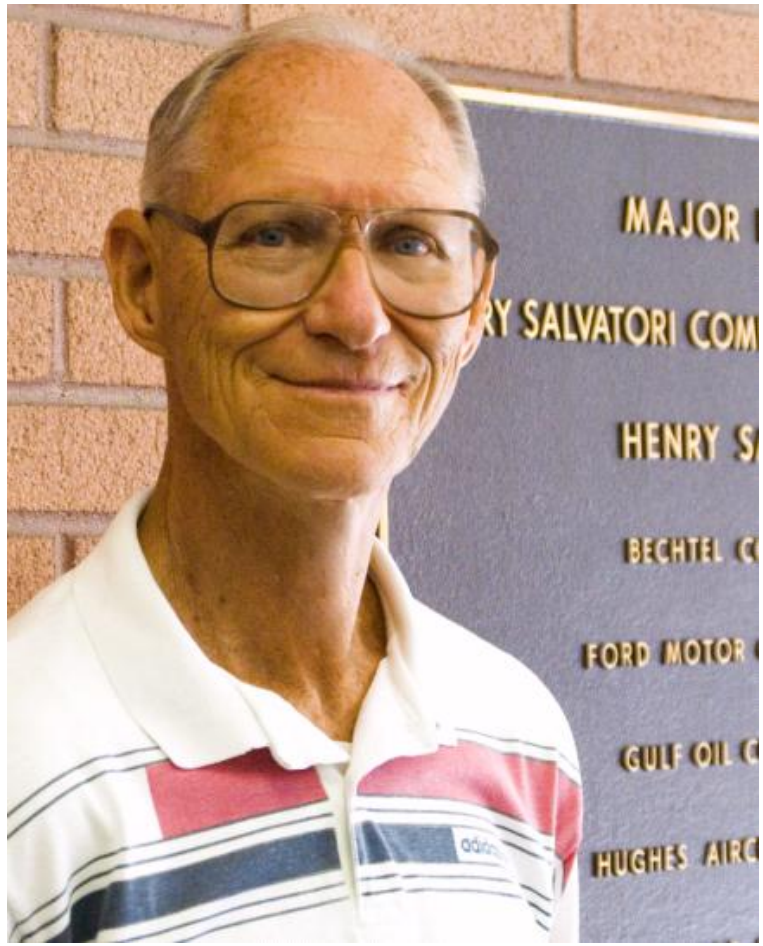
A view of 20th and 21st Century Software Engineering

Integrantes:
Alex Valencia, Francisco Tarulla, Germán Del Tuffo, Guillermo Steren, Leandro Oniszczuk.

Acerca del Autor

Barry Boehm
Lugar de publicacion del paper:ICSE 2006
Fecha:Mayo de 2006

Recibido con una maestría en Matemáticas de Harvard y un doctorado en la UCLA en los 60's.
Sus principales contribuciones fueron el modelo constructivo de coste (COCOMO) y el modelo espiral win-win entre otros.



¿ Por qué este paper es importante ?

"Those who cannot remember the past are condemned to repeat it"

George Santayana - escritor

"In an era of rapid change, those who repeat the past are condemned to a bleak future"

Barry Boehm

Algunas definiciones importantes

Tesis: Conclusión, proposición que se mantiene con razonamientos

Antitesis: Oposición o contrariedad de dos juicios o afirmaciones

Sintesis: Composición de un todo por la reunión de sus partes:

Nos propone....

Identificar las mayores experiencias del software que vale la pena repetir.

Nos propone....

Identificar las mayores fuentes de cambios que afectarán a las prácticas de la ingeniería del software.

Nos propone....

Reconcer principios atemporales y prácticas viejas.

Definicion de Boehm de la Ingenieria de Software

"The application of sciences and mathematics by which the properties of software are made useful to people"

1950's Tesis

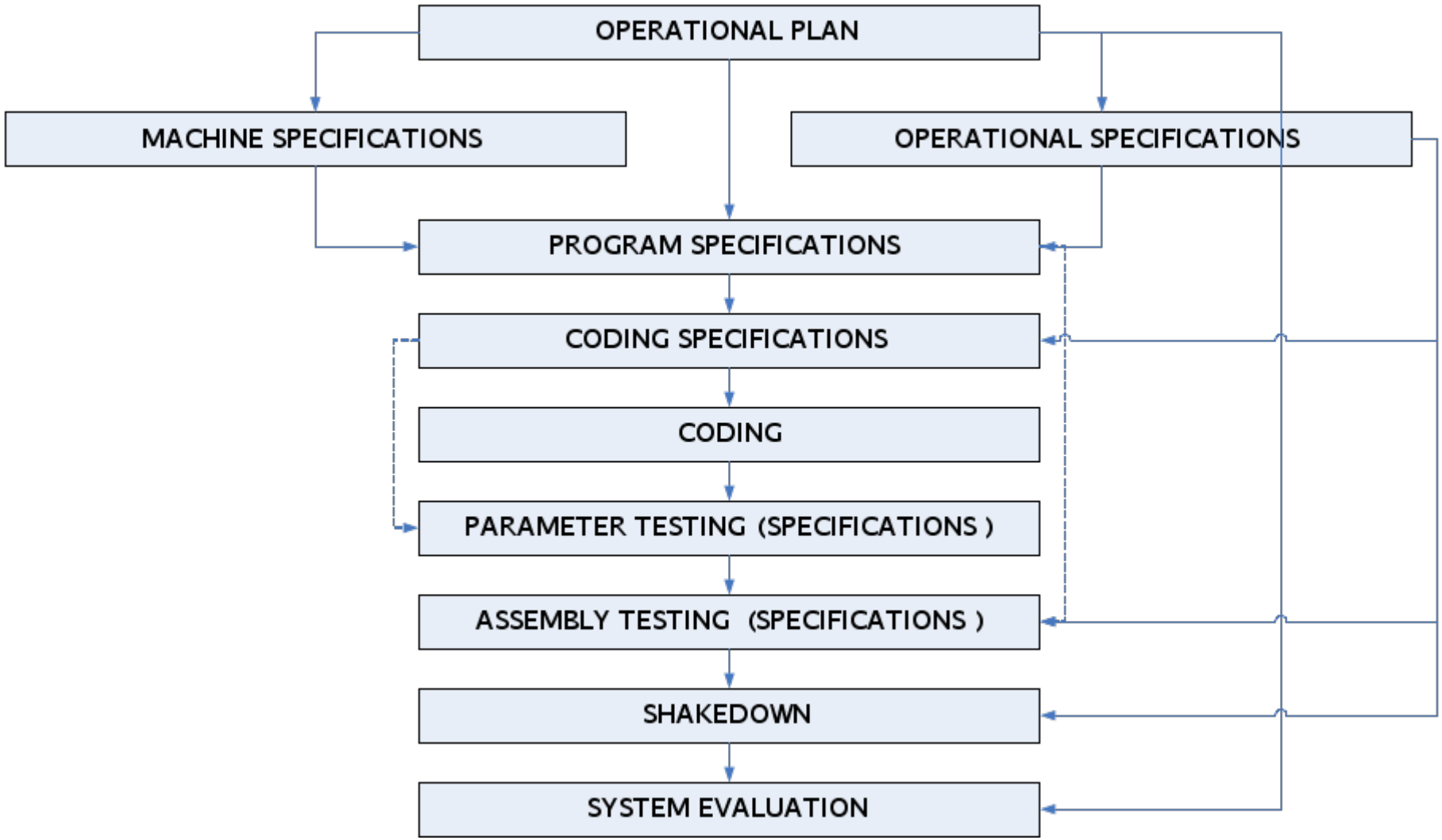
La Ingeniería de Software es como la Ingeniería de Hardware

1950's

- Programadores eran ingenieros o matemáticos.
- Apps de Software: aviones, defensa, puentes, circuitos.
- Regían preceptos de hardware como "Medir 2 veces cortar 1 vez" (hora de computadora costaba 300 veces más que hora de programador). => Mucha revisión y ejecución manual previa a ejecución en computadora.
- Se utilizaba los modelos (orientados a hardware) secuencial Waterfall o Modelo V.
- Se atribuía el éxito a que todos fueran ingenieros.

+: Aprovechar las ciencias, Usar método científico. No hacer commits prematuros (sin analizar bien).
-: Uso de Waterfall, el mundo cambia demasiado rápido y de forma impredecible. Además es lento.

1950's


OPERATIONAL PLAN
MACHINE SPECIFICATIONS
OPERATIONAL SPECIFICATIONS
PROGRAM SPECIFICATIONS
CODING SPECIFICATIONS
CODING
PARAMETER TESTING (SPECIFICATIONS)
ASSEMBLY TESTING (SPECIFICATIONS)
SHAKEDOWN
SYSTEM EVALUATION

1960's Antítesis

Artesanía de Software

1960's

- Software != Hardware

- Aplicaciones pasan a orientarse más a gente que a hardware.
- Software es más intangible y complejo que hardware. Muchas medidas y estrategias pierden validez.

- Informática en Universidades. Cultura Hacker, Open source.

- Comienza a desarrollarse software comercialmente.

1960's

- El software se modifica y distribuye más fácil y económico que el hardware.
- Mucha demanda de software + lenguajes de alto nivel => Reclutó programadores no ingenieros o matemáticos. Arte, sociales...
- Surge estrategia "CODE & FIX" and fix... and fix... and fix... => código spaghetti altamente emparchado. Programadores Cowboy.
- Proyectos largos tendían a fracasar.
- Primeras conferencias de ingeniería de software.

+: Usar inventiva, respetar diferencias de software con hardware. Visibilidad.

-: Evitar programación cowboy (arreglos de último momento).

1970's Síntesis y Antítesis

Formalidad y el Proceso Waterfall

1970's

- No más Code and Fix
- Toman fuerza el análisis y el diseño, se establece el principio de bajo acoplamiento y alta cohesión
- Royce introduce el waterfall con prototipos, aunque es interpretado como secuencial
- Se requieren la creación de métricas para ciertos aspectos del proceso de desarrollo
- Se descubre que el factor humano es importante para el desarrollo
- A fines de esta década el waterfall secuencial muestra sus falencia

1980's Síntesis

Productividad y Escalabilidad

1980's

Iniciativas para solucionar problemas de los 70's

- DoD-STD-2167 y MIL-STD-1521B (DoD - 1985)
- SW-CMM (SEI formado en 1984)
- ISO-9001 (Europa)

Herramientas de Software

- Test
- Gestión de la configuración (CM). Proyecto IMPACT
- Integración de herramientas en entornos de soporte (IPSE's, CASE, Software Factories)
- Entorno RAISE, Toaster Model
- Investigación sobre entornos avanzados de desarrollo

1980's

Procesos de Software

- Entornos de software apoyado en procesos
- *"Software Processes are Software too"* - Osterweils
- Prácticas que son buenas para desarrollar procesos
- Reducir repetición de trabajo + evitar el trabajo = Mayor producitividad
- DoD STARS: Especificaciones formales y generación de código (1970 Automatic Programming)
- No Silver Bullet - Brooks
-

Reutilización de Software

- S.O. más potentes, GUI Builders, etc.
- Metáfora de Desktop (Apple Lisa 1983, Macintosh 1984, MS Windows 3.1 198x)
- Mejor arquitectura e ingeniería de dominio = reuso de componentes
- Lenguajes de 4ta generación (4GL) FOCUS y NOMAD
- Programación Orientados a Objetos (desde Simula-67)

1990's Antítesis

Procesos concurrentes vs. secuenciales

1990's

Continúa el momento "fuerte" de los métodos OO

- Patrones de diseño
- Arquitecturas de Software y lenguajes de descripción
- Desarrollo de UML

Reducir time-to-market

- Software como un factor de competencia
- Productos orientados a la interacción con los usuarios

Control de concurrencia

- Modelo Espiral dirigido por riesgos (falta de guía)
- Reemplazo del proceso waterfall por procesos concurrentes, incrementales y desarrollo evolutivo

Open Source (ej. de ingeniería concurrente)

- Torvalds: Linux (1991)
- Raymond: The Cathedral and the Bazzar (1997)

Usabilidad y Interacción Humano-Computadora (HCI)

- Regla de Oro vs. Regla de Platino

2000's Antítesis y Síntesis parcial

Agilidad y Valor

2000's

Continúa la tendencia hacia Rapid application development
Aceleración de la velocidad de cambio en:

- tecnología
- organizaciones
- contramedidas de competencia
- el entorno

Algunas técnicas entran en crisis:

- planificación pesada
- especificaciones
- otras documentaciones

Métodos Ágiles

- Agile Manifesto (2001)
- El más adoptado: eXtreme Programming
- Proyectos chicos: métodos ágiles
- Proyectos grandes: métodos ágiles + prácticas de plan-driven

2000's

Value-Based Software Engineering

- Mejoras en la usabilidad mediante pequeños incrementos y contenido priorizado por el usuario
- "Computers are working about as fast as we need. The bottleneck is making it all usable" W. Brian Arthur
- Tecnología adaptada a las personas y no viceversa
- Las prioridades del usuario cambian en el tiempo

Criticalidad del software y fiabilidad

- Time-to-market vs. Confiabilidad
- Tecnologías para asegurar confiabilidad (model-checking, model-based testing)

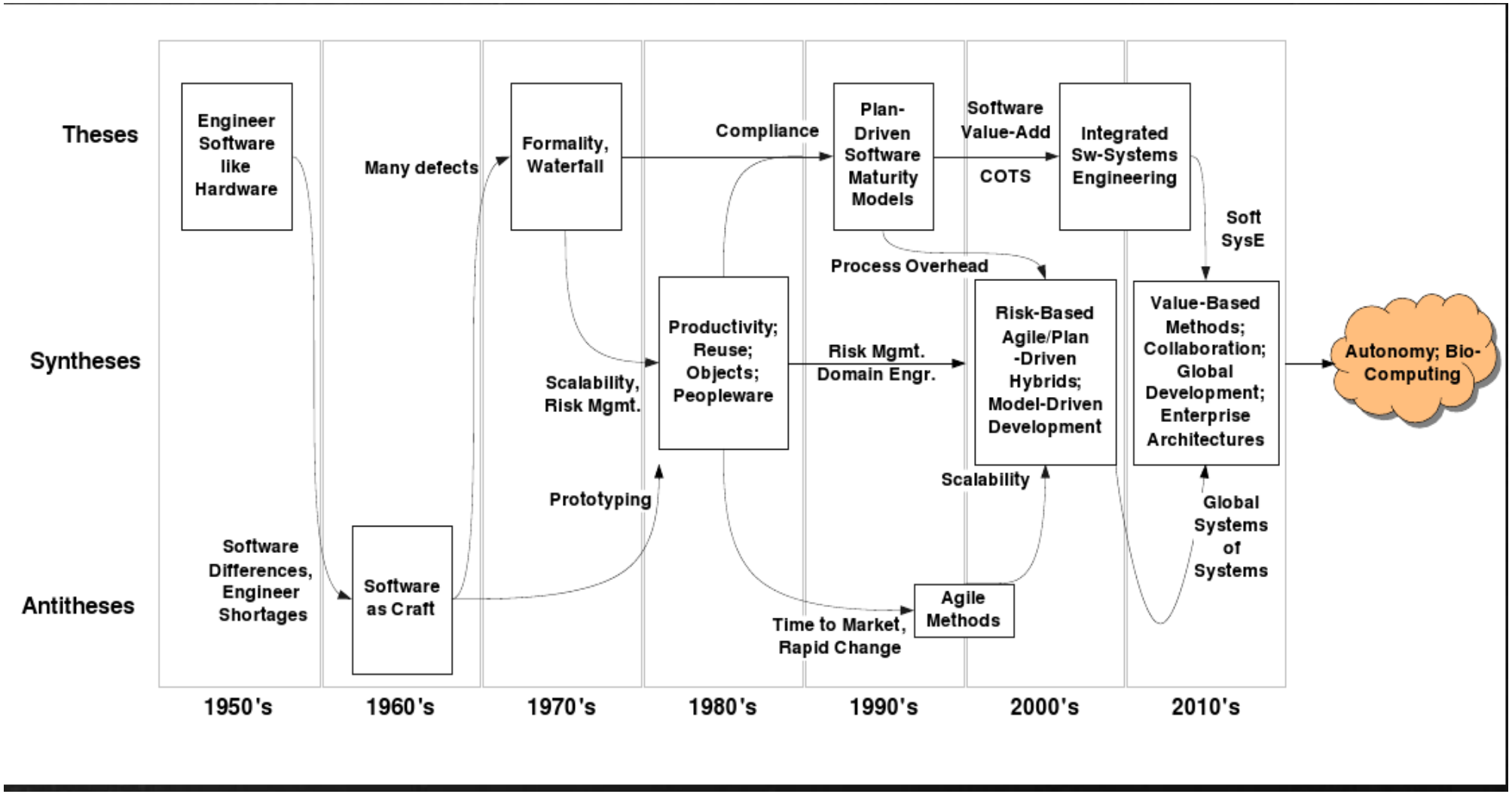
COTS, OpenSource y Legacy software

- Programación = Adaptación + Integración

Desarrollo Model-Driven

Interacción de software e ingeniería de software

Resumen del avance histórico



Una proyección al futuro

Alguna de las tendencias ya se ven hoy en día

- Rapidez ante cambios
  - La facilidad adaptación de un producto muchas veces marca la diferencia entre exito y fracaso
- Mejora en la usabilidad de las aplicaciones.
  - Esta es una tendencia que se hace fuerte en los 90 y su importancia sigue creciendo hoy en día
- Incremento en el uso de COTS
  - Para ahorrar tiempo y dinero los productos cada vez usarán más COTS

- Conectividad mundial
  - Es posible obtener desarrollo de software desde cualquier lugar del mundo para reducir los costos.
- Uso masivo de SOS
  - la cooperacion intensiva entre aplicaciones para resolver nuevos objetivos, aumenta la performance y el alcance.

Más alla del 2020

Nuevos productos: materiales inteligentes

- Biologia & computación
  - Computacion basada en biologia: utiliza elementos biologicos o moleculares para resolver problemas
  - Incorporar al cuerpo/mente humana estructuras roboticas para potencias logros.
- Autonomia de software
  - Cooperación entre agentes para obtener la mejor forma factible de actuar.
  - Autoconfiguración para poder responder ante un cambio inesperado.

Conclusiones:

Este paper muestra cómo la misma ingenieria del Softwate se desarrolla mediante un modelo iterativo. Es decir que asi como el desarrollo de un producto se basa fuertemente en iteraciones incrementales, intentando buscar la menara mas barata y rapida de hacer un producto, la Ingenieria en si misma busca su propia "mejor manera" de ser implementada. En el paper cada iteracion es una decada y los requerimientos cambiantes surgen de cambios en la tecnologia y el mundo y del tipo de problemas que la Ingenieria de Software intenta resolver.
Podríamos ver que la historia de la Ingenieria del Software en si misma se parece a la de un desarrollo particular pero visto de manera macro...

Conclusiones:

Además el paper hace especiál énfasis en distinguir de las estrategias surgidas en respuesta a necesidades del momento, cuáles son útiles más alla del momento en el que surgieron y cuales presentarían riesgos si se intentan repetir. Es interesante ver el efecto marcado por las tendencias y las modas en el software y sobretodo la relación muchas veces dada de tesis y antitesis entre las prácticas desarrolladas en diferentes decadas.

Idea: Intentar repetir los éxitos del pasado y evitar repetir los errores.

Conclusiones (para pensar):

Se ve de una manera muy remarcada el efecto de la economía en los rumbos que la ingeniería del software fue recorriendo, dejando en claro que existe una tendencia a ser una disciplina dominada por intereses económicos más que intereses académicos. Cabe plantear el debate de cuan bueno es esto para la "gente del software" en todos sus perfiles (estudiantes, profesionales, academicos, etc...)

¿Esta bueno que el factor económico juegue el papel más decisivo en la industria del Software?