CD - 1768 - 17

FACULTAD DE CIENCIAS EXACTAS Y NATURALES
FOLIO 6

Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales

Planilla a completar para presentación de Cursos de Posgrado

1.- DEPARTAMENTO de COMPUTACIÓN

2.- NOMBRE DEL CURSO: Historia y Filosofía de los Lenguajes de Programación

3.- DOCENTES:

RESPONSABLE/S: Hernán A. Wilkinson .....
COLABORADORES: .....
AUXILIARES: Agustín Martinez.....

4.- CARRERA de DOCTORADO

5.- AÑO: 2017 CUATRIMESTRE/S: 2

6.- PUNTAJE PROPUESTO PARA CARRERA DE DOCTORADO:
.....3.....

7.- DURACIÓN (anual, cuatrimestral, bimestral u otra): Cuatrimestral

8.- CARGA HORARIA SEMANAL:

Teóricas: 2
Problemas:
Laboratorio: .....
Seminarios: .....
Teórico – Práctico: 2.....
Salida a Campo: .....

3 PTOS

9.- CARGA HORARIA TOTAL: 64

10.- FORMA DE EVALUACIÓN: Un Seminario con tema a elección relacionado con los temas vistos. Un trabajo práctico con presentación oral. Un Proyecto Final integrador

11.- PROGRAMA ANALÍTICO:

Lamentablemente la historia de las Ciencias de Computación es desconocida por sus practicantes y estudiantes. Pocos conocen quiénes son y qué hicieron personas como John McCarthy, Peter Naur, Douglas Engelbarg, Alan Kay, Alan Perlis, David Ungar, J. Sussman, K. Nygard, J. Dahl y Tony Hoare entre otros. Menos aún conocen sus trabajos y la influencia que tuvieron tanto en los lenguajes de programación como en la tecnología que usamos.

Conocer la historia de una profesión ayuda a entender la actualidad y profundizar hacia dónde se dirige.

Conocer lenguajes como Lisp, Scheme, Simula-67, Algol 60, Miranda, Forth y otros ayuda a realizar un análisis crítico de los lenguajes "mainstream" de la actualidad. A la luz de los lenguajes fundacionales, aquellos utilizados actualmente en la industria no presentan características innovadoras. Mientras que lenguajes fundacionales se concibieron en base a una mirada acerca del modo en que el humano conoce el mundo y expresa ese conocimiento, lenguajes recientes y de moda son concebidos como una reversión de aquellos del pasado combinando atributos distintivos ('features') de uno y otro lenguaje previo.

La actualidad de la profesión se caracteriza por una constante aparición de 'nuevas' tecnologías y herramientas sin ningún tipo de valor agregado real. Para distinguir la relevancia de estas nuevas tecnologías, la perspectiva histórica se torna esencial.

Conocer cómo se gestaron las distintas tecnologías que se usan en la actualidad como los avances realizados por D. Engelbart, A. Kay, I. Sutherland, A. Golberg entre otros, ayuda a comprender por qué usamos la tecnología que usamos, cuáles son sus limitaciones y hacia donde estamos yendo.

Durante la cursada de esta materia se verán estos y otros temas relacionados, con el objetivo de generar en el alumno un espíritu crítico de las herramientas y tecnologías que usamos basado en buenos fundamentos técnicos e históricos.

Parafraseando a Alan Kay, "nuestra cultura es una cultura pop... desconocemos nuestra historia y por eso nos pasamos reinventando la rueda pinchada". Nuestro objetivo es proveer herramientas de pensamiento y una perspectiva histórica que nos permita evitar inventar la rueda pinchada.

El abordaje filosófico toma como punto de partida el análisis histórico de los lenguajes fundacionales, focalizando la mirada en las motivaciones que les dieron origen, aquellas explicitadas por sus autores y aquellas más generales de los inicios de la computación (Bush 1945; Licklider 1960; Engelbart 1962). Así se abre camino a la discusión acerca de qué son los lenguajes de programación, qué es el software y de qué se trata la actividad de programar. Diferentes puntos de vista filosóficos son presentados (desde Moor 1978 y Suber 1988 hasta Rapaport 2005; Turner 2013; Irmak 2013; Wang 2016) poniendo en evidencia que caracterizar los lenguajes, al software y a la actividad de programar es un tema abierto y con puntos de vista dispares, a pasar de que en la profesión el enfoque del desarrollo de software como una actividad ingenieril es la dominante. El desarrollo de software como ingeniería ha sido cuestionado desde sus inicios hasta la actualidad (Hoare 1978; Davis 2011) y en oposición han surgido respuestas desde el campo profesional, por ejemplo, Beck et al. (2001) y una década antes Reeves (1992), que declaraba que "programar es una actividad de diseñar" y "el problema que abruma con el desarrollo de software es que todo es parte del proceso de diseño". Punto de vista recuperado por la filosofía recientemente (Martinez 2016) y que será abordado en la materia.

La modalidad del curso incluye:

1. Material multimedia de entrevistas/charlas/conferencias realizadas por las personas a estudiar en cuestión
2. Lectura de papers y material histórico proveniente del congreso de HOPL (History of Programming Languages), A History of Personal Workstation, la organización de Computer History Museum y otros
3. Ejercicios prácticos relacionados a los lenguajes de programación que se verán.

Los temas y unidades que se verán son:

- Unidad 1: Turing vs. Church
  El objetivo de esta unidad es analizar las dos formas de definir la "computabilidad" y sus implicancias en la forma de desarrollar e implementar lenguajes de programación
- Unidad 2: Lisp – John McCarthy
  En esta unidad se verá una implementación casi directa del Lambda Calculus, Lisp. Se analizará las diferencias con lenguajes imperativos y de estilo Turing, que son los que dominaron el principio de nuestra profesión
- Unidad 3: Algol 60 – P. Naur, J. Bakus, T. Hoare
  El objetivo de esta unidad es profundizar en las diferencias con lenguajes tipo Church como Lisp, visto en la unidad anterior, analizando principalmente la restricción de realizar "Funciones de Alto Orden"

- Unidad 4: Augmenting Human Intelect – D. Engelbard
  En esta unidad se empezará a analizar la visión de la computadora como "ayudante" del ser humano y no como "reemplazo" del ser humano, tal como lo proponía la corriente de "inteligencia artificial"
- Unidad 5: Sketchpad – I. Sutherland
  El objetivo de esta unidad es ver cómo la interacción con la computadora a partir de una interfaz gráfica empieza a cambiar completamente el paradigma computacional y empieza a generar nuevos retos que los lenguajes de programación tradicionales no estaban preparados para resolver.
- Unidad 6: Simula 67 – K. Nygard, J. Dahl
  En esta unidad se empieza ver el nacimiento de la necesidad de organizar el conocimiento adquirido mientras se desarrolla software. Se analizará el modelo Clásico-Aristotélico propuesto por Simula-67
- Unidad 7: Smalltalk – A. Kay, D. Ingalls, A. Goldberg
  El objetivo de esta unidad es ver la confluencia de las unidades anteriores en el sistema Smalltalk: lenguaje tipo Church, interacción dinámica con el usuario, computadora como ayuda del ser humano y organización del conocimiento
- Unidad 8: Xerox Parc
  En la década del 70, el laboratorio de investigación de Xerox ubicado en Palo Alto fue le generador de la mayoría de la tecnología utilizada en la actualidad, desde impresoras láser pasando por Ethernet, interfaces gráficas y programación orientada a objetos. Estudiar lo realizado en este laboratorio permitirá darle una perspectiva menos comercial y más científica a estos avances tecnológicos.
- Unidad 9: Scheme – Sussman, Guy Stelle
  El objetivo de esta unidad es ver cómo la unión de lenguajes tipo Church con lenguajes tipo Turing generan nuevas maneras de representación, principalmente el Closure.
- Unidad 10: C++ - B. Stroustrup
  En esta unidad se ve otra manera de interpretar la programación orientada a objetos representada en este caso en el lenguaje C++
- Unidad 11: Self – D. Ungar, R. Smith
  El objetivo de esta unidad es ver otras propuestas al problema de organización de conocimiento dada por los lenguajes de programación. En este caso se analizará la opción basada en prototipos arraigada en la idea propuesta por Wittgenstein sobre "entes prototípicos y familiares"
- Unidad 12: Miranda
  En esta unidad se verá cómo se afianza el paradigma funcional a partir del lenguaje Miranda, que luego es utilizado en otros lenguajes como Haskell.
- Unidad 13: Actualidad: Ruby, JavaScript, Java, .Net, Python, Clojure, Scala
  El objetivo de esta unidad es analizar los lenguajes de programación utilizados en la actualidad en base a los principios de los lenguajes fundacionales.
- Unidad 14: "Objetos vs. Funcional" u "Objetos y Funcional"
  Por último, en esta unidad veremos como la dicotomía objetos vs. Funcional no es más que una separación absurda y que en realidad ambas visiones se complementan.

12.- BIBLIOGRAFÍA:

Jean E. Sammet "History of Programming Languages I". ACM, 1978

Jean E. Sammet, Jan Lee "History of Programming Languages II". ACM, 1993

Barbara Ryder, Brent Hailpern "History of Programming Languages III". ACM, 2007

Harold Abelson, Gerald Jay Sussman "Structure and Interpretation of Computer Programs - 2nd Edition" (MIT Electrical Engineering and Computer Science) Hardcover – July 25, 1996

Michael I. Levin (Author), John McCarthy (Contributor), "LISP 1.5 Programmer's Manual" Paperback – August 15, 1962

Lamprecht Günther (Author), "Introduction to SIMULA 67" Paperback – July 1, 2013

Adele Goldberg (Author), David Robson (Author), Michael A. Harrison (Editor), "Smalltalk-80: The Language and its Implementation", Hardcover – May, 1983

MARCEL VAN DER VEER, "LEARNING ALGOL 68 GENIE", 2013

Michael A. Hiltzik, "Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age", HarperBusiness, 2000

Peter Seibel, "Coders at Work: Reflections on the Craft of Programming", 2009

Susan Lammers. "Programmers at Work: Interviews With 19 Programmers Who Shaped the Computer Industry", Tempus 1989
Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Kern, J. (2001). The agile manifesto. www.agilemanifesto.org

Bush, Vannevar (1945). As We May Think en Life Magazine. 9/10/1945. 112-124. Luego publicado en Atlantic Monthly en September 1946.

Davis, M. (2011). Will software engineering ever be engineering?. Communications of the ACM, 54(11), 32-34.

Engelbart, D. C. (1962). Augmenting Human Intellect: A Conceptual Framework. Summary Report AFOSR-3223 under Contract AF 49 (638)-1024, SRI Project 3578 for Air Force Office of Scientific Research. Stanford Research Institute. Retrieved March, 1, 2007.

Hoare, C. A. R. (1978). The engineering of software: a startling contradiction. In Programming Methodology (pp. 37-41). Springer New York.

Irmak, N. (2013). Software is an abstract artifact. Grazer Philosophische Studien, 86(1), 55-72.
Licklider J. C. R. (1960) Man-Computer Symbiosis. IRE TRANSACTIONS ON HUMiAN FACTORS IN ELECTRONICM. Marzo 1960. 4-11.

Martínez, A. (2016) "¿Software es diseño?" En VII Coloquio de Filosofía de la Técnica, sede ADIUC, Ciudad Universitaria, Córdoba Capital, Argentina, Octubre de 2016.

Moor, J. H. (1978). Three myths of computer science. The British Journal for the Philosophy of Science, 29(3), 213-222.

Rapaport, W. J. (2005). Philosophy of Computer Science: A Course Outline. En Buffalo.edu http://www.cse.buffalo.edu/~rapaport/Papers/philcs.pdf

Reeves, J. W. (1992). What is software design. C++ Journal, 2(2), 14-12.

Suber, P. (1988). What is software?. The Journal of Speculative Philosophy, 89-119.

Turner, R. (2013). The philosophy of computer science. Stanford Encyclopedia of Philosophy

Wang, X., Guarino, N., Guizzardi, G., & Mylopoulos, J. (2016) Software as an Information Artifact. En ResearchGate.net

Walter Isaacson, The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital, Simon & Schuster; Reprint edition (October 6, 2015)

Leo Brodie, Thinking Forth Paperback, December 27, 2004

A. Ríos