Greg Wilson

Enseñar tecnología en comunidad

Dedication

Para mi madre, Doris Wilson,
que enseñó a cientos de niñas y niños a leer y creer en sí mismas/os.

Y para mi hermano Jeff, que no vivió para verlo terminado.
"Recuerda, todavía tienes muchos momentos buenos frente a ti."

La traducción de este libro al español está dedicada a la memoria de Rebeca Cherep de Guber[1].

Todas las regalías de la venta de este libro se donan a MetaDocencia[2], una organización basada en trabajo voluntario que enseña a docentes de habla hispana de todo el mundo a enseñar de forma efectiva usando prácticas basadas en evidencia.

Índice general

Las reglas

1. Sé amable: todo lo demás son detalles.

2. Recuerda que tú no eres tus estudiantes...

3. ...que la mayoría de la gente prefiere fracasar antes que cambiar...

4. ...y que el 90 % de la magia consiste en saber una cosa más (que tu público).

5. Nunca enseñes sólo/a.

6. Nunca dudes en sacrificar la verdad por la claridad.

7. Haz de cada error una lección.

8. Recuerda que ninguna clase sobrevive al primer contacto con estudiantes...

9. ...que cada clase es demasiado corta para quien enseña y demasiado larga para quien la recibe...

10. ...y que nadie tendrá más entusiasmo que tú por tu clase.

Sobre la traducción

Este es el sitio web de la versión en español, **aún en proceso de traducción**, de *Teaching Tech Together* de Greg Wilson. La traducción de *Enseñar Tecnología en Comunidad* es un proyecto colaborativo de la comunidad de R-Ladies[3] y de MetaDocencia[4] en Latinoamérica, que tiene por objetivo traducir al español material actualizado y de calidad para hacerlo accesible a hispanohablantes. Iniciamos la traducción en Marzo del año 2020 y esta actualización corresponde a Diciembre de 2020.

Quienes trabajamos en este proyecto somos (en orden alfabético): Laura Acion[5], Mónica Alonso[6], Zulemma Bazurto[7], Alejandra Bellini[8], Yanina Bellini Saibene[9], Juliana Benitez Saldivar[10], Lupe Canaviri Maydana[11], Silvia Canelón[12], Ruth Chirinos[13], Paola Corrales[14], María Dermit[15], Ana Laura Diedrich[16], Patricia Loto[17], Priscilla Minotti[18], Natalia Morandeira[19], Lucía Rodríguez Planes[20], Paloma Rojas[21], Gabriela Sandoval[22], Yuriko Sosa[23] Natalie Stroud[24], y Yara Terrazas-Carafa[25].

La coordinación del trabajo está a cargo de Yanina Bellini Saibene y la edición final a cargo de Yanina Bellini Saibene y Natalia Morandeira.

[3] https://rladies.org/
[4] https://www.metadocencia.org/
[5] https://twitter.com/_lacion_
[6] https://twitter.com/MonicaLA2000
[7] https://twitter.com/Zjbb
[8] https://twitter.com/AlejaBellini
[9] https://twitter.com/yabellini
[10] https://twitter.com/July_Benitezs
[11] https://twitter.com/luucamay_
[12] https://twitter.com/spcanelon
[13] https://twitter.com/ruthy_root
[14] https://twitter.com/PaobCorrales
[15] https://twitter.com/DermitMaria
[16] https://twitter.com/anadiedrichs
[17] https://twitter.com/patriloto
[18] https://twitter.com/pmnatural
[19] https://twitter.com/Nat_Mora_
[20] https://twitter.com/_luciarp_
[21] https://twitter.com/palolili23
[22] https://twitter.com/GabySandovalM
[23] https://twitter.com/YkSosaP
[24] https://www.linkedin.com/in/natalie-stroud-63110a113/
[25] https://twitter.com/_yarena

Malena Zabalegui[26] nos aconsejó sobre el uso de lenguaje no sexista e inclusivo para la realización de esta traducción.

También generamos un glosario y diccionario bilingüe de términos de educación y tecnología[27] a partir del glosario del libro y del listado de términos a traducir (o no) del libro. El desarrollo de este glosario está a cargo de Yanina Bellini Saibene.

Todos los detalles del proceso de traducción se pueden consultar en la documentación del proyecto[28].

[26] https://www.instagram.com/malenazabalegui/
[27] https://yabellini.shinyapps.io/T3Glossary/
[28] https://github.com/gvwilson/teachtogether.tech/blob/master/es/README.md

1

Introducción

En todo el mundo han surgido comunidades de práctica para enseñar programación, diseño web, robótica y otras habilidades a **free-range learners**. Estos grupos existen para que la gente no tenga que aprender estas cosas por su cuenta, pero, irónicamente, sus fundadoras/es y docentes están muchas veces enseñándose a sí mismas/os cómo enseñar.

Hay una forma más conveniente. Así como conocer un par de cuestiones básicas sobre gérmenes y nutrición te puede ayudar a permanecer saludable, conocer un par de cosas sobre psicología cognitiva, diseño instruccional, inclusión y organización comunitaria te puede ayudar a aumentar tu efectividad como docente. Este libro presenta ideas clave que puedes usar inmediatamente, explica por qué creemos que son ciertas y te señala otros recursos que te ayudarán a ir más allá.

1.1. Quién eres

La Section 6.1 explica cómo averiguar quiénes son tus estudiantes. Los cuatro tipos de personas a las que está destinado este libro son usuarias/os finales docentes:

[1] http://carpentries.github.io/instructor-training/

[2] https://es.wikipedia.org/wiki/Uso_justo

la enseñanza no es su ocupación primaria, tienen poco o ningún conocimiento sobre pedagogía y posiblemente trabajan fuera de clases institucionales.

Emily está entrenada como bibliotecaria y actualmente trabaja como diseñadora web y gestora de proyectos en una pequeña empresa consultora. En su tiempo libre, ayuda a impartir clases de diseño para mujeres que ingresan a la tecnología como una segunda carrera. Se encuentra está reclutando colegas para dar más clases en su área y quiere saber cómo preparar lecciones que otras personas puedan usar, a la par de hacer crecer una organización de enseñanza voluntaria.

Moshe es un programador profesional, cuyos dos hijos adolescentes asisten a una escuela que no ofrece clases de programación. Se ha ofrecido como voluntario para dirigir un club de programación mensual después del horario de clases. A pesar de que expone presentaciones frecuentemente a sus colegas, no tiene experiencia de enseñanza en el aula. Quiere aprender a enseñar cómo construir lecciones efectivas en un tiempo razonable, y le gustaría aprender más acerca de los pros y contras de las clases en línea en las que cada asistente cursa a su propio ritmo.

Samira es una estudiante de robótica, que está considerando ser docente luego de graduarse. Quiere ayudar a sus pares en los talleres de robótica de fines de semana, pero nunca ha enseñado en una clase antes, y en gran medida siente el **síndrome de la impostora**. Quiere aprender más acerca de educación en general para decidir si la enseñanza es para ella y también está buscando sugerencias específicas que la ayuden a dar lecciones más efectivamente.

Gene es docente de ciencias de la computación en una universidad. Ha estado enseñando cursos de grado sobre sistemas operativos por seis años y cada vez se convence más de que tiene que haber una mejor manera de enseñar. El único entrenamiento disponible a través del centro de enseñanza y aprendizaje de su universidad es sobre publicar tareas y enviar evaluaciones en el sistema en línea de gestión del aprendizaje, por lo que quiere descubrir qué otro entrenamiento podría pedir.

Estas personas tienen *una variedad de conocimientos técnicos previos* y *alguna experiencia previa con la enseñanza*, pero *carecen de entrenamiento formal en enseñanza, diseño de lecciones u organización comunitaria*. La mayoría trabaja con *estudiantes de rango libre* y están *enfocadas en adolescentes y personas adultas* más que en niñas/os; todas estas personas *tienen tiempo y recursos limitados*. Esperamos que nuestro cuarteto use este material de la siguiente manera:

Emily tomará parte de un grupo de lectura semanal en línea con sus voluntarias.

Moshe va a abordar parte de este libro en un taller de un día durante un fin de semana y estudiará el resto por su cuenta.

Samira usará este libro en un curso de grado de un semestre que incluirá tareas, un proyecto y un examen final.

Gene leerá el libro por su cuenta en su oficina o mientras viaja en el transporte público, deseando entretanto que las universidades hagan más para apoyar la enseñanza de alta calidad.

1.2. Qué otras cosas leer

Si tienes prisa o quieres tener una idea de lo que cubrirá este libro, [Brow2018] presenta diez sugerencias basadas en evidencias para enseñar computación. También puedes disfrutar:

- El entrenamiento para instructoras/es de The Carpentries (Las Carpentries)[3], en el cual está basado este libro.
- [Lang2016] y [Hust2012], que son textos cortos y accesibles, que conectan las cosas que puedes aplicar inmediatamente con la investigación que hay detrás de ellas y las fundamenta.
- [Berg2012; Lemo2014; Majo2015; Broo2016; Rice2018; Wein2018b] están repletos de sugerencias prácticas sobre cosas que puedes hacer en tu clase, pero pueden cobrar más sentido una vez que tengas un marco conceptual para entender por qué sus ideas funcionan.
- [DeBr2015], el cual explica qué cosas son ciertas sobre la educación al explicar qué cosas no lo son y [Dida2016], que fundamenta la teoría del aprendizaje en psicología cognitiva.
- [Pape1993], que continúa siendo una visión inspiradora sobre cómo las computadoras pueden cambiar la educación. La excelente descripción de Amy Ko[4] es una síntesis de las ideas de Papert, mejor que la que podría hacer yo, y [Craw2010] es una compañía provocadora y estimulante a ambos textos.
- [Gree2014; McMi2017; Watt2014] explica por qué tantos intentos de reformas educativas han fallado a lo largo de los últimos cuarenta años, cómo colegas que trabajan sólo por dinero han explotado y exacerbado la desigualdad en nuestra sociedad, y cómo la tecnología repetidamente ha fracasado en revolucionar la educación.
- [Brow2007] y [Mann2015], porque no puedes enseñar bien sin cambiar el sistema en el que enseñamos y no puedes lograr este cambio por tu cuenta.

Quienes deseen material más académico pueden encontrar gratificante leer a

[3] http://carpentries.github.io/instructor-training/

[4] https://medium.com/bits-and-behavior/mindstorms-what-did-papert-argue-and-what-does-it-mean-for-learning-and-education-c8324b58aca4

[Guzd2015a; Hazz2014; Sent2018; Finc2019; Hpl2018], mientras que el blog de Mark Guzdial[5] ha sido consistentemente informativo, sugerente y motivador.

1.3. Agradecimientos

Este libro no existiría sin las contribuciones de Laura Acion, Jorge Aranda, Mara Averick, Erin Becker, Yanina Bellini Saibene, Azalee Bostroem, Hugo Bowne-Anderson, Neil Brown, Gerard Capes, Francis Castro, Daniel Chen, Dav Clark, Warren Code, Ben Cotton, Richie Cotton, Karen Cranston, Katie Cunningham, Natasha Danas, Matt Davis, Neal Davis, Mark Degani, Tim Dennis, Paul Denny, Michael Deutsch, Brian Dillingham, Grae Drake, Kathi Fisler, Denae Ford, Auriel Fournier, Bob Freeman, Nathan Garrett, Mark Guzdial, Rayna Harris, Ahmed Hasan, Ian Hawke, Felienne Hermans, Kate Hertweck, Toby Hodges, Roel Hogervorst, Mike Hoye, Dan Katz, Christina Koch, Shriram Krishnamurthi, Katrin Leinweber, Colleen Lewis, Dave Loyall, Paweł Marczewski, Lenny Markus, Sue McClatchy, Jessica McKellar, Ian Milligan, Julie Moronuki, Lex Nederbragt, Aleksandra Nenadic, Jeramia Ory, Joel Ostblom, Elizabeth Patitsas, Aleksandra Pawlik, Sorawee Porncharoenwase, Emily Porta, Alex Pounds, Thomas Price, Danielle Quinn, Ian Ragsdale, Erin Robinson, Rosario Robinson, Ariel Rokem, Pat Schloss, Malvika Sharan, Florian Shkurti, Dan Sholler, Juha Sorva, Igor Steinmacher, Tracy Teal, Tiffany Timbers, Richard Tomsett, Preston Tunnell Wilson, Matt Turk, Fiona Tweedie, Martin Ukrop, Anelda van der Walt, Stéfan van der Walt, Allegra Via, Petr Viktorin, Belinda Weaver, Hadley Wickham, Jason Williams, Simon Willison, Karen Word, John Wrenn, y Andromeda Yelton. También estoy agradecido a Lukas Blakk por el logotipo, a Shashi Kumar por la ayuda con LaTeX, a Markku Rontu por hacer que los diagramas se vean mejor, y a toda aquella persona que ha usado este material a lo largo de los años. Cualquier error que permanezca es mío.

1.4. Ejercicios

Cada capítulo finaliza con una variedad de ejercicios que incluyen un formato sugerido y cuánto tiempo toma usualmente hacerlos en persona. Muchos pueden ser usados en otros formatos —en particular, si estás avanzando en este libro por tu cuenta, puedes todavía hacer muchos de los ejercicios que son destinados a grupos— y siempre puedes dedicar más tiempo a un ejercicio que el que sugiero.

Si estás usando este material en un taller de formación docente, puedes darles los siguientes ejercicios a quienes participen, con uno o dos días de anticipación,

[5] http://computinged.wordpress.com

para que tengan una idea de quiénes son y cuál es la mejor manera en que les puedes ayudar. Por favor lee las advertencias en la Section 9.4 antes de hacer estos ejercicios.

Altos y bajos (clase completa/5')

Escribe respuestas breves a las siguientes preguntas y compártelas con tus pares. (Si estás tomando notas colaborativas en línea como se describe en la Section 9.7, puedes escribir tus respuestas allí.)

1. ¿Cuál es la mejor clase o taller que alguna vez hayas tomado? ¿Qué la hacía tan buena?
2. ¿Cuál fue la peor? ¿Qué la hacía tan mala?

Conócete (clase completa/5')

Comparte respuestas breves a las siguientes preguntas con tus pares. Guarda tus respuestas para que puedas regresar a ellas como referencia a la par que avanzas en el estudio de este libro.

1. ¿Qué es lo que más quieres enseñar?
2. ¿A quiénes tienes más ganas de enseñarles?
3. ¿Por qué quieres enseñar?
4. ¿Cómo sabrás si estás enseñando bien?
5. ¿Qué es lo que más ansías aprender acerca de enseñanza y aprendizaje?
6. ¿Qué cosa específica crees que es cierta acerca de enseñanza y aprendizaje?

¿Por qué aprender a programar? (individual/20')

Las/los políticas/os, líderes de negocios y educadoras/es usualmente dicen que la gente debe aprender a programar porque los trabajos del futuro lo requerirán. Sin embargo, como Benjamin Doxtdator ha señalado[6], muchas de estas afirmaciones están construidas sobre cimientos débiles. Incluso si las afirmaciones fueran reales, la educación no debería preparar a la gente para los trabajos del futuro: les debería dar el poder de decidir qué tipos de trabajos hay y asegurarles que vale la pena hacer esos trabajos. Además, como señala Mark Guzdial[7], hay realmente muchas razones para aprender a programar:

1. Para entender nuestro mundo.

[6] http://www.longviewoneducation.org/field-guide-jobs-dont-exist-yet/

[7] https://computinged.wordpress.com/2017/10/18/why-should-we-teach-programming-hint-its-not-to-learn-problem-solving/

2. Para estudiar y entender procesos.
3. Para ser capaz de hacer preguntas sobre las cosas que influyen en nuestras vidas.
4. Para usar una importante nueva forma de alfabetización.
5. Para tener una nueva manera de aprender arte, música, ciencia y matemática.
6. Como una habilidad laboral.
7. Para usar mejor las computadoras.
8. Como un medio en el cual aprender resolución de problemas.

Dibuja una grilla de 3×3 cuyos ejes estén etiquetados: “baja,” “media,” y “alta” y coloca cada razón en un sector de acuerdo a la importancia que tienen para ti (el eje X) y para la gente a la que planeas enseñar (el eje Y).

1. ¿Qué puntos están estrechamente alineados en importancia (es decir, en la diagonal de tu grilla)?
2. ¿Qué puntos están desalineados (es decir, en las esquinas por fuera de la diagonal)?
3. ¿Cómo debería afectar esto lo que tú enseñas?

2

Modelos mentales y evaluación formativa

La primera tarea en la enseñanza es descifrar quiénes son tus estudiantes. Nuestra aproximación está basada en el trabajo de investigadores/as como Patricia Benner, quien estudió cómo las personas progresan de novatas a expertas en la carrera de enfermería [Benn2000]. Benner identificó cinco etapas de desarrollo cognitivo que la mayor parte de la gente atraviesa de forma bastante consistente. Para nuestros propósitos, simplificamos esta evolución en tres etapas:

Personas novatas no saben qué es lo que no saben, es decir, aún no tienen un modelo mental utilizable del dominio del problema.

Practicantes competentes tienen un modelo mental que es adecuado para los propósitos diarios. Pueden llevar a cabo tareas normales con un esfuerzo normal bajo circunstancias normales y tienen algún entendimiento de los límites de su conocimiento (es decir, saben lo que no saben).

Personas expertas tienen modelos mentales que incluyen excepciones y casos especiales, los cuales les permiten manejar situaciones que están por fuera de lo ordinario. Discutiremos sobre la experiencia o pericia en más detalle en el Chapter 3.

Entonces, ¿qué *es* un **modelo mental**? Como el nombre lo sugiere, es una representación simplificada de las partes más importantes de algún dominio del problema; que a pesar de ser simplificada es suficientemente buena para permitir la resolución del problema. Un ejemplo es el modelo molecular de bolas y varillas que se usa en las clases de química de la escuela. Los átomos no son en realidad bolas y las uniones atómicas no son en realidad varillas, pero el modelo permite a la gente razonar sobre los componentes químicos y sus reacciones. Un modelo más sofisticado de un átomo es aquel con una bola pequeña en el centro (el núcleo) rodeada de electrones orbitantes. También es incorrecto, pero la complejidad extra le permite a la gente explicar más y resolver más problemas. (Como con el software, los modelos mentales nunca son finalizados: simplemente son utilizados.)

Presentar a personas novatas un montón de hechos es contraproducente porque aún no tienen un modelo donde ubicarlos. Incluso, presentar demasiados hechos muy pronto puede reforzar el modelo mental incorrecto que han improvisado. Como observó [Mull2007a] en un estudio sobre video-instrucción para estudiantes de ciencia:

> Los/las estudiantes tienen ideas existentes acerca de… los fenómenos antes de ver un video. Si el video presenta… conceptos de una forma clara, bien

> ilustrada, los/las estudiantes creen que están aprendiendo, pero no se involucran con el video en un nivel suficientemente profundo como para darse cuenta de que lo que se les ha presentado difiere de sus conocimientos previos... Sin embargo, hay esperanza. Se ha demostrado que el aprendizaje aumenta al presentar en un video ... las concepciones erróneas comunes de los/las estudiantes junto con los... conceptos a enseñar, ya que aumenta el esfuerzo mental que los/las estudiantes realizan mientras miran el video.

Tu objetivo cuando enseñes a personas novatas debe por lo tanto ser ayudarles a construir un modelo mental para que tengan algún lugar en el que ordenar los hechos. Por ejemplo, la lección sobre la consola Unix[1] de Software Carpentry introduce quince comandos en tres horas. Eso sería un comando cada doce minutos, lo que parece muy lento hasta que te das cuenta de que el propósito real de la lección no es enseñar esos quince comandos: es enseñar las rutas de acceso, la historia de comandos, el autocompletado con el tabulador, los comodines, los *pipes* los argumentos de la línea de comando y las redirecciones. Los comandos específicos no tienen sentido hasta que las personas novatas entienden estos conceptos; una vez que lo hacen, pueden empezar a leer manuales, pueden buscar las palabras claves correctas en la web, y pueden decidir si los resultados de sus búsquedas son útiles o no.

Las diferencias cognitivas entre personas novatas y practicantes competentes apuntalan las diferencias entre dos tipos de materiales educativos. Un **tutorial** ayuda a construir un modelo mental a quienes recién llegan a un determinado campo; un **manual**, por otro lado, ayuda a practicantes competentes a llenar los baches de su conocimiento. Los tutoriales frustran a practicantes competentes porque avanzan demasiado lento y dicen cosas que son obvias (aunque no son *para nada* obvios para personas novatas). De la misma manera, los manuales frustran a las personas novatas porque usan jergas y *no* explican las cosas. Este fenómeno se llama el **el efecto inverso de la experiencia** [Kaly2003], y es una de las razones por las que tienes que decidir tempranamente para quiénes son tus lecciones.

Un puñado de excepciones

Una de las razones por las que Unix y C se hicieron populares es que [Kern1978; Kern1983; Kern1988] de alguna manera consiguieron tener buenos tutoriales y *buenos manuales al mismo tiempo. [Fehi2008] y [Ray2014] están entre los otros pocos libros de computación que consiguieron esto; incluso luego de re-leerlos varias veces, no sé cómo lo lograron.*

[1] https://swcarpentry.github.io/shell-novice-es/

2.1. ¿Están aprendiendo tus estudiantes?

Mark Twain escribió: "No es lo que no sabes lo que te mete en problemas. Es lo que tienes seguridad de saber y simplemente no es así." Uno de los ejercicios al construir un modelo mental es, por lo tanto, despejar las cosas que *no* pertenecen al modelo. En sentido amplio, las concepciones erróneas de las personas novatas caen en tres categorías:

Errores fácticos como creer que Río de Janeiro es la capital de Brasil (es Brasilia). Estos errores generalmente son fáciles de corregir.

Modelos rotos como creer que el movimiento y la aceleración deben estar en la misma dirección. Podemos lidiar con estos errores haciendo que las personas novatas razonen a través de ejemplos en los que sus modelos den una respuesta incorrecta.

Creencias fundamentales como por ejemplo "el mundo solo tiene algunos miles de años de antigüedad" o "algunos tipos de personas son naturalmente mejores en programación que otros" [Guzd2015b; Pati2016]. Estos errores están generalmente conectados profundamente con la identidad social del/de la estudiante, por lo que resisten a las evidencias y racionalizan las contradicciones.

La gente aprende más rápido cuando los/las docentes identifican y aclaran los conceptos erróneos de sus estudiantes mientras se está dando la lección. Esto se llama **evaluación formativa** porque forma (o le da forma a) la enseñanza mientras se está llevando a cabo. Los/as estudiantes no aprueban o reprueban una evaluación formativa. En cambio, la evaluación formativa da, tanto a quien enseña como a quien aprende, retroalimentación sobre qué tan bien les está yendo y en qué se deberían enfocar en los próximos pasos. Por ejemplo, un/una docente de música le puede pedir a un/una estudiante que toque una escala muy lentamente para chequear su respiración. Entonces, el/la estudiante averigua si la respiración es correcta, mientras que el/la docente recibe una devolución sobre si la explicación que acaba de dar fue comprendida.

En resumen

*El contrapunto de la evaluación formativa es la **evaluación sumativa**, que tiene lugar al final de la lección. La evaluación sumativa es como un examen de conducir: le dice a quien está aprendiendo a conducir si ha dominado el tópico y a quien le está enseñando si su lección ha sido exitosa. Una forma de pensar la diferencia entre los dos tipos de evaluaciones es que quien prueba la comida mientras cocina está haciendo evaluaciones formativas, mientras que quien es comensal y prueba la comida cuando se le sirve está haciendo una evaluación sumativa. Desafortunadamente, la escuela ha entrenado a la mayoría de la gente para creer que toda evaluación es sumativa, es decir, que si algo se siente como un examen, le va a*

> *jugar en contra resolverlo pobremente. Hacer que las evaluaciones formativas se sientan informales reduce esta ansiedad; en mi experiencia, usar cuestionarios en línea, o donde deba hacerse click, o cualquier cosa semejante parece aumentar la ansiedad, ya que hoy en día la mayoría de la gente cree que todo lo que hace en internet está siendo mirado y grabado.*

Para ser útil durante la enseñanza, una evaluación formativa debe ser rápida de administrar (de manera que no rompa el flujo de la lección) y debe tener una respuesta correcta no ambigua (de manera que pueda ser usada en grupos). El tipo de evaluación formativa más ampliamente usado es probablemente el cuestionario de opciones múltiples (COM). Muchos y muchas docentes tienen una mala opinión de ellos, pero cuando están bien diseñados pueden revelar mucho más que si alguien sabe o no algunos hechos específicos. Por ejemplo, supón que estás enseñando a niños y niñas cómo hacer sumas de números con múltiples dígitos [Ojos2015] y les das este COM:

¿Cuánto es 37 + 15?
a) 52
b) 42
c) 412
d) 43

La respuesta correcta es 52, pero las otras respuestas proporcionan información valiosa:

- Si el/la niño/a elige 42, no entiende qué significa "llevarse" una unidad. (Podría escribir 12 como respuesta a 7+5, pero luego reemplazaría el 1 con el 4 que obtiene de la suma de 3+1.)
- Si elige 412, está tratando a cada columna de números como un problema separado. Esto sigue siendo incorrecto, pero es incorrecto por un motivo distinto.
- Si elige 43, entonces sabe que tiene que llevarse el 1, pero lo lleva de vuelta a la columna de donde viene. De nuevo, es un error distinto que requiere de una explicación clarificadora diferente por parte de quien enseña.

Cada una de estas respuestas incorrectas es un **distractor plausible** con **poder diagnóstico**. Un distractor es una respuesta incorrecta o peor que la mejor respuesta; "plausible" significa que parece que podría ser correcta, mientras que "poder diagnóstico" significa que cada uno de los distractores ayuda al docente a darse cuenta de qué explicar a continuación a estudiantes particulares.

La variedad de respuestas a una evaluación formativa te guía cómo continuar. Si una cantidad suficiente de la clase tiene la respuesta correcta, avanzas. Si la mayoría de la clase elige la misma respuesta incorrecta, deberías retroceder y trabajar en corregir la concepción errónea que ese distractor señala. Si las respuestas de la clase se dividen equitativamente entre varias opciones, probablemente están arriesgando, entonces deberías retroceder y re-explicar la idea de una manera distinta. (Repetir

exactamente la misma explicación probablemente no será útil, lo cual es uno de los motivos por los que tantos cursos por video son pedagógicamente ineficientes.)

¿Qué pasa si la mayoría de la clase vota por la respuesta correcta pero un grupo pequeño vota por las incorrectas? En ese caso, tienes que decidir si deberías destinar tiempo a que la minoría entienda o si es más importante mantener a la mayoría cautivada. No importa cuán duro trabajes o qué prácticas de enseñanza uses, no siempre vas a conseguir darle a todos y todas lo que necesitan; es tu responsabilidad como docente tomar la decisión.

¿De dónde vienen las respuestas incorrectas?

Para diseñar distractores plausibles, piensa en las preguntas que tus estudiantes hicieron o en los problemas que tuvieron la última vez que enseñaste esta temática. Si no la has enseñado antes, piensa en tus propios conceptos erróneos, pregúntale a colegas sobre sus experiencias o busca la historia de tu campo temático: si las demás personas tuvieron los mismos malentendidos sobre tu temática cincuenta años atrás, hay chances de que la mayoría de tus estudiantes aún malentiendan la temática de la misma forma al día de hoy. También puedes hacer preguntas abiertas en clase para recoger las concepciones erróneas sobre los temas que vas a abarcar en una clase posterior, o consultar sitios de preguntas y respuestas como Quora[2] o Stack Overflow[3] para ver con qué se confunden quienes aprenden la temática en cualquier otro lugar.

Desarrollar evaluaciones formativas hace mejor tus lecciones porque te fuerza a pensar en los modelos mentales de tus estudiantes. En mi experiencia, al pensar evaluaciones formativas automáticamente escribo la lección de forma de abarcar los baches y errores más probables. Las evaluaciones formativas, por lo tanto, dan buenos resultados incluso si no son utilizadas (aunque la enseñanza es más efectiva cuando sí se utilizan).

Los COMs no son el único tipo de evaluación formativa: el Chapter 12 describe otros tipos de ejercicios que son rápidos y no son ambiguos. Cualquiera sea la evaluación que escojas, deberías hacer algo que tome un minuto o dos cada 10–15 minutos de manera de asegurarte que tus estudiantes están realmente aprendiendo. Este ritmo no está basado en un límite de atención intrínseco: [Wils2007] encontró poca evidencia a favor de la afirmación usualmente repetida de que los/las estudiantes solo pueden prestar atención durante 10–15 minutos. En cambio, la guía asegura que, si un número significativo de personas se ha quedado atrás, solo tienes que repetir una pequeña porción de la lección. Las evaluaciones formativas frecuentes también mantienen el compromiso de tus estudiantes, particularmente si se involucran en una discusión en un grupo pequeño. (Section 9.2).

Las evaluaciones formativas también pueden ser usadas *antes* de las lecciones. Si comienzas una clase con un COM y toda la clase lo contesta correctamente, puedes evitar explicar algo que tus estudiantes ya saben. Este tipo de **enseñanza activa** te

[2] http://www.quora.com
[3] https://stackoverflow.com/

da más tiempo para enfocarte en las cosas que tus estudiantes no saben. También le muestra a tus estudiantes que respetas su tiempo lo suficiente como para no desperdiciarlo, lo que ayuda a la motivación (Chapter 10).

Inventario de conceptos

Con una cantidad de datos suficiente, los COMs pueden ser sorprendentemente precisos. El ejemplo más conocido es el ***inventario del concepto de fuerza*** *[Hest1992], que evalúa la comprensión sobre los mecanismos básicos newtonianos. Al entrevistar un gran número de participantes, correlacionando sus concepciones erróneas con los patrones de respuestas correctas e incorrectas, así como mejorando las preguntas, los creadores de este inventario construyeron una herramienta de diagnóstico que permite identificar concepciones erróneas específicas. Las personas que investigan pueden utilizar dicha herramienta para medir el efecto de los cambios en los métodos de enseñanza [Hake1998]. Tew y colaboradores desarrollaron y validaron una evaluación independiente del lenguaje para programación introductoria [Tew2011]; [Park2016] la replicaron y [Hamo2017] está desarrollando un inventario de conceptos sobre la recursividad. Sin embargo, es muy costoso construir herramientas de este tipo y su validez está cada vez más amenazada por la habilidad de los/las estudiantes para buscar respuestas en línea.*

Desarrollar evaluaciones formativas en una clase solo requiere un poco de preparación y práctica. Puedes darles a tus estudiantes tarjetas coloreadas o numeradas para que respondan un COMs simultáneamente (en lugar de que tengan que levantar sus manos en turnos), incluyendo como una de las opciones “No tengo idea” y alentándoles a hablar con sus pares más cercanos por un par de segundos antes de responder. Todas estas prácticas te ayudarán a asegurar que el flujo de enseñanza no sea interrumpido. La Section 9.2 describe un método de enseñanza poderoso y basado en evidencias, construido a partir de estas simples ideas.

Humor

Los/las docentes a veces incluyen respuestas supuestamente tontas en los COMs, como “¡mi nariz!”, particularmente en los cuestionarios destinados a estudiantes jóvenes. Sin embargo, estas respuestas no proveen ninguna idea sobre las concepciones erróneas de los/las estudiantes, y la mayoría de la gente no las encuentran graciosas. Como regla, deberías solo incluir un chiste en una lección si lo encuentras gracioso la tercera vez que lo vuelves a leer.

Las evaluaciones formativas de una lección deberían preparar a los/las estudiantes para su evaluación sumativa: nadie debería encontrar nunca una pregunta en un examen para la cual la enseñanza no lo/la ha preparado. Esto no significa que nunca debes incluir nuevos tipos de problemas en un examen, pero, si lo haces, deberías de antemano haberles dado a tus estudiantes prácticas para abordar problemas nuevos. El Chapter 6 explora este punto en profundidad.

2.2. Máquina nocional

El término **pensamiento computacional** está muy extendido, en parte porque la gente coincide en que es importante aún cuando con el mismo término se suele referir a cosas muy distintas. En vez de discutir qué incluye y qué no incluye el término, es más útil pensar en la **máquina nocional** que quieres que tus estudiantes entiendan [DuBo1986]. De acuerdo a [Sorv2013], una máquina nocional:

- es una abstracción idealizada del *hardware* de computación y de otros aspectos de los entornos del tiempo de ejecución de los programas;
- permite describir la semántica de los programas; y
- refleja correctamente qué hacen los programas cuando son ejecutados.

Por ejemplo, mi máquina nocional para Python es:

1. Ejecutar programas en el momento en la memoria, la cual se divide en la pila de llamadas y el *heap* .
2. La memoria para los datos siempre es asignada desde el *heap*.
3. Cada conjunto de datos se almacena en una estructura de dos partes. La primera parte dice de qué tipo de datos se trata y la segunda parte es el valor real.
4. Booleanos, números y caracteres de texto nunca son modificados una vez que se crean.
5. Las listas, conjuntos y otras colecciones almacenan referencias a otros datos en lugar de almacenar estos valores directamente. Pueden ser modificados una vez que se crean, i.e. una lista puede ser ampliada o nuevos valores pueden ser agregados a un conjunto.
6. Cuando un código se carga a la memoria, Python lo convierte a una secuencia de instrucciones que son almacenadas como cualquier otro tipo de dato. Este es el motivo por el que es posible asignar funciones a variables y luego pasarlas como parámetros.
7. Cuando un código es ejecutado, Python sigue las instrucciones paso a paso, haciendo lo que cada instrucción le indica de a una por vez.
8. Algunas instrucciones hacen que Python lea datos, haga cálculos y cree nuevos datos. Otras instrucciones controlan qué instrucciones ejecuta Python, que es el modo en que los bucles y condicionales trabajan. Otra instrucción le indica a Python que llame a una función.
9. Cuando se llama a una función, Python coloca un nuevo marco de pila en la pila de llamadas.

10. Cada marco de pila almacena los nombres de las variables y las referencias a los datos. Los parámetros de las funciones son simplemente otro tipo de variable.

11. Cuando una variable es utilizada, Python la busca en el marco de la pila superior Si no la encuentra allí, busca en el último marco en la memoria global.

12. Cuando la función finaliza, Python la borra del marco del de pila y vuelve a las instrucciones que estaba ejecutando antes de llamar a la función. Si no encuentra un "antes," el programa ha finalizado.

Uso esta versión caricaturizada de la realidad siempre que enseño Python. Después de 25 horas de instrucción y 100 horas de trabajar en su propio tiempo, espero que la mayor parte del grupo tenga un modelo mental que incluya todas o la mayoría de estas características.

2.3. Ejercicios

Tus modelos mentales (pensar-parejas-compartir/15)

¿Cuál es el modelo mental que usas para entender tu trabajo? Escribe unas pocas oraciones para describirlo y hazle una devolución a tu pareja sobre su modelo mental. Una vez que has hecho esto en pareja, algunas pocas personas de la clase compartirán sus modelos con el grupo completo. ¿Está todo el grupo de acuerdo sobre qué es un modelo mental? ¿Es posible dar una definición precisa?, ¿o el concepto es útil justamente porque es difuso?

Síntomas de ser una persona novata (clase completa/5)

Decir que las personas novatas no tienen un modelo mental de un dominio particular no es equivalente a decir que no tienen ningún modelo mental. Las personas novatas tienden a razonar por analogía y arriesgan conjeturas: toman prestado fragmentos y partes de modelos mentales de otros dominios que superficialmente parecen similares.

La gente que hace esto generalmente dice cosas queni siquiera son falsas[4]. Como clase, discutir qué otros síntomas hay de ser una persona novata. ¿Qué dice o hace una persona para llevarte a clasificarla como novata en algún dominio?

Modelar modelos mentales de las personas novatas (parejas/20)

Crea un cuestionario de múltiples opciones relacionado a un tópico que has enseñado o pretendas enseñar y explica el poder diagnóstico de cada uno de sus distractores (i.e. qué concepción errónea pretende ser identificada con cada distractor).

[4]https://es.wikipedia.org/wiki/Ni_siquiera_es_falso

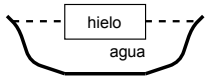

Figura 2.1: Hielo en un recipiente

Una vez que hayas finalizado, intercambia COMs con tu pareja. ¿Son sus preguntas ambiguas? ¿Son las concepciones erróneas plausibles? ¿Los distractores realmente evalúan esas concepciones erróneas? ¿Hay otras posibles concepciones erróneas que *no* sean evaluadas?

Pensar en las cosas (clase completa/15)

Una buena evaluación formativa requiere que la gente piense profundamente en un problema. Por ejemplo, imagina que has colocado un bloque de hielo en un recipiente y luego llenas de agua este recìpiente, hasta el borde. Cuando el hielo se derrite, ¿el nivel de agua aumenta (de manera que el recipiente rebasa)?, ¿el nivel de agua baja?, ¿o se mantiene igual(Figure 2.1)?

La solución correcta es que el nivel de la tina permanece igual: el hielo desplaza a su propio peso en el agua, por lo que completa exactamente el "agujero" que ha dejado al derretirse. Para darse cuenta del porqué la gente construye un modelo de la relación entre el peso, el volumen y la densidad [Epst2002].

Describe otra evaluación formativa que hayas visto o hayas utilizado, alguna que consideres que lleve a los/las estudiantes a pensar profundamente en algo, y por lo tanto ayude a identificar los defectos en sus razonamientos.

Cuando hayas finalizado, explícale tu ejemplo a a otra persona de tu grupo y dale una devolución sobre su ejemplo.

Una progresión diferente (individuos/15)

El modelo de desarrollo de habilidades de persona novata-competente-experta es a veces llamado modelo Dreyfus[5]. Otra progresión comúnmente utilizada es el modelo de las cuatro etapas de la competencia[6]:

Incompetencia inconsciente: la persona no sabe lo que no sabe.

Incompetencia consciente: la persona se da cuenta de que no sabe algo.

Competencia consciente: la persona ha aprendido cómo hacer algo, pero solo lo puede hacer mientras mantiene su concentración y quizás aún deba dividir la tarea en varios pasos..

Competencia inconsciente: la habilidad se ha transformado en una segunda naturaleza y la persona puede realizarla reflexivamente.

[5]https://es.qwe.wiki/wiki/Dreyfus_model_of_skill_acquisition
[6]https://es.wikipedia.org/wiki/Cuatro_etapas_de_competencia

Identifica una temática en la que te encuentres en cada uno de los niveles de desarrollo de habilidades. En la materia que enseñas, ¿en qué nivel están usualmente la mayoría de tus estudiantes? ¿Qué nivel estás tratando que alcancen? ¿Cómo se relacionan estas cuatro etapas con la clasificación persona novata-competente-experta?

¿Qué tipo de computación? (individuos/10)

[Tedr2008] resume tres tradiciones en computación:

Matemática: Los programas son la encarnación de los algoritmos. Son correctos o incorrectos, así como más o menos eficientes.

Científica: Los programas son modelos de procesos de información más o menos adecuados que pueden ser estudiados usando el método científico.

Ingenieril: Los programas son objetos que se construyen, tales como los diques y los aviones, y son más o menos efectivos y confiables.

¿Cuál de estas tradiciones coincide mejor con tu modelo mental de la computación? Si ninguna de ellas coincide, ¿qué modelo tienes?

Explicar por qué no (parejas/5)

Un/a estudiante de tu curso piensa que hay algún tipo de diferencia entre el texto que tipea carácter por carácter y el texto idéntico que copia y pega. Piensa en una razón por la que tu estudiante puede creer esto o en algo que pueda haber sucedido para darle esa impresión. Luego, simula ser esa persona mientras tu pareja te explica por qué no es así. Intercambia roles con tu pareja y vuelve a probar.

Tu modelo ahora (clase completa/5)

Como clase, creen una lista de elementos clave de tu modelo mental de enseñanza. ¿Cuál es la media docena de conceptos más importantes y cómo se relacionan?

Tus máquinas nocionales (grupos pequeños/20)

En grupos pequeños, escriban una descripción de la máquina nocional que quieren que sus estudiantes usen para entender cómo corren sus programas. ¿En qué difiere una máquina nocional para un lenguaje basado en bloques como Scratch de la máquina nocional para Python? ¿Y en qué difiere de una máquina nocional para hojas de cálculo o para un buscador que está interpretando HTML y CSS cuando renderiza una página web?

Disfrutar sin aprender (individuos/5)

Muchos estudios han mostrado que las evaluaciones de la enseñanza no se correlacionan con los resultados de la enseñanza [Star2014; Uttl2017], i.e. cuán alto sea

el puntaje del grupo de estudiantes en un curso no predice cuánto recuerdan. ¿Alguna vez has disfrutado de una clase en la que en realidad no has aprendido nada? Si la respuesta es sí, ¿qué hizo que disfrutes esa clase?

Revisión

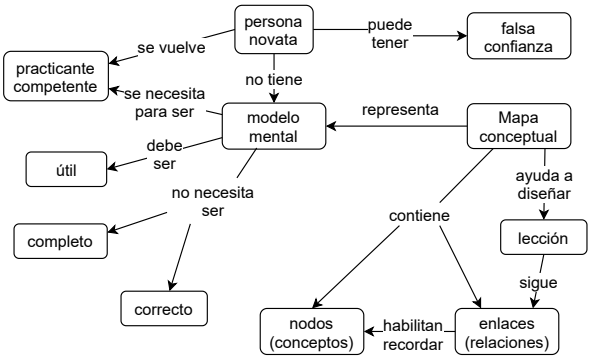

Figura 2.2: Conceptos: modelos mentales

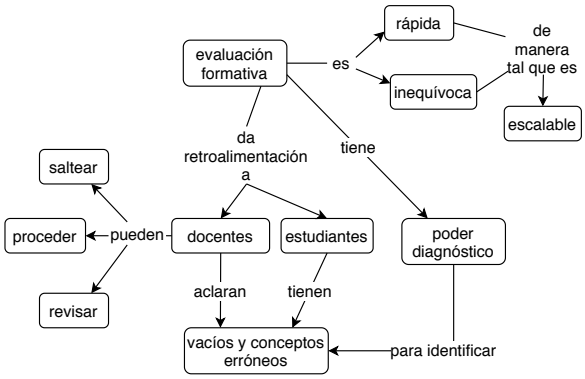

Figura 2.3: Conceptos: evaluación

3

Pericia y memoria

> La memoria es el remanente del pensamiento.
> — Daniel Willingham, *Por qué a tus estudiantes no les gusta la escuela(Why Students Don't Like School)*

El capítulo anterior explicaba las diferencias entre personas novatas y practicantes competentes. En este capítulo se observa la pericia: qué es, cómo se puede adquirir, y cómo puede ser tanto perjudicial como de ayuda. Luego introduciremos uno de los límites más importantes en el aprendizaje y analizaremos cómo el dibujar nuestros modelos mentales nos puede ayudar a convertir el conocimiento en lecciones.

Para empezar, ¿a qué nos referimos cuando decimos que alguien es una persona experta? La respuesta habitual es que puede resolver problemas mucho más rápido que la persona que es "simplemente competente", o que puede reconocer y entender casos donde las reglas normales no se pueden aplicar. Es más, de alguna manera una persona experta hace que parezca que resolver ciertos problemas no requiere esfuerzo alguno: en muchos casos, parece saber la respuesta correcta de un vistazo [Parn2017].

Pericia es más que solo conocer más hechos: las/los practicantes competentes pueden memorizar una gran cantidad de trivialidades sin mejorar notablemente sus desempeños. En cambio, imagina por un momento que almacenamos conocimiento como una red o grafo en el cual los hechos son nodos y las relaciones son arcos[1]. La diferencia clave entre personas expertas y practicantes competentes es que los modelos mentales de las personas expertas están mucho más densamente conectados, es decir, es más probable que conozcan una conexión entre dos hechos cualesquiera.

La metáfora del grafo explica por qué ayudar a tus estudiantes a hacer conexiones es tan importante como presentarles los hechos: sin esas conexiones la gente no puede recordar y usar aquello que sabe. Esta metáfora también explica varios aspectos observados del comportamiento experto:

- Las personas expertas pueden saltar directamente de un problema a una solución porque realmente existe una conexión directa entre ambas cuestiones en sus mentes. Mientras un practicante competente debería razonar $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$, una persona experta puede ir de A a E en un único paso. Esto lo llamamos **intuición**: en vez de razonar su camino hacia una solución, la persona experta reconoce una solución de la misma manera que reconocería una cara familiar.

[1] Esto definitivamente *no* es como nuestro cerebro trabaja, pero es una metáfora útil.

- Los grafos densamente conectados son también la base para la **representación fluida** de las personas expertas, es decir sus habilidades para cambiar una y otra vez entre distintas formas de ver un problema [Petr2016]. Por ejemplo, tratando de resolver un problema en matemáticas, una persona experta puede cambiar entre abordarlo de manera geométrica y representarlo como un conjunto de ecuaciones.

- Esta metáfora también explica por qué las personas expertas son mejores en diagnósticos que las/los practicantes competentes: mayor cantidad de conexiones entre hechos hace más fácil razonar hacia atrás, de síntomas a causas. (Esta es la razón de por qué durante una entrevista de trabajo es preferible pedirle a programadoras/es que depuren un programa a pedirles que programen: da una impresión más precisa de su habilidad)

- Finalmente, las personas expertas están muchas veces tan familiarizadas con su tema que no pueden imaginarse cómo puede ser *no* ver el mundo de esa manera. Esto significa que muchas veces están menos capacitadas para enseñar un tema que personas con menor experiencia, que aún recuerdan cómo lo han aprendido.

El último de estos puntos se llama **punto ciego de las personas expertas**. Como se definió originalmente en [Nath2003], es la tendencia de las personas expertas a organizar una explicación de acuerdo a los principios principales del tema en lugar de guiarse por aquello que ya conocen quienes están aprendiendo. Esto se puede superar con entrenamiento, pero es parte de la razón por la cual no hay correlación entre lo bien que investiga alguien en un área y lo bien que esa misma persona enseña la temática [Mars2002].

La letra S

Las personas expertas a menudo caen en sus puntos ciegos usando la palabra "solo," como en, "Oh, es fácil, solo enciendes una nueva máquina virtual y luego solo instalas estos cuatro parches a Ubuntu y luego solo reescribes todo tu programa en un lenguaje funcional puro." Como discutimos en Chapter 10, hacer esto indica que quien habla piensa que el problema es trivial y por lo tanto la persona que lucha con el problema debe ser estúpida: entonces, no tengas esta actitud.

3.1. Mapas conceptuales

La herramienta que elegimos para representar el modelo mental de alguien es un **mapa conceptual**, en el cual los hechos son burbujas y las conexiones son relaciones etiquetadas. Como ejemplos, Figure 3.1 muestra por qué la Tierra tiene estaciones (de

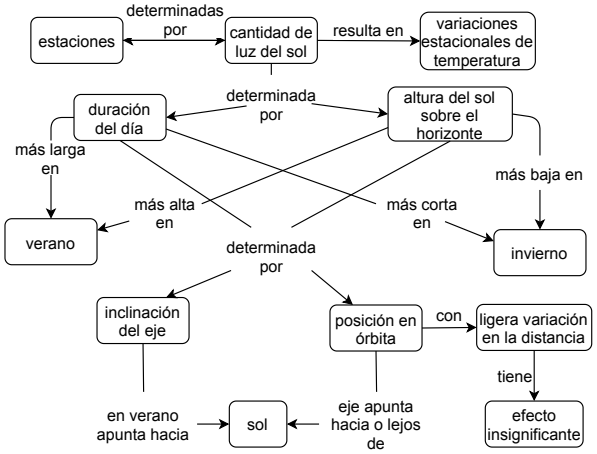

Figura 3.1: Mapa conceptual para estaciones

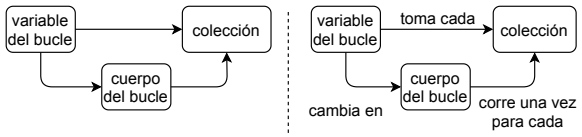

Figura 3.2: Mapa conceptual para un bucle `for`

IHMC[2]), y Appendix G presenta mapas conceptuales de librerías desde tres puntos de vista distintos.

Para mostrar cómo pueden ser usados los mapas conceptuales para enseñar programación, considera este `for` bucle en Python:

```
for letter in "abc":
    print(letter)
```

cuya salida es:

```
a
b
c
```

Los tres "cosas" clave en este bucle se muestran al principio de Figure 3.2, pero son solo la mitad de la historia. La versión ampliada en la parte inferior muestra las relaciones entre esas cosas, las cuales son tan importantes para la comprensión como los conceptos en sí mismos.

[2]https://cmap.ihmc.us/

Los mapas conceptuales pueden ser usados de varias maneras:

Para ayudar a docentes a descubrir qué están tratando de enseñar. Un mapa conceptual separa el contenido del orden: en nuestra experiencia, las personas rara vez terminan enseñando las cosas en el orden en que las dibujaron por primera vez.

Para mejorar la comunicación entre quienes diseñan las lecciones Si dos docentes tienen ideas muy diferentes de aquello que están tratando de enseñar, es probable que arrastren a sus estudiantes en diferentes direcciones. Dibujar y compartir mapas conceptuales puede ayudar a prevenirlo. Y sí, personas diferentes pueden tener mapas conceptuales diferentes para el mismo tema, pero el mapeo conceptual hace explícitas estas diferencias.

Para mejorar la comunicación con estudiantes. Si bien es posible dar a tus estudiantes un mapa pre-dibujado al inicio de la lección para que puedan anotar, es mejor dibujarlo parte por parte mientras se está enseñando, para reforzar la relación entre lo que muestra el mapa y lo que tú dices. Volveremos a esta idea en Section 4.1.

Para evaluación. Hacer que las/los estudiantes dibujen lo que creen que acaban de aprender le muestra a quien enseña lo que se pasó por alto y lo que se comunicó mal. Revisar los mapas conceptuales de estudiantes insume demasiado tiempo para utilizarlo como una evaluación formativa durante las clases, pero es muy útil en clases semanales *una vez que las/los estudiantes están familiarizadas/os con la técnica*. La calificación es necesaria porque cualquier manera nueva de hacer algo inicialmente enlentece a la gente—si quien está aprendiendo intenta encontrarle el sentido a la programación básica, pedirle que se imagine cómo esquematizar sus pensamientos al mismo tiempo es una carga no conveniente.

Algunas/os docentes son escépticos a que las personas novatas puedan mapear efectivamente lo que entendieron, dado que la introspección y la explicación de lo entendido son generalmente habilidades más avanzadas que la comprensión en sí misma. Por ejemplo, [Kepp2008] observó el uso del mapeo conceptual en la enseñanza de computación. Uno de los hallazgos fue que "… el mapeo conceptual es problemático para muchas/os estudiantes porque evalúa la comprensión personal en lugar del conocimiento que simplemente se aprendió de memoria." Como alguien que valora la comprensión sobre el conocimiento de memoria, yo lo considero un beneficio.

Comienza por cualquier lugar

Cuando se pide por primera vez dibujar un mapa conceptual, muchas personas no saben por dónde empezar. Si esto ocurre, escribe dos palabras asociadas con el tema que estás tratando de mapear, luego dibuja una línea entre ellas y agrega una etiqueta explicando cómo estas dos ideas están relacionadas. Puedes entonces preguntarte qué otras cosas están relacionadas en el mismo sentido, qué partes tienen esas cosas, o qué sucede antes o después con los conceptos que ya están en la hoja a fin de descu-

brir más nodos y arcos. Después de eso, casi siempre la parte más difícil está terminada.

Los mapas conceptuales son solo una forma de representar nuestro conocimiento de un tema [Eppl2006]; otros incluyen diagramas de Venn, diagramas de flujo y árboles de decisión [Abel2009]. Todos ellos **externalizan la cognición**, es decir hacen visibles los modelos mentales de manera que pueden ser comparados y combinados[3].

Trabajo crudo y honestidad

Muchas/os diseñadoras/es de interfaces de usuario creen que es mejor mostrar a la gente bocetos de sus ideas en lugar de maquetas pulidas porque estiman que las personas dan una opinión más honesta sobre algo que consideran solo ha requerido unos pocos minutos crear: si parece que lo que están criticando tardó horas en hacerse, la mayoría suavizará sus golpes. Al dibujar mapas de concepto para motivar un intercambio de ideas, deberías entonces usar lápices y papel borrador (o bolígrafos y una pizarra) en lugar de sofisticadas herramientas de dibujo por computadora.

3.2. Siete más o menos dos

Mientras el modelo gráfico de conocimiento es incorrecto pero útil, otro modelo simple tiene bases fisiológicas profundas. Como una aproximación rápida, la memoria humana se puede dividir en dos capas distintas. La primera, llamada **long-term** o **memoria persistente**, es donde almacenamos cosas como los nombres de nuestra gente amiga, nuestra dirección, y lo que hizo un payaso en nuestra fiesta de cumpleaños de ocho que nos asustó mucho. La capacidad de esta capa de memoria es esencialmente ilimitada, pero es de acceso lento—demasiado lenta para ayudarnos a lidiar con leones hambrientos y familiares descontentos.

La evolución entonces nos ha dado un segundo sistema llamado **short-term** o **memoria de trabajo**. Es mucho más rápida, pero también más pequeña: [Mill1956] estimó que la memoria de trabajo del adulto promedio solo podía contener 7 ± 2 elementos a la vez. Esta es la razón por la cual los números de teléfono[4] son de 7 u 8 dígitos de longitud: antes los teléfonos tenían dial en vez de teclado y esa era la cadena de números más larga que la mayoría de los adultos podía recordar con precisión durante el tiempo que tardaba el dial en girar varias veces.

Participación

El tamaño de la memoria de trabajo a veces se usa para explicar por qué los equipos deportivos tienden a formarse con aproximadamente media

[3] Parafraseando a Lady Windermere, obra de Oscar Wilde, las personas a menudo no saben lo que piensan hasta que se escuchan a sí mismas decirlo

[4] https://www.quora.com/Why-did-Bell-Labs-create-phone-numbers-of-7-digits-10-digits-Is-there-a-reason-that-dashes-and-brackets-are-used

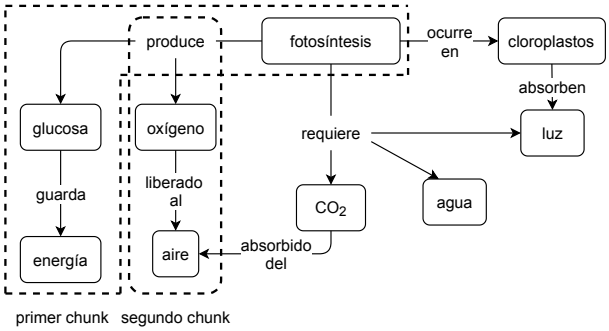

Figura 3.3: Usando mapas conceptuales en el diseño de la lección

docena de miembros o se separan en sub-grupos como delanteras/os y línea de tres cuartos en rugby. También se usa para explicar por qué las reuniones solo son productivas hasta un cierto número de participantes: si veinte personas tratan de discutir algo, o bien se arman tres reuniones al mismo tiempo o media docena de personas hablan mientras los demás escuchan. El argumento es que la habilidad de las personas para llevar registro de sus pares está limitada al tamaño de la memoria de trabajo, pero hasta donde sé, la relación jamás fue probada.

7±2 es simplemente el número más importante al enseñar. Quien enseña no puede colocar información directamente en la memoria a largo plazo de una/un estudiante. En cambio, cualquier cosa que presente se almacena primero en la memoria a corto plazo de la estudiante o del estudiante y solo se transfiere a la memoria a largo plazo después que ha sido mantenida ahí y ensayada (Section 5.1). Si quien enseña presenta demasiada información y muy rápidamente, esa nueva información desplaza la vieja antes que esta última se transfiera.

Esta es una de las razones de usar mapas conceptuales cuando se diseña una lección: sirve para asegurarse que la memoria a corto plazo de las/los estudiantes no estará sobrecargada. Una vez que se dibuja el mapa, la/el docente elegirá un fragmento que se ajuste para la memoria a corto plazo y continuará con una evaluación formativa (Figure 3.3), luego agregará otro fragmento para la próxima lección y así sucesivamente.

Construyendo juntos mapas conceptuales

La próxima vez que tengas una reunión de equipo, entrega a cada persona una hoja de papel y pídeles que pasen unos minutos dibujando sus propios

mapas conceptuales del proyecto en el que están trabajando. A la cuenta de tres, haz que todos revelen sus mapas conceptuales al grupo. La discusión que sigue puede ayudar a las personas a comprender por qué han estado tropezando.

Ten en cuenta que el modelo simple de memoria presentado aquí ha sido reemplazado en gran medida por uno más sofisticado en el que la memoria a corto plazo se divide en varios almacenamientos (p. ej. para memoria visual versus memoria lingüística), cada uno de los cuales realiza un preprocesamiento involuntario [Mill2016a]. Nuestra presentación es entonces un ejemplo de un modelo mental que ayuda al aprendizaje y al trabajo diario.

Reconocimiento de patrones

Investigaciones recientes sugieren que el tamaño real de la memoria a corto plazo podría ser tan bajo como 4±1 elementos [Dida2016]. Para manejar conjuntos de información más grandes, nuestras mentes crean **fragmentos**. Por ejemplo, la mayoría de nosotros recordamos palabras como elementos simples más que como secuencia de letras. Del mismo modo, el patrón formado por cinco puntos en cartas o dados se recuerda como un todo en lugar de cinco piezas de información separadas.

Las personas expertas tienen más fragmentos y de mayor tamaño que las no-expertas, p.ej.`ven" patrones más grandes y tienen más patrones con los que contrastar cosas. Esto les permite razonar a un nivel superior y buscar información de manera más rápida y precisa. Sin embargo, la fragmentación también puede engañarnos si identificamos mal las cosas: quienes recién llegan a veces pueden ver cosas que personas expertas han visto y pasado por alto.

Dada la importancia de la fragmentación para pensar, es tentador identificar design patterns[5] y enseñarlos directamente. Estos patrones ayudan a practicantes competentes a pensar y dialogar en varios dominios (incluida la enseñanza [Berg2012]), pero los catálogos de patrones son demasiado duros y abstractos para que personas novatas les encuentren sentido por su cuenta. Dicho esto, dar nombres a un pequeño número de patrones parece ayudar con la enseñanza, principalmente dando a las/los estudiantes un vocabulario más rico para pensar y comunicarse [Kuit2004; Byck2005; Saja2006]. Volveremos a este tema en Section 7.5.

3.3. Convirtiéndose en una persona experta

Entonces, ¿cómo se convierte alguien en una persona experta? La idea de que diez mil horas de práctica lo conseguirán es ampliamente citada pero probablemente no sea verdad[6]: hacer lo mismo una y otra vez es más probable que fortalezca los

[5] https://es.wikipedia.org/wiki/Patrón_de_diseño

[6] http://www.goodlifeproject.com/podcast/anders-ericsson/

malos hábitos a que mejore la actuación. Lo que realmente funciona es hacer cosas similares pero sutilmente diferentes, prestando atención a qué funciona y qué no, y luego cambiar el comportamiento como respuesta a las devoluciones, para mejorar de forma acumulativa. Esto se llama **deliberate** or **práctica reflectiva**, y una progresión común es que las personas pasen por tres etapas:

Actuar según la devolución de otros. Las/los estudiantes pueden escribir un ensayo sobre qué hicieron en sus vacaciones de verano y recibir devoluciones de una/un docente que les diga cómo mejorarlo.

Dar devoluciones sobre el trabajo de otros. Las/los estudiantes pueden realizar críticas de la evolución de un personaje en una novela de Harry Potter y recibir una devolución de una/un docente sobre esas críticas.

Darse devoluciones a una/o misma/o. En algún punto, las/los estudiantes empiezan a criticar sus propios trabajos como lo hacen usando las habilidades que ahora han construido. Hacer esto es mucho más rápido que esperar los comentarios de otras personas que esta aptitud de pronto empieza a despegar.

¿Qué cuenta como práctica deliberada?

[Macn2014] descubrió que, "... la práctica deliberada explicaba el 26% de la varianza en el rendimiento de los juegos, 21% para música, 18% para deportes, 4% para educación, y menos del 1% para profesiones." Sin embargo, [Eric2016] criticó este hallazgo diciendo: "Resumir cada hora de cualquier tipo de práctica durante la carrera de un individuo implica que el impacto de todos los tipos de actividad práctica respecto a rendimiento es igual ——una suposición que... es inconsistente con la evidencia." Para ser efectivo, la práctica deliberada requiere tanto un objetivo de rendimiento claro como una devolución informativa inmediata. Se trata de dos cosas que las/los docentes de cualquier manera, deberían esforzarse en conseguir.

3.4. Ejercicios

Mapear conceptos (de a pares/30)

Dibuja un mapa conceptual sobre algo que puedas enseñar en cinco minutos. Discutan con tu colega y critiquen el mapa que cada cual elaboró. ¿Presentan conceptos o detalles de superficie? ¿Cuáles de las relaciones en el mapa de tu colega consideras conceptos y viceversa?

Mapeo de conceptos (nuevamente) (grupos pequeños/20)

Trabajar en grupos de 3–4, cada persona debe dibujar independientemente del resto un mapa conceptual mostrando su modelo mental de qué sucede en un aula. Cuando todas las personas hayan terminado, comparen los mapas conceptuales. ¿Dónde coinciden y difieren sus modelos mentales?

Mejora de la memoria a corto plazo (individual/5 minutos)

[Cher2007] sugiere que la razón principal por la que las personas dibujan diagramas cuando discuten cosas es para ampliar su memoria a corto plazo: señalar una burbuja dibujada hace unos minutos provoca el recuerdo de varios minutos de debate. Cuando intercambiaste mapas conceptuales en el ejercicio anterior, ¿qué tan fácil fue para otras personas entender lo que significaba tu mapa? ¿qué tan fácil sería para ti si lo dejas de lado por un día o dos y luego lo miras de nuevo?

Eso es un poco autorreferencial, ¿no? (toda la clase/30)

Trabajando independientemente, dibuja un mapa conceptual para mapas conceptuales. Compara tu mapa conceptual con aquellos dibujados por las demás personas. ¿Qué incluyeron la mayoría de las personas? ¿Cuáles fueron las diferencias más significativas?

Notar tus puntos ciegos (grupos pequeños/10)

Elizabeth Wickes listó todo aquello que necesitas para entender[7] para leer esta línea de Python:

```
answers = ['tuatara', 'tuataras', 'bus', "lick"]
```

- Los corchetes rodeando el contenido significan que estamos trabajando con una lista (lo opuesto a corchetes inmediatamente a la derecha de algo, que es la notación utilizada para una extracción de datos).
- Los elementos se separan por comas fuera y entre comillas (en vez de adentro, como sería para un texto citado).
- Cada elemento es una cadena de caracteres, y lo sabemos por las comillas. Aquí podríamos tener números u otro tipo de datos si quisiéramos; necesitamos comillas porque estamos trabajando con cadenas.
- Estamos mezclando el uso de comillas simples y dobles; A Python no le importa eso siempre que estén balanceadas alrededor de las cadenas individuales (para cada comilla que abre haya una que cierre).

[7] https://twitter.com/elliewix/status/981285432922202113

- A cada coma le sigue un espacio, que no es obligatorio para Python, pero que preferimos para una lectura más clara.

Cada uno de estos detalles no sería ni percibido por una persona experta. Trabajando en grupos de 3–4 personas, Selecciona algo igualmente corto de una lección que hayas enseñado o aprendido y divídelo a este nivel de detalle.

Qué enseñar a continuación (individual/5)

Vuelve al mapa conceptual para la fotosíntesis en Figure 3.3. ¿Cuántos conceptos y relaciones hay en los fragmentos seleccionados? ¿Qué incluirías en el próximo fragmento de la lección y por qué?

El poder de fragmentación (individual/5)

Mira Figure 3.4 por 10 segundos, luego mira hacia otro lado e intenta escribir tu número de teléfono con estos símbolos[8]. (Usa un espacio para '0'.) Cuando hayas terminado, mira la representación alternativa en Appendix H. ¿Cuánto más fáciles de recordar son los símbolos cuando el patrón se hace explícito?

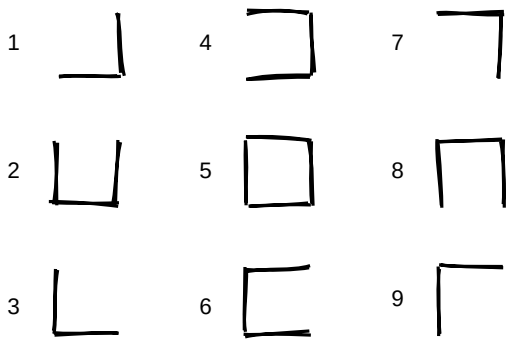

Figura 3.4: representación no fragmentada

[8] Mi agradecimiento a Warren Code por presentarme este ejemplo.

4

Arquitectura cognitiva

Hemos hablado acerca de modelos mentales como si fueran cosas reales, pero ¿qué es lo que realmente sucede en el cerebro de un aprendiz cuando está aprendiendo? La respuesta corta es que no lo sabemos, la respuesta larga es que sabemos mucho más que antes. Este capítulo profundizará en lo que el cerebro hace mientras el aprendizaje sucede y cómo podemos aprovechar eso para diseñar y brindar lecciones de manera más efectiva.

4.1. ¿Qué es lo que sucede allí?

La figura Figure 4.1 es un modelo simplificado de la arquitectura cognitiva humana. El núcleo de este modelo es la separación entre la memoria a corto y a largo plazo vistas en Section 3.2. La memoria a largo plazo es como tu sótano: almacena objetos de forma más o menos permanente pero tu conciencia no puede acceder a ella directamente. En cambio, confías en tu memoria a corto plazo, que es como el escritorio de tu mente.

Cuando necesitas algo, tu cerebro lo rescata de la memoria a largo plazo y lo coloca en la memoria a corto plazo. Por el contrario, la nueva información que llega a la memoria a corto plazo debe codificarse para poder ser almacenada en la memoria a largo plazo. Si esa información no está codificada y almacenada, no se recuerda y esto significa que no se ha aprendido.

La información ingresa a la memoria a corto plazo principalmente a través de tu

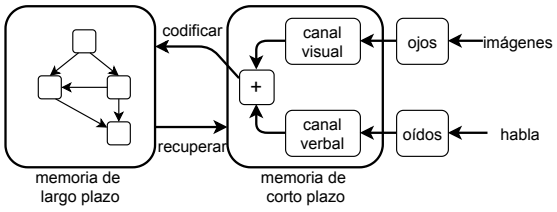

Figura 4.1: Arquitectura Cognitiva

canal verbal (para el habla) y del canal visual (para las imágenes)[1]. La mayoría de las personas confía principalmente en su canal visual, pero cuando las imágenes y las palabras se complementan entre sí, el cerebro hace un mejor trabajo al recordarlas a ambas: se codifican juntas, de modo que el recuerdo de una más tarde ayude a activar el recuerdo de la otra.

Las entradas lingüísticas y visuales son procesadas por diferentes partes del cerebro humano, y a su vez los recuerdos lingüísticos y visuales son almacenados también de manera separada. Esto significa que correlacionar flujos de información lingüísticos y visuales requiere esfuerzo cognitivo: si alguien lee algo mientras lo escucha en voz alta, su cerebro no puede evitar comprobar que obtiene la misma información por ambos canales.

Por lo tanto, el aprendizaje aumenta cuando la información se presenta de manera simultánea por dos canales diferentes, pero se reduce cuando esa información es redundante, en lugar de ser complementaria, tal fenómeno es conocido como **efecto de atención dividida** [Maye2003]. Por ejemplo, en general las personas encuentran difícil aprender de un video que tiene narración y al mismo tiempo capturas de pantalla más que de uno que solo tiene narración o capturas pero no ambos elementos, porque parte de su atención ha sido utilizada para chequear que la narración y las capturas se correspondan entre sí. Dos notables excepciones a esto, son las personas que aún no hablan bien un idioma y las que tienen algún impedimento auditivo u otras necesidades especiales, quienes quizás encuentren que el valor de la información redundante supera el esfuerzo de procesamiento adicional.

Pieza por pieza

El efecto de la atención dividida explica porque es más efectivo dibujar un diagrama pieza por pieza mientras enseñas que presentar todo el gráfico de una sola vez. Si las partes de un diagrama aparecen al mismo tiempo en que los gráficos son explicados, ambos elementos serán correlacionados en la memoria del aprendiz. Enfocarnos luego solo en una parte del diagrama es lo más parecido a activar la recuperación de lo que fue dicho cuando esa parte fue dibujada.

El efecto de la atención dividida *no* significa que los estudiantes no deberían intentar conciliar múltiples flujos de información entrantes, después de todo, esto es lo que ellos tienen que hacer en el mundo real [Atki2000]. En cambio, significa que la instrucción no debería solicitar a las personas que lo hagan mientras ellos son los primeros en manejar las habilidades de la unidad, en lugar de usar múltiples fuentes de información de manera simultánea debe tratarse como una tarea de aprendizaje separada.

No todos los gráficos son creados iguales

[Sung2012] presenta un elegante estudio que distingue los gráficos seductores *(los cuales son altamente interesantes pero no son directamente relevantes al objetivo de la enseñanza), los gráficos*decorativos *(los cuales son*

[1] Un modelo más completo también incluiría el sentido del tacto, del olfato y del gusto, pero por ahora, los ignoraremos.

neutros pero no son directamente relevantes al objetivo de la enseñanza), y por último los gráficos instructivos *(los cuales si son directamente relevantes al objetivo de la enseñanza). Los estudiantes que recibieron cualquier tipo de gráfico obtuvieron calificaciones de satisfacción del material más altas que aquellos que no obtuvieron gráficos, pero en realidad solo los estudiantes que obtuvieron gráficos instructivos obtuvieron mejores resultados.*

Del mismo modo, [Stam2013; Stam2014] descubrió que tener más información, en realidad puede disminuir el rendimiento. Les mostraron a los niños dibujos, dibujos y números, y simplemente números para dos tareas. Para algunos, tener imágenes o imágenes y números superó al tener solo números, pero para otros, tener imágenes superó a las imágenes y números, lo que superó solo tener números.

4.2. Carga cognitiva

En [Kirs2006], Kirschner, Sweller y Clark escribieron:

> Aunque los enfoques educativos no guiados o mínimamente guiados son muy populares e intuitivamente atractivos... estos enfoques ignoran tanto las estructuras que constituyen la arquitectura cognitiva humana como la evidencia de estudios empíricos de los últimos cincuenta años que indican sistemáticamente que la instrucción guiada mínimamente es menos eficaz y menos eficiente que los enfoques educacionales que hacen un fuerte énfasis en la orientación del proceso de aprendizaje del estudiante. La ventaja de la orientación disminuye sólo cuando los estudiantes tienen un conocimiento previo suficientemente elevado para proporcionar una orientación "interna".

Debajo de la jerga, los autores afirmaban que el hecho de que los estudiante hagan sus propias preguntas, establezcan sus propias metas y encuentren su propio camino a través de un tema es menos efectivo que mostrarles cómo hacer las cosas paso a paso. El enfoque "elige tu propia aventura" se conoce como **aprendizaje basado en la indagación** y es intuitivamente atractivo: después de todo, ¿quién se *opondría* a tener estudiantes que utilicen su propia iniciativa para resolver problemas del mundo real de forma realista? Sin embargo, pedir a los estudiante que lo hagan en un nuevo dominio les sobrecarga al exigirles que dominen el contenido fáctico de un dominio y sus estrategias de resolución de problemas al mismo tiempo. Más específicamente, **la teoría de la carga cognitiva** propone que la gente tiene que lidiar con tres cosas cuando está aprendiendo:

Carga Intrínseca es lo que la gente tiene que tener en cuenta para aprender el material nuevo.

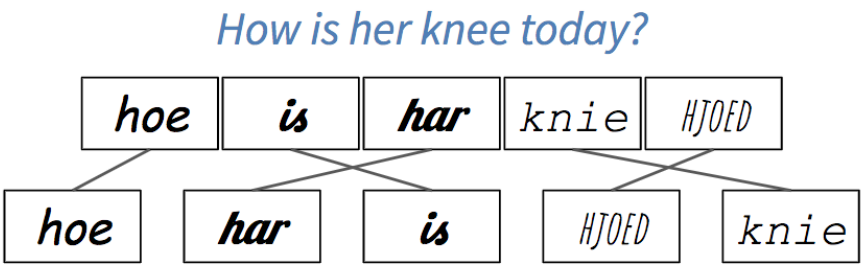

Figura 4.2: Construyendo una oración

Carga Pertinente es el esfuerzo mental (deseable) requerido para vincular la nueva información con la antigua, que es una de las cosas que distinguen el aprendizaje de la memorización.

Carga Extrínseca es cualquier cosa que distraiga del aprendizaje.

La teoría de la carga cognitiva sostiene que la gente tiene que dividir una cantidad fija de memoria de trabajo entre estas tres cosas. Nuestro objetivo como profesores es maximizar la memoria disponible para manejar la carga intrínseca, lo cual significa reducir la carga pertinente en cada paso y eliminar la carga extrínseca.

Problemas de Parsons

Un tipo de ejercicio que puede ser explicado en términos de carga cognitiva se utiliza a menudo en la enseñanza de idiomas. Supongamos que le pides a alguien que traduzca la frase, "¿Cómo está tu rodilla hoy?" a lengua frisona(frisón). Para resolver el problema, necesitan recordar tanto el vocabulario como la gramática, que es una carga cognitiva doble. Si les pides que pongan "hoe", "har", "is", "hjoed" y "knie" en el orden correcto, por otro lado, les permites que se centren únicamente en el aprendizaje de la gramática. Si escribes estas palabras en cinco fuentes o colores diferentes, sin embargo, has aumentado la carga cognitiva externa, porque involuntariamente (y posiblemente de manera inconsciente) invertirán algo de esfuerzo tratando de averiguar si las diferencias son significativas (Figure 4.2).

El equivalente de codificación de este se llama **Problema de Parsons**[2] [Pars2006].

Cuando se enseña a la gente a programar, puedes darles las líneas de código que necesitan para resolver un problema y pedirles que las pongan en el orden correcto. Esto les permite concentrarse en el flujo de control y las dependencias de datos sin distraerse con la denominación de las variables o tratando de recordar qué funciones llamar. Múltiples estudios han demostrado que los problemas de Parsons les toma a los estudiantes menos tiempo resolverlo pero producen resultados educativos equivalentes [Eric2017].

[2]Nombrado debido a uno de sus creadores.

Ejemplos desvanecidos

Otro tipo de ejercicio que se puede explicar en términos de carga cognitiva es dar a los estudiantes una serie de ejemplos desvanecidos **faded examples**. El primer ejemplo de una serie presenta un uso completo de una estrategia particular de resolución de problemas. El siguiente problema es del mismo tipo, pero tiene algunas lagunas que el estudiante debe llenar. Cada problema sucesivo le da al estudiante menos **scaffolding**, hasta que se le pide que resuelva un problema completo desde cero. Al enseñar álgebra en la escuela secundaria, por ejemplo, podríamos comenzar con esto:

$$\begin{aligned}(4x + 8)/2 &= 5\\ 4x + 8 &= 2 * 5\\ 4x + 8 &= 10\\ 4x &= 10 - 8\\ 4x &= 2\\ x &= 2 / 4\\ x &= 1 / 2\end{aligned}$$

y luego pide a los estudiantes que resuelvan esto:

$$\begin{aligned}(3x - 1)*3 &= 12\\ 3x - 1 &= \_ / \_\\ 3x - 1 &= 4\\ 3x &= \_\\ x &= \_ / 3\\ x &= \_\end{aligned}$$

y esto:

$$\begin{aligned}(5x + 1)*3 &= 4\\ 5x + 1 &= \_\\ 5x &= \_\\ x &= \_\end{aligned}$$

y finalmente, esto:

$$\begin{aligned}(2x + 8)/4 &= 1\\ x &= \_\end{aligned}$$

Un ejercicio similar para enseñar Python podría comenzar mostrando a los estudiantes cómo encontrar la longitud total de una lista de palabras:

```
# total_length(["red", "green", "blue"]) => 12
define total_length(list_of_words):
    total = 0
    for word in list_of_words:
        total = total + length(word)
    return total
```

y luego pídales que llenen los espacios en blanco en esto (lo que centra su atención en las estructuras de control):

```
# word_lengths(["red", "green", "blue"]) => [3, 5, 4]
define word_lengths(list_of_words):
    list_of_lengths = []
    for ____ in ____:
        append(list_of_lengths, ____)
    return list_of_lengths
```

El siguiente problema podría ser este (que centra su atención en actualizar el resultado final):

```
# join_all(["red", "green", "blue"]) => "redgreenblue"
define join_all(list_of_words):
    joined_words = ____
    for ____ in ____:
        ____
    return joined_words
```

Finalmente, se pedirá a los estudiantes que escriban una función completa por su cuenta:

```
# make_acronym(["red", "green", "blue"]) => "RGB"
define make_acronym(list_of_words):
    ____
```

Los ejemplos difusos funcionan porque presentan la estrategia de resolución de problemas pieza por pieza: en cada paso, los estudiantes tienen un nuevo problema que abordar, que es menos intimidante que una pantalla en blanco o una hoja de papel en blanco (Section 9.11). También anima a los estudiantes a pensar en las similitudes y diferencias entre varios enfoques, lo que ayuda a crear los vínculos en sus modelos mentales que ayudan a la recuperación de la información.

La clave para construir un buen ejemplo desvanecido es pensar en la estrategia de resolución de problemas que se pretende enseñar. Por ejemplo, los problemas de programación sobre todo utilizan el patrón de diseño del acumulador, en el que los resultados del procesamiento de elementos de una colección se agregan repetidamente a una sola variable de alguna manera para crear el resultado final.

Aprendizaje cognitivo

*Un modelo alternativo de aprendizaje e instrucción que también usa andamiaje y desvanecimiento es el **aprendizaje cognitivo**, que enfatiza la forma en que un maestro transmite habilidades y conocimientos a un aprendiz. El maestro proporciona modelos de desempeño y resultados, luego entrena a los principiantes explicando qué están haciendo y por qué [Coll1991; Casp2007]. El aprendiz reflexiona sobre su propia resolución de problemas, por ejemplo, pensando en voz alta o criticando su propio trabajo, y finalmente explora problemas de su propia elección.*

Este modelo nos dice que los profesores deben presentar varios ejemplos al explicar una nueva idea para que los estudiantes puedan ver qué generalizar, y que deben variar la forma del problema para dejar en claro cuáles son y cuáles no son características superficiales features[3]*. Los problemas deben presentarse en contextos del mundo real, y debemos fomentar la autoexplicación para ayudar a los estudiantes a organizarse y dar sentido a lo que se les acaba de enseñar (Section 5.1).*

Subobjetivos etiquetados

Labeling subgoals significa dar nombre a los pasos en una descripción paso a paso de un proceso de resolución de problemas. [Marg2016; Morr2016] descubrieron que los estudiantes con subobjetivos etiquetados resolvían los problemas de Parsons mejor que los estudiantes sin estos, y se observa el mismo beneficio en otros dominios [Marg2012]. Volviendo al ejemplo de Python usado anteriormente, los objetivos secundarios para encontrar la longitud total de una lista de palabras o construir un acrónimo son:

1. Crea un valor vacío del tipo que se devolverá.
2. Obtiene el valor que se agregará al resultado de la variable de ciclo.
3. Actualiza el resultado con ese valor.

Etiquetar subobjetivos funciona porque agrupar los pasos relacionados en fragmentos con nombre(Section 3.2) ayuda a los estudiantes a distinguir lo que es genérico de lo que es específico del problema en cuestión. También les ayuda a construir un modelo mental de ese tipo de problema para que puedan resolver otros problemas de ese tipo y les da una oportunidad natural para la autoexplicación (Section 5.1).

Manuales mínimos

La aplicación más pura de la teoría de la carga cognitiva puede ser el manual mínimo de John Carroll **minimal manual** [Carr1987; Carr2014]. Su punto de partida es una cita de un usuario: "Quiero hacer algo, no aprender a hacer todo". Carroll y sus colegas rediseñaron la capacitación para presentar cada idea como una tarea autónoma de una sola página: un título que describa de qué trata la página, instrucciones paso a paso sobre cómo hacer una sola cosa (por ejemplo, cómo eliminar una línea en blanco en un editor de texto) y luego varias notas sobre cómo reconocer y resolver problemas comunes. Descubrieron que reescribir los materiales de capacitación de esta manera los hacía más cortos en general y que las personas que los usaban aprendían más rápido. Estudios posteriores confirmaron que este enfoque superó al enfoque tradicional independientemente de la experiencia previa con computadoras [Lazo1993]. [Carr2014] resumieron este trabajo diciendo:

[3] Por mucho tiempo, creí que la variable que contenía el valor que una función iba a devolver *tenía* que llamarse `resultado` porque mi maestro siempre usaba ese nombre en los ejemplos.

> Nuestros diseños "minimalistas" buscaban aprovechar la iniciativa del usuario y el conocimiento previo, en lugar de controlarlo mediante advertencias y pasos ordenados. Enfatizó que los usuarios generalmente aportan mucha experiencia y conocimiento a este aprendizaje, por ejemplo, conocimiento sobre el dominio de la tarea, y que dicho conocimiento podría ser un recurso para los diseñadores instruccionales. El minimalismo aprovechó los episodios de reconocimiento, diagnóstico y corrección de errores, en lugar de intentar simplemente prevenirlos. Enmarca la resolución de problemas y la corrección como oportunidades de aprendizaje en lugar de aberraciones.

4.3. Otros modelos de aprendizaje

Los críticos de la teoría de la carga cognitiva a veces han argumentado que cualquier resultado puede justificarse a posteriori al etiquetar las cosas que perjudican el rendimiento como cargas extrañas y las que no lo hacen como intrínsecas o pertinentes. Sin embargo, la instrucción basada en la teoría de la carga cognitiva es innegablemente efectiva. Por ejemplo, [Maso2016] rediseñó un curso de base de datos para eliminar la atención dividida y los efectos de redundancia y para proporcionar ejemplos prácticos y subobjetivos. El nuevo curso redujo la tasa de reprobación del examen en un 34 % y aumentó la satisfacción del estudiante.

Una década después de la publicación de [Kirs2006], un número creciente de personas cree que la teoría de la carga cognitiva y los enfoques basados en la investigación son compatibles si se ven de la manera correcta. [Kaly2015] sostiene que la teoría de la carga cognitiva es básicamente una microgestión del aprendizaje dentro de un contexto más amplio que considera cosas como la motivación, mientras que [Kirs2018] extiende la teoría de la carga cognitiva para incluir aspectos colaborativos del aprendizaje. Al igual que con [Mark2018] (discutido en Section 5.1), las perspectivas de los investigadores pueden diferir, pero la implementación práctica de sus teorías a menudo termina siendo la misma.

Uno de los desafíos en la investigación educativa es que lo que queremos decir con "aprendizaje" resulta complicado una vez que se mira más allá del aula occidental estandarizada. Dos perspectivas específicas de la **psicología educativa** han influido en este libro. El que hemos utilizado hasta ahora es el **cognitivismo**, que se centra en cosas como el reconocimiento de patrones, la formación de la memoria y el recuerdo. Es bueno para responder preguntas de bajo nivel, pero generalmente ignora cuestiones más importantes como, "¿Qué queremos decir con 'aprendizaje'?" y "¿Quién decide?" El otro es el **aprendizaje situado**, que se centra en llevar a las personas a una comunidad y reconoce que la enseñanza y el aprendizaje siempre están arraigados en quiénes somos y quiénes aspiramos a ser. Lo discutiremos con más detalle en el Chapter 13.

El sitio web de Learning Theories, teorías de aprendizaje en inglés[4] y [Wibu2016] tienen buenos resúmenes de estas y otras perspectivas. Además del cognitivismo, los que se encuentran con mayor frecuencia incluyen el **conductismo** (que trata la educación como un condicionamiento de estímulo/respuesta), el **constructivismo** (que considera el aprendizaje como un proceso activo durante el cual los estudiantes construyen conocimiento por sí mismos) y el **conectivismo**. (que sostiene que el conocimiento se distribuye, que el aprendizaje es el proceso de navegar, crecer y podar conexiones, y que enfatiza los aspectos sociales del aprendizaje que Internet hace posible). Estas perspectivas pueden ayudarnos a organizar nuestros pensamientos, pero en la práctica, siempre tenemos que probar nuevos métodos en la clase, con estudiantes reales, para descubrir qué tan bien equilibran las muchas fuerzas en juego.

4.4. Ejercicios

Crear un ejemplo desvanecido (pares/30')

Es muy común que los programas cuenten cuántas cosas caen en diferentes categorías: por ejemplo, cuántas veces aparecen colores diferentes en una imagen o cuántas veces aparecen palabras diferentes en un párrafo de texto.

1. Crea un ejemplo breve (no más de 10 líneas de código) que muestre a las personas cómo hacer esto, y luego crea un segundo ejemplo que resuelva un problema similar de una manera similar pero que tenga un par de espacios en blanco para que los estudiante los completen. ¿Decides qué desvanecer? ¿Cuál sería el siguiente ejemplo de la serie?

2. Define la audiencia de sus ejemplos. Por ejemplo, ¿son estos principiantes que solo conocen algunos conceptos básicos de programación? ¿O estos estudiantes tienen alguna experiencia en programación?

3. Muestra tu ejemplo a un compañero, pero no le digas para qué nivel crees que es. Una vez que hayan llenado los espacios en blanco, pídeles que adivinen el nivel deseado.

Si hay personas entre los aprendices que no programan en absoluto, intenta ubicarlos en diferentes grupos y pídeles que hagan el papel de aprendices para esos grupos. Alternativamente, elija un dominio de problema diferente y desarrolle un ejemplo difuso para él.

[4] http://www.learning-theories.com/

Clasificación de carga (grupos pequeños/15')

1. Elige una lección corta que un miembro de tu grupo haya enseñado o tomado recientemente.
2. Haz una lista en forma de puntos de las ideas, instrucciones y explicaciones que contiene.
3. Clasifica cada uno como intrínseco, pertinente o extraño. ¿En qué partes estaban todos de acuerdo? ¿Dónde estuvo en desacuerdo y por qué?

(El ejercicio "Cómo darse cuenta de su punto ciego" en Section 3.4 le dará una idea de cuán detallada debe ser su lista de formularios de puntos).

Crear un problema de Parsons (en parejas/20')

Escribe cinco o seis líneas de código que hagan algo útil, mezclalas y pídele a tu compañero que las ponga en orden. Si estás utilizando un lenguaje basado en identación como Python, no utilices sangría en ninguna de las líneas; si estás utilizando un lenguaje de llaves como Java, no incluyas ninguna de las llaves. (Si tu grupo incluye personas que no son programadores, usa un dominio de problema diferente, como hacer budín/pan de plátano).

Manuales mínimos (individual/20')

Escribe una guía de una página para hacer algo que tus estudiantes puedan encontrar en una de tus clases, como centrar el texto horizontalmente o imprimir un número con un cierto número de dígitos después del punto decimal. Intentá enumerar al menos tres o cuatro resultados incorrectos que el estudiante pueda ver e incluí una explicación de una o dos líneas de por qué ocurre cada uno y cómo corregirlo.

Aprendizaje cognitivo (parejas/15')

Elige un problema de codificación que puedas resolver en dos o tres minutos y piensa en voz alta mientras lo resuelves, al mismo tiempo tu compañero te hace preguntas sobre lo que estás haciendo y por qué. No solo explica lo que estás haciendo, sino también por qué lo estás haciendo, cómo sabes que es lo correcto y qué alternativas has considerado pero descartado. Cuando hayas terminado, intercambia roles con tu compañero y repetí el ejercicio.

Ejemplos resueltos (parejas/15')

Ver ejemplos resueltos ayuda a las personas a aprender a programar más rápido que simplemente escribiendo mucho código [Skud2014], y deconstruir/desagregar el código rastreándolo o depurándolo también aumenta el aprendizaje [Grif2016]. Trabajando en parejas, revisa un fragmento de código de 10 a 15 líneas y explica qué

hace cada declaración y por qué es necesaria. ¿Cuánto tiempo te tardas? ¿Cuántas cosas crees que necesitas explicar por línea de código?

Gráficos críticos (individual/30')

[Maye2009; Mill2016a] presentan seis principios para una buena enseñanza de gráficos:

Señalización: resalta visualmente los puntos más importantes para que se destaquen del material menos crítico.

Contigüidad espacial: coloca los subtítulos lo más cerca posible de los gráficos para compensar el costo de cambiar entre los dos.

Contigüidad temporal: Presenta narraciones y gráficos hablados tan seguidos en el tiempo como sea práctico. (Presentar ambos a la vez es mejor que presentarlos uno tras otro).

Segmentación: Cuando presentes una secuencia larga de material o cuando los estudiantes no tengan experiencia con el tema, divide la presentación en segmentos cortos y deja que los estudiantes controlen la rapidez con que avanzan al siguiente.

Pre-entrenamiento: Si los estudiantes no conocen los conceptos y la terminología principales utilizados en tu presentación, enseña solo esos conceptos y términos de antemano.

Modalidad: las personas aprenden mejor de las imágenes más la narración que de las imágenes más el texto, a menos que no sean hablantes nativos o haya palabras o símbolos técnicos.

Elige un video de una lección o charla en línea que utilice diapositivas u otras presentaciones estáticas y califique sus gráficos como "deficientes", "promedio" o "buenos" de acuerdo con estos seis criterios.

Revisión

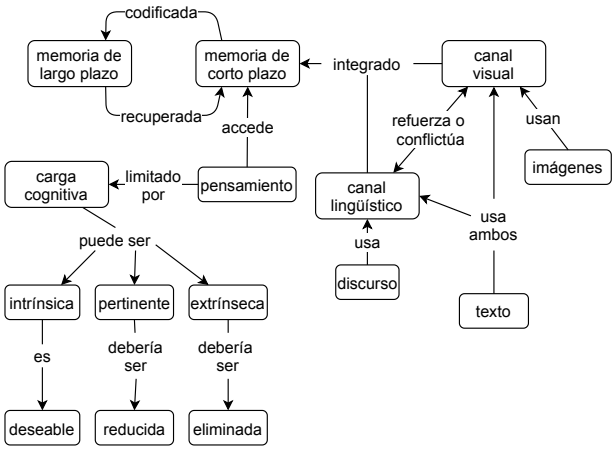

Figura 4.3: Concepto: Carga congnitiva

5

Aprendizaje individual

Los capítulos previos han explorado como las/los docentes pueden ayudar a sus estudiantes. Este capítulo se enfoca en cómo las/los estudiantes pueden ayudarse a si mismos al cambiar sus estrategias de estudio y descansando lo necesario.

La estrategia más efectiva es hacer el cambio de **aprendizaje pasivo** a **aprendizaje activo** [Hpl2018], ya que mejora la taza de rendimiento y reduce la taza de fracaso [Free2014]:

Pasivo	**Activo**
Leer sobre un tema	Hacer ejercicios
Mirar un video	Discutir un tema
Asistir a una clase	Tratar de explicar un tema

Haciendo referencia a nuestro modelo simplificado de la arquitectura cognitiva (Figure 4.1), el aprendizaje activo es más efectivo porque retiene la información nueva en la memoria a corto plazo por más tiempo, lo cual aumenta la chance que sea codificada con éxito y almacenada en la memoria a largo plazo. Y al usar la nueva información a medida que llega, tus estudiantes pueden construir y fortalecer los lazos entre la información nueva y la información que ya poseen, lo cual incrementa a la vez, las chances de que puedan recuperarla más tarde.

La otra clave para poder sacarle más provecho al aprendizaje es la **metacognición**, o en otras palabras, pensar sobre lo que una/uno está pensando. Así como las/los musicos escuchan lo que están tocando, y las/los buenas docentes reflexionan sobre su enseñanza (Chapter 8), tus estudiantes aprenderán mejor y más rápido si hacen planes, fijan metas y monitorean su progreso. Para tus estudiantes es difícil dominar estas habilidades en el abstracto—solo diciéndoles que planeen no tiene ningún efecto—pero las lecciones pueden diseñarse para motivar buenas prácticas de estudio, y al hacer referencia a estas prácticas en la clase ayudas a que tus estudiantes se den cuenta que aprender es una habilidad que pueden mejorar como cualquier otra [McGu2015; Miya2018].

El gran premio es la **transferencia del aprendizaje**, que ocurre cuando algo que hemos aprendido nos ayuda a aprender algo nuevo más rápido. Investigadores distinguen entre **transferencia cercana**, que ocurre entre áreas similares o relacionadas como las fracciones y decimales en matemáticas, y **transferencia lejana**, que ocurre entre dominios diferentes—por ejemplo, la idea de que aprender ajedrez ayudará al razonamiento matemático y viceversa.

La transferencia cercana ocurre indudablemente—ningún tipo de aprendizaje

más allá de la simple memorización podría ocurrir si no fuera así—y las/los docentes lo utilizan todo el tiempo al dar a sus estudiantes ejercicios similares al material que acaba de ser presentado en la lección. Sin embargo, [Sala2017] analizó varios estudios sobre la transferencia lejana y concluye que:

> ... los resultados muestran un efecto pequeño a moderado. Sin embargo, el tamaño de efecto está inversamente relacionado a la calidad del diseño experimental.... Concluimos que la transferencia lejana raramente sucede.

Cuando la transferencia lejana sucede, parece que sucede solamente cuando el tema ha sido dominado [Gick1987]. En la práctica, esto significa que aprender a programar no ayudará a jugar ajedrez y vice versa.

5.1. Seis estrategias

Las/los psicólogos estudian el aprendizaje en una amplia variedad de formas, pero han llegado a conclusiones similares sobre lo que realmente funciona [Mark2018]. Los Learning Scientists[1] han catalogado seis de estas estrategias y las resumieron en un set de afiches para descargar[2]. Si enseñas estas estrategias a tus estudiantes, y las mencionas por su nombre cuando las utilices en la clase, las/los ayudas a aprender cómo aprender más rápido y mejor [Wein2018a; Wein2018b].

Práctica Distribuida

Diez horas de estudio repartidas en cinco días es más efectivo que dos días de cinco horas, y mucho mejor que un día de diez horas. Por lo tanto, deberías crear un horario de estudio donde distribuyas las actividades de estudio a lo largo del tiempo: reserva al menos media hora para estudiar cada tema, cada día en vez de amontonar todo para la noche antes del examen [Kang2016].

También deberías revisar los materiales después de cada clase, pero no inmediatamente después—toma al menos media hora de receso. Cuando repases, asegurate de incluir al menos un poquito del material más antiguo: por ejemplo, utiliza veinte minutos para revisar las notas de la clase de hoy y luego cinco minutos para revisar material de los días anteriores y de la semana pasada. Esto te ayudará a identificar algun vacio o errores en tus apuntes previos cuando todavía haya tiempo para corregirlos o hacer preguntas: es doloroso darse cuenta la noche del examen que no tienes idea por qué subrayaste “Demodular!!” tres veces.

Al repasar, haz notas sobre las cosas que te hayas olvidado: por ejemplo, haz una tarjeta de memoria para cada concepto que no pudiste recordar o que recordaste incorrectamente [Matt2019]. Esto te ayudará a enfocarte en las cosas que necesitan más atención cuando vuelvas a estudiar.

[1] http://www.learningscientists.org/
[2] http://www.learningscientists.org/downloadable-materials

El Valor de las Clases Magistrales

Según [Mill2016a], "Las clases magistrales que predominan en los cursos presenciales son relativamente formas | ineficientes de enseñar, pero probablemente contribuyen a distribuir el material en el tiempo, porque se desenvuelven en un cronograma establecido en el tiempo. Al contrario, dependiendo de cómo se organicen los cursos, las/los estudiantes en línea a veces pueden evitar exponerse al material por completo hasta que una tarea esté cerca."

Recordar lo Aprendido, del inglés Retrieval Practice

El factor limitante de la memoria a largo plazo no es retener (lo que se almacena) pero recordar (lo que puede accederse). Recordar una información específica mejora con la práctica, asi que los resultados en situaciones reales pueden mejorar al hacer examenes de práctica o resumiendo en detalle un tema de memoria y luego revisando que fue y que no fue recordado. Por ejemplo, [Karp2008] encontró que hacer exámenes de forma repetida mejora el recuerdo de listas de palabras, de un 35 % al 80 %.

La habilidad de recordar mejora cuando en la práctica se utilizan actividades similares a las que se evalúan. Por ejemplo, escribir entradas en un diario personal ayuda con los examenes de opción múltiple, pero menos que hacer examenes de práctica [Mill2016a]. Este fenómeno se llama **transferencia apropiada de procesamiento**.

Una manera de ejercitar las habilidades de recuerdo es resolver problemas dos veces. La primera vez, hacerlo completamente de memoria, sin notas o discusiones con pares. Después de evaluar tu propio trabajo con una rúbrica de respuestas distribuidas por la/el docente, resuelve el problema de nuevo, utilizando cualquier material de apoyo que quieras. La diferencia entre ambos te muestra que tan bien pudiste recordar y aplicar el conocimiento.

Otro método (mencionado previamente) es crear tarjetas de estudio. Las tarjetas físicas tienen una pregunta en un lado y la respuesta en el otro, y existen múchas aplicaciones para generarlas disponibles para teléfono móbil. Si estas estudiando con un grupo de estudio, intercambiar las tarjetas de estudio con tu colega te ayudará a descubrir ideas importantes que tal vez habias obviado o malinterpretado.

Leer-cubrir-recordar es una alternativa rápida a las tarjetas de estudio. Mientras lees algo, cubre los términos clave o secciones con notas adhesivas pequeñas. Cuando hayas terminado, vuelve a leer y ve que tan bien puedes adivinar las palabras cubiertas por las notas adhesivas. Independientemente del método que uses, no sólo practiques recordar datos y definiciones: asegurate de evaluar la comprensión de ideas grandes y de las conexiones entre ellas. Diagramar un mapa conceptual y compararlo con tus apuntes o con un mapa conceptual dibujado previamente es una forma rápida de hacer esto.

Hipercorrección

*Un descubrimiento poderoso en la investigación del aprendizaje es el **fe-***

nómeno de hipercorrección *[Metc2016]. A la mayoría de la gente no le gusta que le digan cuando dicen algo incorrecto, asi que sería razonable asumir que mientras mas confianza tenga una persona en la respuesta que dan en un examen, más dificíl es cambiarle de opinión si el resultado era incorrecto. Y resulta que lo opuesto es cierto: mientras mas confianza tenga una persona de que tiene la razón, más probable que no repitan el error si este es corregido.*

Práctica intercalada

Una forma de espaciar la práctica es intercalar el estudio de diferentes temas: en vez de dominar un tema, luego el segundo y el tercero, alterna las sesiones de estudio. Aún mejor, cambia el orden: A-B-C-B-A-C es mejor que A-B-C-A-B-C, que a la vez es mejor que A-A-B-B-C-C [Rohr2015]. Esto funciona porque intercalar permite la creación de más vínculos entre los diferentes temas, lo cual mejora el aprendizaje.

Cuánto deberías tardar en cada ítem depende del tema y que tan bien lo conozcas. Entre 10 y 30 minutos es un tiempo suficiente para entrar en tema(Section 5.2) pero no para deambular. Intercalar el estudio parecerá más difícil que enfocarse en un sólo tema al principio, pero ese es un signo de que está funcionando. Si usas tarjetas de estudio, o haces exámenes de práctica para medir tu progreso, deberías ver una mejora después de un par de días.

Elaboración

Explicar los temas a uno mismo mientras se estudia permite entenderlos y recordarlos. Una forma de hacer esto es complementar la respuesta en un exámen práctico con la explicación de por qué la respuesta es correcta, o al contrario, con una explicación de por qué otras respuestas plausibles no lo serían. Otra alternativa es decirse a uno mismo cómo una nueva idea es similar o diferente a una que ya hayas visto previamente.

Hablarse a una misma o a uno mismo puede parecer una forma extraña de estudiar, pero [Biel1995] encontró que las personas entrenadas en auto-explicarse descatan cuando se comparan quienes no se entrenaron. Similarmente, [Chi1989] encontró que estudiantes simplemente frenan cuando se encuentran con un paso que no entienden al tratar de resolver problemas. Otros pausan y generan una explicación de lo que está pasando, y aprenden más rápido. Un ejercicio para construir esta habilidad es revisar un ejercicio de programación línea por línea con una clase, y que diferentes personas expliquen cada línea, y digan por qué está ahí y qué es lo que produce.

Ejemplos concretos

Una forma particularmente útil de elaborar es el uso de ejemplos concretos. Cuando uno tiene una definición de un principio general, intenta proveer uno o más ejemplos de su uso, o por el contrario, toma un problema en particular y enuncia los principios generales que representa. [Raws2014] encontró que intercalar ejemplos y

definiciones de esta forma permite que las/los estudiantes puedan recordar lo último correctamente.

Una forma estructurada de hacer esto es con el método ADEPT[3]: da una **A**nalogía, dibuja un **D**iagrama, presenta un **E**jemplo, describe la idea en un lenguaje **S**ensillo (del inglés Plain Language), y luego da los detalles **T**écnicos. De nuevo, si estás estudiando con alguien o en grupo, puedes intercambiar y revisar el trabajo: ve si estás de acuerdo si los ejemplos de otras personas representan el principio que se está discutiendo, o qué principios se usan en un ejemplo que no hayan anotado.

Otra técnica útil es enseñar por contraste, p.ej. mostrar a tus estudiantes cuál *no* es la solución o cuál es técnica que *no* resolverá un problema. Por ejemplo, al mostrar a las/los niños cómo simplificar fracciones, es importante darles a menos un par de ejemplos como 5/7 que no pueden simplificarse, para que no se frustren buscando respuestas que no existen.

Programación Dual

La última de las seis estrategias principales descritas en Learning Scientists[4] es presentar palabras e imágenes juntas. Cómo discutimos en Section 4.1, diferentes subsistemas en nuestro cerebro manejan y almacenan la información linguistica y visual, de tal manera que, si se presenta informacion complementaria por ambos canales, se refuerzan mutuamente. Sin embargo, aprender es menos efectivo cuando la misma información se presenta de forma simultánea por dos canales diferentes, porque el cerebro tiene que hacer un esfuerzo para comparar los canales entre si [Maye2003].

Una forma de aprovechar la programación dual es dibujar o etiquetar lineas de tiempo, mapas o árboles familiares, o cualquier otro material que sea relevante. (Personalmente, me gustan las imágenes que muestran qué funciones llaman a qué otras en un programa.) Dibujar un diagrama *sin* etiquetas, y después volver atrás para etiquetarlo, es una práctica excelente para recordar.

5.2. Gestión del tiempo

Solía presumir sobre la cantidad de horas que trabajaba. No en muchas palabras, obviamente—tenía *algunas* habilidades sociales—pero me presentaba en clases alrededor del mediodía, sin rasurar y bostezando, y casualmente mencionaba a quien sea que pudiera escuchar que estaba trabajando desde las 6:00 a.m.

Mirando atrás, no puedo recordar a quién estaba tratando de impresionar. Pero lo que si recuerdo es, cuánto del trabajo que hice durante las trasnochadas tuve que tirar una vez que dormí un poco, y cuanto daño le hizo a mis notas lo que no tiré.

[3] https://betterexplained.com/articles/adept-method/
[4] http://www.learningscientists.org/

Mi error fue confundir "trabajar" con "ser productivo." No puedes producir software (o cualquier otra cosa) sin hacer algo de trabajo, pero se puede hacer fácilmente mucho trabajo sin producir nada de valor. Convencer a la gente de esto es difícil, especialmente cuando son adolescentes o veinteañeras y veinteañeros, pero paga tremendos dividendos.

El estudio científico en trabajo exceso y deprivación del sueño se remonta al ménos a la década de 1890s—vea [Robi2005] para un resumen breve y legible. Los resultados más importantes para estudiantes son:

1. Trabajar más de 8 horas al día por un periodo extendido de tiempo disminuye la productividad total, no sólo la productividad por hora —i.e. haces menos en total (no sólo por hora) cuando tienes trabajo acumulado y cerca de una fecha límite de entrega.

2. Trabajar durante 21 horas seguidas aumenta la chance de que tengas un error catastrófico tanto como estar legalmente en estado de ebriedad.

3. La productividad varía a lo largo de la jornada laboral, con la mayor productividad en las primeras 4 a 6 horas. Después de cierta cantidad de horas, la productividad disminuye a cero; y eventualmente se vuelve negativa.

Estos hechos se han reproducido y verificado durante más de un siglo, y los datos detrás de ellos son tan sólidos como los que relacionan el tabaquismo con el cáncer de pulmón. El problema es que *las personas generalmente no notan que sus habilidades disminuyen*. Como cuando personas en estado de ebriedad creen que todavía pueden conducir, las personas que están deprivadas de sueño no se dan cuenta que no están terminando sus oraciones (o pensamientos). Se ha demostrado que cinco días de 8 horas por semana maximizan la producción total a largo plazo en todas las industrias que se han estudiado; estudiar o programar no es diferente.

Pero que pasa con las rachas que surgen de vez en cuando, como trabajar toda la noche para cumplir con un plazo? Eso también se ha estudiado, Y los resultados no son agradables. La habilidad de pensar dismuye en un 25 % por cada 24 horas sin dormir. Puesto de otra forma, el coeficiente intelectual de una persona promedio es sólo 75 después de una trasnochada, lo que la desplaza al 5 % inferior de la población. Si haces dos trasnochadas seguidas, tu coeficiente intelectual es de 50, que es el nivel en el que las personas suelen ser consideradas incapaces de vivir de forma independiente.

"Pero—pero—tengo tantas tareas que hacer!" dices tú. "Y todas tienen que ser entregadas a la misma vez! *Tengo* que trabajar horas extra para completarlas!" No: las personas tienen que priorizar y enfocarse para ser productivas, y para hacerlo, deben ser ser enseñadas como. Una técnica ampliamente utilizada es hacer una lista de tareas que deben hacerse, organizadas por prioridad, y luego desconectarse del correo electrónico u otras interrupciones por 30–60 minutos y completar una de esas tareas. Si una de esas tareas de la lista para-hacer lleva más de una hora, sepárala en pedazos más pequeños y priorizalos de forma separada.

La parte más importante de esto es apagar las interrupciones. A pesar de lo que

mucha gente quiere creer, los seres humanos no somos buenos en hacer múltiples tareas a la vez. pero, en lo que podemos volvernos buenos **automaticity**, es en la habilidad de hacer algo de forma rutinaria de fondo mientras realizamos otra tarea [Mill2016a]. La mayoría de las personas puede hablar mientras corta una cebolla, o tomar café mientras lee; con la práctica, también podemos tomar notas mientras escuchamos, pero no podemos estudiar de forma efectiva, programar, o hacer otra tarea mentalmente exigente mientras prestamos atención a algo más—sólo creémos que podemos.

El punto de organizarse y prepararse es para entrar en el estado mental más productivo posible. Quienes estudiaron psicología lo llaman **flujo** [Csik2008]; las personas que realizan atletismo lo llaman "estar en la zona," y las/los músicos hablan de perderse en lo que están tocando. CUalquier nombre que uses, las personas producen mucho más por unidad de tiempo en este estado que en un estado normal. La mala noticia es que tarda aproximadamente 10 minutos volver a entrar en este estado después de tener una interrupción, sin importar lo corta que haya sido la interrupción. Lo que significa que si te interrumpieron seis veces por hora, *nunca* llegaste al máximo de tu productividad.

Cómo lo supo?

En su breve historia en 1961 "Harrison Bergeron[5]*," Kurt Vonnegut describió un futuro en el que las personas están obligadas a ser iguales. Las personas atractivas tienen que usar máscaras, las personas atléticas tienen que cargar pesas—y las personas inteligentes estan obligadas a llevar radios que interrumpen sus pensamientos en intervalos aleatorios. A veces me pregunto si—oh, un momento, mi teléfono acaba de—perdón, de qué estábamos hablando?*

5.3. Evaluación de pares

Perdirle a las personas de un equipo que evalúen a sus pares es una práctica común en la industria. [Sond2012] revisó la literatura en evaluación de pares, distinguiendo entre calificar y evaluar. Descubrieron que la evaluación por pares aumentaba la cantidad, la diversidad y la puntualidad de la retroalimentación, ayudó a las/los estudiantes a ejercitar el pensamiento de nivel superior, fomenta la práctica reflexiva, y apoya el desarrollo de habilidades sociales. Las preocupaciones fueron predecibles: validez y confiabilidad, motivación y procrastinación, trolls, colusión y plagio.

Sin embargo, la evidencia muestra que estas preocupaciones no fueron significantes en la mayoría de las clases. Por ejemplo, [Kauf2000] comparó las evaluaciones y calificaciones confidenciales de pares en varios ejes para dos cursos en licenciatura en ingeniería, y descubrió que la auto-calificación y la evaluación de pares tuvo una concordancia estadística, que la colusión no fue significativa (i.e. no dieron irrespectivamente la nota más alta a todos sus pares), que las/los estudiantes no inflaron sus auto-calificaciones, y crucialmente, que las calificaciones no estaban sesgadas por género o raza.

Una forma de implementar la evaluación de pares es **contribuyendo a la pedagogía estudiantil**, en la cual tus estudiantes producen artefactos para contribuir al aprendizaje de otros. Esto puede hacerse desarrollando una lección corta y compartiéndola con la clase, contribuir a un banco de preguntas, o escribir notas de una lección en particular para una publicación durante la clase. Por ejemplo, [Fran2018] evidenció que las/los estudiantes que realizan videos cortos para enseñar conceptos a sus pares tuvieron un incremento significativo de su propio aprendizaje comparado al de aquellos que sólo estudiaron el material o vieron los videos. Yo he visto que pedir a mis estudiantes que muestren un error en su cóodigo y que muestren la solución con la clase cada día, ayuda en sus habilidades analíticas y disminuye el síndrome del impostor.

Otra alternativa es la **revisión por pares calibrada**, en la cual tus estudiantes revisan uno o más ejemplos utilizando una rúbrica y comparan su evaluación con

[5] https://en.wikipedia.org/wiki/Harrison_Bergeron

la evaluación del cuerpo docente [Kulk2013]. Una vez que la evaluación de tu estudiante sea similar a la del cuerpo docente, pueden empezar a evaluar el trabajo de sus pares. Si se combinan muchas evaluaciones de pares, estas pueden ser tan precisas como la evaluación del cuerpo docente [Pare2008].

Como todo lo demás, la evaluación es ayudada por rúbricas. La planilla de evaluación Section F.2 muestra un ejemplo para que puedas empezar. Para usarla, evalúate y a tus colegas, luego calcula y compara las notas. Una diferencia grande generalmente indica que la necesidad de una major conversación.

5.4. Ejercicios

Estrategias de aprendizaje (individual/20)

1. Cuál de la seis estrategias de aprendizaje generalmente usas? Cuáles generalmente no usas?

2. Escribe tres conceptos generales que quieras que tus estudiantes aprendan y da dos ejemplos específicos para cada caso (practica ejemplos concretos). Para cada uno de estos conceptos, trabaja al revés, parte de uno de los ejemplos y elabora sobre el concepto que lo explica (elaboración).

Conectando ideas (en pares/5)

Este ejercicio es un ejemplo de utilizar la elaboración para mejorar la retención. Elige a una/un colega que cada persona independientemente elija una idea, luego anuncia tu idea y trata de encontrar una cadena de cuatro lazos que conecte ambas ideas. Por ejemplo, si dos ideas son “Saskatchewan” y “estadística,” los lazos pueden ser:

- Saskatchewan es una provincia de Canadá;
- Canada es un país;
- países tienen gobiernos;
- los gobiernos utilizan estadísticas para analizar la opinión pública.

Evolución Convergente (en pares/15)

Una práctica que no hemos cubierto anteriormente son las **notas guiadas**, que son notas preparadas por la/el docente que solicita a las/los estudiantes a responder preguntas respecto a la información clave de una lectura o discusión. Estas preguntas pueden ser espacios en blancos dónde tus estudiantes agregan información, asteriscos junto a términos que deben definir, etcétera.

Crea dos a cuatro tarjetas con notas guidas para una lección que hayas enseñado recientemente o que vas a enseñar. Intercambia tarjetas con tu colega: que tan fácil es entender lo que se está preguntando? Cuánto tiempo te lleva llenar las respuestas? Qué tan bien funciona esto para ejemplos de programación?

Cambiando de opinión (en pares/10)

[Kirs2013] argumenta que los mitos sobre nativos digitales, estilos de aprendizaje, y personas autodidactas es que todas son reflejos de creencias equivocadas que las/los estudiantes saben lo que es mejor para ellos, y advierte que podemos estar en una espiral descendente en la que todos los intentos de las/los investigadores en educacion para refutar estos mitos confirman la creencia de sus oponentes de que aprender ciencia es seudo-ciencia. Elige una cosa que hayas aprendido hasta ahora en es este libro que te haya sorprendido o contradecido a algo que creías previamente y practica explicar esto a tu colega en 1–2 minutos. Cuán convincente eres?

Tarjetas de estudio (individual/15)

Utiliza las notas adhesivas o algo similar que tengas a mano para hacer seis tarjetas de estudio para un tema que recientemente hayas enseñado o aprendido. Intercambia con tu colega y ve cuánto tiempo tardas en recordar al 100 % cada tarjeta Deja las tarjetas a un lado cuando termines, y vuelve media hora después para evaluar cuál es tu taza de retención.

Utilizando ADEPT (toda la clase/15)

Elige un tema que recién hayas enseñado o aprendido, y delínea una pequeña lección que utilice los cinco pasos del método ADEPT para introducir el tema.

El costo de la multitarea (en pares/10)

The Learning Scientists blog[6] describe un experimento simple que puedes hacer con un sólo un cronómetro, para demostrar el costo de la multitarea. Trabajando en pares, mide cuánto tarda cada persona en hacer cada una de estas tres tareas:

- Contar del 1 al 26 dos veces
- Recitar el alfabeto de la A a la Z dos veces.
- Intercalar números y letras, i.e. decir, “1, A, 2, B, . . . ” y continuar.

Que cada pareja reporte sus números. Sin práctica específica, la tercera tarea siempre toma significativamente más tiempo que cada uno de los componentes de la tarea.

[6] http://www.learningscientists.org/blog/2017/7/28-1

Mitos en la educación de computación (toda la clase/20)

[Guzd2015b] presenta una lista del top 10 de creencias erróneas sobre educación en computación, que incluye:

1. La ausencia de mujeres en Ciencias de la Computación es similar en otras áreas de STEM.
2. Para tener más mujeres en las Ciencias de Computación necesitamos más docentes mujeres en el área.
3. Las evaluaciones de estudiantes son la mejor forma de evaluar la enseñanza.
4. Las/los buenos docentes personalizan la educación a los estilos de aprendizaje de sus estudiantes.
5. Una/un buen docente en ciencias de la computación debería modelar buenas practicas de desarrollo de software porque su trabajo es producir ingenieras e ingenieros de software excelentes.
6. Algunas personas son naturalmente mejores programadoras que otras.

Invita a que cada persona vote +1 (de acuerdo), -1 (en desacuerdo), o 0 (neutro) para cada punto, luego lee la explicación completa en el artículo original[7] y vuelve a hacer la votación. En cuáles preguntas las personas cambiaron de parecer? Cuáles todavía creen que son verdad, y por qué?

Revisión por pares calibrada (pares/20)

1. Crea una rúbrica de 5–10 puntos con entradas como "buenos nombres de variables," "sin código redundante," y "flujo de control propiamente anidado" para calificar el tipo de programa que esperarías que tus estudiantes escriban.
2. Elige o crea un pequeño programa que contenga 3–4 violaciones a estas entradas.
3. Califica el programa de acuerdo a la rúbrica.
4. Pide a tu colega que califique el mismo programa con la misma rúbrica. Qué aceptó que tu no aceptaste? Qué criticaron que tu criticaste?

[7] https://cacm.acm.org/blogs/blog-cacm/189498-top-10-myths-about-teaching-computer-science/fulltext

Revisión

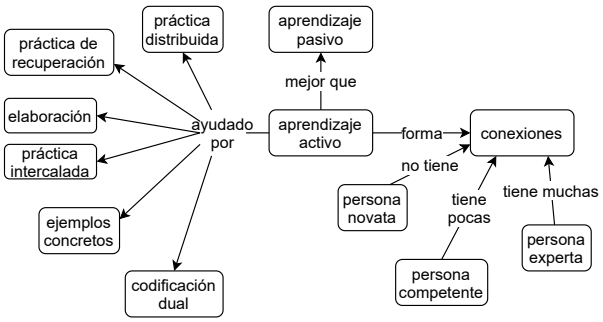

Figura 5.1: Concepts: Active learning

6

Un proceso para diseñar lecciones

La mayoría de las personas diseñan lecciones de esta manera:

1. Alguien te pide que enseñes algo que sabes muy poco o que no has pensado en años.
2. Empiezas escribiendo diapositivas para explicar lo que sabes sobre el tema.
3. Después de 2 o 3 semanas, preparas una tarea basada en lo que has enseñado hasta ahora
4. Repites el paso 3 varias veces.
5. Permaneces sin dormir hasta altas horas de la mañana para crear un examen final y te prometes que serás más organizada/o la próxima vez.

Un método más efectivo es similar en esencia a una práctica de programación llamada **desarrollo-impulsado-por pruebas**(TDD por sus siglas en inglés). Las programadoras y los programadores que usan TDD no escriben software y luego testean/prueban que esté funcionando correctamente. En su lugar, escriben la prueba primero, luego escriben suficiente software nuevo para que esas pruebas pasen.

TDD funciona porque escribir pruebas obliga a quienes programan a ser más precisas/os acerca de lo que están intentando lograr y cómo se ve "hecho". TDD también evita el pulido sin fin: cuando pasan las pruebas, dejas de codificar. Finalmente, esto reduce el riesgo de sesgo de confirmación: alguien que aún no ha escrito un fragmento de software será más objetivo que alguien que acaba de dedicar varias horas al trabajo duro y realmente, realmente quiere terminar.

Un método similar denominado **Reingeniería** funciona muy bien para el diseño de lecciones. Este método fue desarrollado en forma independiente en [Wigg2005; Bigg2011; Fink2013] y está resumido en [McTi2013] En forma simplificada, sus pasos son:

1. Crear o reciclar estudiante tipo (discutido en la siguiente sección) para imaginar a quién estás intentando ayudar y qué les atraerá.
2. Haz una lluvia de ideas para tener una idea aproximada de lo que quieres cubrir, cómo lo vas a hacer, qué problemas o conceptos erróneos esperas encontrar, qué *no* no se va a incluir, y etc. Dibujar mapas conceptuales puede ayudar mucho en esta etapa (Section 3.1).

3. Crea una evaluación sumativa (Section 2.1) para definir tu objetivo general. Esto puede ser el examen final para un curso o el proyecto final para un taller de un día; independientemente de su forma o tamaño, muestra lo lejos que esperas llegar más claramente que una lista puntual de objetivos.

4. Crea evaluaciones formativas eso le dará a las personas una oportunidad para practicar las cosas que están aprendiendo. Estas también te dirán a ti (y a las personas) si están progresando y dónde deben centrar su atención. El mejor camino para hacer esto es detallar los conocimientos y las habilidades utilizadas en la evaluación sumativa que desarrollaste en el paso anterior y luego crear al menos una evaluación sumativa para cada uno.

5. Ordena las evaluaciones formativas para crear un esquema del curso en función de su complejidad, sus dependencias, y cuán bien los temas motivarán a tus aprendices (Section 10.1).

6. Escribe material para conseguir que los/las alumnos/as pasen de una evaluación formativa a la siguiente. Cada hora de instrucción debe constar de tres a cinco episodios.

7. Escribe una descripción resumida del curso para ayudar a tu audiencia objetivo a encontrarlo y averiguar si es adecuado para ella.

Este método ayuda a mantener la enseñanza enfocada en sus objetivos. También asegura que las/los estudiantes no se enfrenten a nada para lo que no están preparadas/os al final del curso.

Incentivos Perversos

La reingeniería no *es lo mismo que grefg:teaching-to-the-testenseñar para el examen. Cuando se usa la reingeniería, el conjunto de docentes establece objetivos para ayudar en el diseño de las lecciones; es posible que ellas/ellos nunca den el examen final que escribieron. En muchos sistemas escolares, por otro lado, una autoridad externa define los criterios de evaluación para todas/todos las/los estudiantes, independientemente de sus situaciones individuales. Los resultados de esas evaluaciones sumativas afectan directamente el salario y promoción de las y los profesores, lo que significa que estos tienen un incentivo para enfocarse en que las/los estudiantes pasen las pruebas en lugar de ayudarles a aprender.*

[Gree2014] argumenta que enfocarse en las pruebas y la medición hace un llamado a quienes tienen el poder de establecer las pruebas, pero es poco probable mejorar los resultados a menos que esto se acompañe con apoyo para que las/los profesores realicen mejoras basadas en los resultados de las pruebas. Esto último a menudo falta porque las grandes organizaciones usualmente valoran uniformidad por sobre la productividad [Scot1998].

El diseño inverso se describe como una secuencia, pero casi nunca se hace de esa manera. Podemos, por ejemplo, cambiar nuestra opinión acerca de lo que queremos

enseñar en base a algo que se nos ocurre mientras estamos escribiendo un MCQ, o re-evaluar a quién estamos intentando ayudar una vez que tengamos un resumen de la lección. Sin embargo, las notas que dejamos atrás deben presentar cosas en el orden descrito anteriormente para que quien tenga que usar o mantener la lección después de nosotras/nosotros pueda seguir sobre nuestro pensamiento [Parn1986].

6.1. Estudiante tipo

El primer paso en el proceso de diseño inverso es averiguar quién es tu audiencia. Una manera para hacer esto es escribir dos o tres **learner personas** como los de Section 1.1. Esta técnica es tomada de diseñadores de experiencia de usuario, quienes crean perfiles breves de usuarios típicos para ayudarles a pensar en su audiencia. Una/un estudiante tipo consiste en:

1. antecedentes generales de la persona
2. lo que ya saben;
3. lo que quieren hacer; y
4. cualquier necesidad especial que tengan.

Las personas en Section 1.1 tienen los cuatro puntos listados anteriormente, junto con un breve resumen de cómo este libro les ayudará. Una/un estudiante tipo para un grupo de voluntarios que realiza talleres de Python los fines de semana sería:

1. Jorge se acaba de mudar de Costa Rica a Canadá para estudiar ingeniería agrícola. El se ha unido al equipo de fútbol universitario y espera aprender a jugar hockey sobre hielo.
2. Aparte de usar Excel, Word y el internet, la experiencia previa más significativa de Jorge con computadoras es ayudar a su hermana a construir un sitio en WordPress para el negocio familiar en casa.
3. Jorge quiere medir las propiedades del suelo en granjas cercanas usando un dispositivo de mano que envía datos a su computadora. Ahora mismo él tiene que abrir cada archivo de datos en Excel, eliminar la primera y la última columna, y calcular algunas estadísticas sobre lo que queda. El tiene que recopilar al menos 600 mediciones en los próximos meses y realmente no quiere tener que hacer estos pasos a mano para cada uno.
4. Jorge puede leer Inglés bien, pero algunas veces le cuesta sostener una conversación hablada que involucre mucha jerga.

En lugar de escribir nuevos tipos para cada lección o curso, las/los profesores usualmente crean y comparten media docena que cubren a todas las personas a las que probablemente enseñan, luego eligen algunos de ese conjunto para describir a la audiencia un material en particular. Los tipos que se usan de esta manera se convierten en un conveniente atajo para los problemas de diseño: Al hablar entre nosotras/os, las/los profesores pueden decir, "¿Jorge entendería por qué estamos haciendo esto?" o "¿Qué problemas de instalación enfrentaría Jorge?"

Sus metas, no las tuyas

Los tipo deberían siempre describir lo que la/el estudiante quiere hacer en lugar de lo que creen que necesitan. Pregúntate lo que ellas/ellos están buscando en línea; probablemente no incluirá jerga que no conocen aún, así que parte de lo que tienes que hacer como diseñadora/diseñador instruccional es descubrir cómo hacer tu lección encontrable (visible). indexfindability!of lessons

6.2. Objetivos de aprendizaje

Evaluaciones formativas y sumativas ayudan a las/los profesores a descubrir lo que van a enseñar, pero para comunicar eso a las/los estudiantes y otro conjunto de docentes, debe tener también una descripción del curso. **learning objectives**. Estos ayudan a asegurar que todos tengan el mismo entendimiento de lo que se supone que una lección debe lograr. Por ejemplo, una declaración como "entendiendo Git" podría significar cualquiera de los siguientes ítemes:

- Las/los estudiantes pueden describir tres maneras en la cual los sistemas de versión de control como Git son mejores que herramientas para compartir archivos como Dropbox y dos formas que son las peores.
- Las/los estudiantes pueden hacer commit a un archivo modificado en un repositorio de Git usando una herramienta GUI de escritorio.
- Las/los estudiantes pueden explicar qué es un HEAD por separado y recuperarlo usando operaciones de línea de comandos.

Objetivos vs. Resultados

*Un objetivo de aprendizaje es lo que una lección se esmera por lograr. Un **resultado de aprendizaje** es lo que realmente se logra, es decir, lo que las/los estudiantes realmente se llevan. El rol de la evaluación sumativa es por lo tanto para comparar resultados de aprendizajes con objetivos de aprendizajes.*

Un objetivo de aprendizaje describe cómo la/el estudiante demostrará lo que ha aprendido una vez que ha completado exitosamente una lección. Más específicamente, este tiene un *verbo medible o verificable* que establece lo que la/el estudiante hará

y especifica los *criterios aceptables de rendimiento*. Escribirlos puede inicialmente parecer restrictivo, pero ellos te harán a ti, a tus compañeras/compañeros docentes, y a tus estudiantes más felices a largo plazo: terminarás con guías claras tanto para su enseñanza como para su evaluación, y tus estudiantes apreciarán tener expectativas claras.

Una forma de comprender lo que constituye un buen objetivo de aprendizaje es ver cómo se puede mejorar uno pobre:

- *La/el estudiante tendrá oportunidad para aprender buenas prácticas de programación.*
  Esto describe el contenido de la lección, no los atributos de éxito de las/los estudiantes.

- *La/el estudiante tendrá una mejor apreciación de las buenas prácticas de programación.*
  Esto no empieza con un verbo activo ni define el nivel de aprendizaje, y el tema de aprendizaje no tiene contexto y no es específico.

- *La/el estudiante comprenderá cómo programar en R.*
  Si bien esto comienza con un verbo activo, no define el nivel aprendizaje, y el tema de aprendizaje es todavía demasiado vago para evaluarlo.

- *La/el estudiante escribirá scripts de análisis de datos para leer, filtrar y resumir datos tabulares usando R.*
  Esto comienza con un verbo activo, define el nivel de aprendizaje y provee contexto para asegurar que los resultados puedan evaluarse.

Cuando se trata de elegir verbos, la mayoría de docentes usan la grefg:bloomstaxonomíaTaxonomía de Bloom. Publicado por primera vez en 1956 y actualizado a principios de siglo [Ande2001], es un marco ampliamente usado para discutir los niveles de comprensión. Su forma más reciente tiene 6 categorías; la lista a continuación da algunos de los verbos típicamente usados en los objetivos de aprendizaje escritos para cada uno:

Recordar: Demostrar memoria del material previamente aprendido recordando hechos, términos, conceptos básicos y respuestas. *(reconocer, listar, describir, nombrar, encontrar.)*

Comprender: Demostrar comprensión de los hechos e ideas organizando, comparando, traduciendo, interpretando, dando descripciones y estableciendo ideas principales. *(interpretar, resumir, parafrasear, clasificar, explicar)*

Aplicar: Resolver problemas nuevos aplicando los conocimientos, hechos, técnicas y reglas adquiridos de una forma diferente *(construir, identificar, usar, planificar, seleccionar)*

Analizar: Examinar y dividir la información en partes identificando motivos o causas, hacer inferencias y encontrar evidencia para apoyar generalizaciones. *(comparar, contrastar, simplificar)*

Evaluar: Presentar y defender opiniones emitiendo juicios sobre información, validez de las ideas, o calidad del trabajo basada en un conjunto de criterios. *(comprobar, elegir, criticar, probar, calificar)*

Crear: Recopilar información de forma diferente combinando elementos en un nuevo patrón o proponiendo soluciones alternativas. *(diseñar, construir, mejorar, adaptar, maximizar, resolver)*

La Taxonomía de Bloom aparece en casi todos los libros de texto sobre educación, pero [Masa2018] encontró que Incluso los educadores experimentados tienen problemas para ponerse de acuerdo sobre cómo clasificar cosas específicas. Los verbos siguen siendo útiles, aunque, al igual que la noción de construir la comprensión en pasos: como Daniel Willingham ha dicho, la gente no puede pensar sin algo en qué pensar [Will2010], y esta taxonomía puede ayudar a las/los docentes a asegurarse que las/los estudiantes tengan esas cosas cuando las necesiten.

Otra manera de pensar acerca de los objetivos de aprendizaje proviene de [Fink2013], el cual define aprendizaje en términos del cambio que se supone que debe producirse en la/el estudiante. La **Taxonomía de Fink** también tiene seis categorías, pero a diferencia de las de Bloom ellas son complementarias en lugar de jerárquicas:

Conocimiento fundamental: Comprender y recordar información e ideas. *(recordar, comprender, identificar)*

Aplicación: habilidades, pensamiento crítico, gestión de proyectos. *(usar, resolver, calcular, crear)*

Integración: conectar ideas, experiencias de aprendizaje y vida real *(conectar, relatar, comparar)*

Dimensión Humana: Aprender sobre sí misma/mismo y otras/otros. *(llegar a verse a sí misma/mismo, entender a las/los demás en términos de, decidir ser)*

Cuidando: Desarrollar nuevos sentimientos, intereses y valores *(emocionarse, estar preparada/preparado para, valorar)*

Aprendiendo a aprender: Convertirse en una/un mejor estudiante. *(identificar la fuente de información para, enmarcar preguntas útiles sobre)*

Un conjunto de objetivos de aprendizaje basados en esta taxonomía para un curso introductorio sobre HTML y CSS sería:

- Explicar qué son las propiedades de CSS y cómo funcionan los selectores de CSS.
- Diseñar una página web usando etiquetas comunes y propiedades CSS.

- Comparar y contrastar la escritura HTML y CSS para escribir con herramientas de edición de escritorio.
- Identificar y corregir problemas en páginas web de muestra que dificultarían la interacción de las personas con discapacidad visual.
- Describir las características de los sitios web favoritos cuyo diseño te atraiga de forma particular y explica el por qué.
- Describir tus dos fuentes de información favoritas en línea acerca de CSS y explica qué te gusta de ellas.

6.3. Mantenimiento

Una vez que una lección ha sido creada alguien debe mantenerla, y hacerlo es mucho más fácil si se ha construido de manera que se pueda mantener. ¿Pero qué significa exactamente "mantenible´´? La respuesta corta es que una lección es mantenible si es más barato actualizarla que reemplazarla. Esta ecuación depende de 4 factores:

¿Qué tan bien documentado está el diseño del curso? Si la persona que realiza el mantenimiento no conoce (o no recuerda) lo que se supone la lección debe lograr, o por qué los temas son presentados en un orden en particular, le llevará más tiempo actualizarla. Una razón para usar el diseño inverso es captar decisiones sobre por qué cada curso es como es.

¿Qué tan fácil es para los colaboradores ayudar? Las/los docentes suelen compartir material enviándose por correo archivos de PowerPoint entre ellas/ellos o poniéndolos en una unidad compartida. Herramientas de escritura colaborativa como *Google Docs*[1] and wikis son una gran mejora, ya que permiten que muchas personas actualicen el mismo documento y comenten las actualizaciones de otras personas. El sistema de control de versiones usado por programadores, tales como *GitHub*[2], son otro enfoque. Permiten que cualquier número de personas trabajen de forma independiente y luego unir sus cambios en forma controlada y revisable. Desafortunadamente, los sistemas de control de versión tienen una curva de aprendizaje pronunciada y no manejan formatos de documentos de oficina comunes.

Qué tan dispuestas están las personas a colaborar. Las herramientas necesarias para construir un Wikipedia para lecciones existen hacen veinte años, pero la mayoría de docentes no escriben ni comparten sus lecciones de la misma manera en que escriben y comparten las entradas de enciclopedias.

[1] http://docs.google.com
[2] http://github.com

Qué tan útil es compartir en realidad. La **Paradoja de la Reusabilidad** establece que cuanto más reutilizable es un objetivo de aprendizaje, menos efectivo pedagógicamente es [Wile2002]. La razón es que una buena lección se parece más a una novela que a un programa: sus partes están estrechamente acopladas en lugar de ser cajas negras independientes. Por lo tanto, la reutilización directa puede ser el objetivo equivocado de las lecciones; podríamos llegar más lejos tratando de hacerlas más fáciles de mezclar.

Si la paradoja de Reusabilidad es cierta, la colaboración será más probable si las cosas en las que se colabora son pequeñas. Esto se ajusta a la teoría de Mike Caulfield *choral explanations*[3](explicaciones corales), que sostiene que sitios como *Stack Overflow*[4] tienen éxito porque proporcionan un coro de respuestas para cada pregunta, cada una de las cuales es más adecuada para una persona que pregunta y que es ligeramente diferente a las demás que preguntan. Si esto es correcto las lecciones de mañana puedes ser visitas guiadas de repositorios de preguntas y respuestas seleccionadas por la comunidad diseñadas para estudiantes en niveles muy diferentes.

6.4. Ejercicios

Crear estudiantes tipo (grupos pequeños/30')

Trabajando en grupos pequeños, crea un tipo de 4 puntos que describa a una/uno de sus estudiantes estándar.

Clasificar Objetivos de Aprendizaje (pares/10')

Mira el ejemplo de objetivos de aprendizaje para un curso introductorio sobre HTML y CSS en Section 6.2 y clasifica cada uno de acuerdo a la Taxonomía de Bloom. Compara tus respuestas con las de tu pareja. ¿Dónde estuvieron de acuerdo y en desacuerdo?

Escribir Objetivos de Aprendizaje (pares/20')

Escribe uno o más objetivos de aprendizaje para algo que actualmente enseñas o planeas enseñar utilizando la Taxonomía de Bloom. Trabajando con una pareja, critica y mejora los objetivos. ¿Cada uno tiene un verbo verificable y establece claramente los criterios para un desempeño aceptable?

[3] https://hapgood.us/2016/05/13/choral-explanations/
[4] https://stackoverflow.com/

Escribir más Objetivos de Aprendizaje (pares/20')

Escribe uno o más objetivos de aprendizaje para algo que actualmente enseñas o planeas enseñar utilizando la Taxonomía de Fink. Trabajando con una pareja, critica y mejora los objetivos.

Ayúdame a hacerlo sola/solo (grupos pequeños /15')

El teórico de la educación Lev Vygotsky introdujo la noción de una **Zona de Desarrollo Proximal** (ZPD por sus siglas en inglés), la cual incluye los problemas que las personas no pueden resolver aún por sí mismas pero que son capaces de resolver con la ayuda de una mentora/un mentor. Estos son los problemas que resultan más fructíferos abordar a continuación, ya que están fuera de tu alcance pero son alcanzables.

Trabajando en grupos pequeños, escoge una/un estudiante tipo que hayas desarrollado y describe dos o tres problemas que se encuentran en la ZPD de esa/ese estudiante.

Construyendo lecciones, restando complejidad (individual/20')

Una forma para construir una lección de programación es escribir el programa que deseas que las/los estudiantes terminen, luego elimina la parte más compleja que deseas que escriban y conviértela en el último ejercicio. Tú puedes luego remover la siguiente parte más compleja que deseas que escriban y conviértela en el penúltimo ejercicio, etc. Todo lo que quede después de haber retirado los ejercicios, como cargar librerías o leer datos, se convierte en el código al inicio les das. Elige un programa o página web que desees que tus estudiantes puedan crear, y trabaja hacia atrás para dividirlo en partes que se asimilen. ¿Cuántos hay? ¿Qué idea clave introduce cada uno?

Rareza no esencial (individual/15')

Betsy Leondar-Wright acuñó la frase "*inessential weirdness* [5]"(rareza no esencial) para describir cosas que hacen los grupos que no son realmente necesarias, pero que alienan a personas que aún no son miembros de ese grupo Sumana Harihareswara luego usó esta noción como base para una charla sobre *inessential weirdnesses in open source software*[6](rarezas no esenciales en el software de código abierto), que incluye códigos como el uso de herramientas de línea de comandos con nombres crípticos. Toma unos minutos para leer estos artículos, luego has una lista de las rarezas no esenciales que crees que tus estudiantes podrían encontrar Cuando les enseñes por primera vez ¿Cuántas de estas puedes evitar?

[5] http://www.classmatters.org/2006_07/its-not-them.php
[6] https://www.harihareswara.net/sumana/2016/05/21/0

PETE (individual/15')

Un patrón que trabaja bien para lecciones de programación es PETE: Introduce el **P**roblema, trabaja a través de un **E**jemplo, explica la **T**eoría, y luego **E**labora un Segundo ejemplo para que las/los estudiantes puedan ver qué es específico en cada caso y qué se aplica a todos los casos. Elige algo que ya hayas enseñado o les hayan enseñado. Y delinea una pequeña lección que siga estos cinco pasos. .

PRIMM (individual/15')

Otro patrón de lección es PRIMM [Sent2019]: **P**redecir el comportamiento o salida de un programa, Correr, del inglés **R**un, el programa para ver lo que realmente hace, **I**nvestigar por qué lo hace pasando a través del mismo en un depurador o dibujando el flujo de control. **M**odificar el programa (o sus entradas), y luego hacer, del inglés **M**ake, algo similar desde cero. Elige algo que hayas enseñado o te hayan enseñado recientemente y bosqueja una breve lección que siga los siguientes cinco pasos.

Concreto-Figurativo-Abstracto (pares/15')

Concrete-Representational-Abstract [7] (Concreto-Representativo-Abstracto) (CRA), por sus siglas en inglés, es un enfoque para introducir nuevas ideas que es usado principalmente con estudiantes más jóvenes: manipular físicamente un objeto **C**oncreto, **R**epresentar el objeto con una imagen, luego realizar las mismas operaciones usando números, símbolos o algo **A**bstracto.

1. Escribe cada uno de los números 2, 7, 5, 10, 6 en una nota adhesiva.
2. Simula un bucle que encuentre el valor más grande buscando cada uno por turno (concreto).
3. Dibuja un diagrama del proceso que usaste etiquetando cada paso (representativo)
4. Escribe instrucciones que alguien más podría seguir por medio de los pasos (abstracto)

Compara tus materiales representativo y abstracto con tus compañeras/compañeros.

Evaluación de un repositorio de lecciones (grupos pequeños/10')

[Leak2017] explora por qué las/los docentes de ciencias computacionales no utilizan sitios para compartir lecciones y recomienda formas para hacerlas más atractivas:

[7] https://makingeducationfun.wordpress.com/2012/04/29/concrete-representational-abstract-cra/

1. La página de destino debe permitir a los visitantes del sitio identificar sus antecedentes y sus intereses al visitarlo. Los sitios deben hacer dos preguntas: "¿Cuál es su función actual? " y "¿En qué curso y grado estás interesada/interesado? "

2. Los sitios deben mostrar todos los recursos de aprendizaje en el contexto del curso completo para que los usuarios potenciales puedan comprender su contexto de uso previsto.

3. A muchas/muchos docentes les preocupa que sus pares juzguen su (falta de) conocimiento si publican en los foros de discusión de los sitios. Por tanto, estos foros deberían permitir la publicación anónima.

En grupos pequeños, Discute si estas tres características serían suficientes para convencerte de usar un sitio para compartir lecciones, y si no, lo que haría.

Revisión

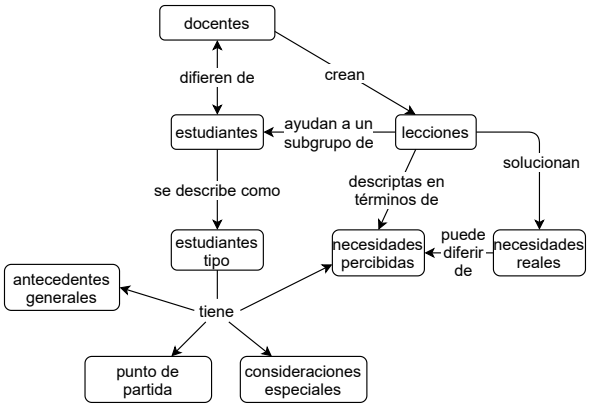

Figura 6.1: Concepto: Estudiantes

7

Conocimiento de la pedagogía del contenido

Cada docente necesita tres cosas:

conocimiento del contenido por ejemplo, como programar;

conocimiento general de pedagogía por ejemplo, comprensión de la psicología del aprendizaje; y

conocimiento de la pedagogía del contenido (PCK por Pedagogical Content Knowledge en inglés), que es el conocimiento específico de cómo enseñar un concepto particular a una audiencia en particular. En informática, PCK incluye cosas tales como qué ejemplos usar cuando se enseña, como se incluyen parámetros en una función o qué conceptos erróneos son los más comunes sobre etiquetas HTML anidadas.

Podemos agregar conocimiento técnico a este conjunto [Koeh2013], pero eso no cambia el punto clave: no es suficiente saber sobre el tema y como enseñar—tienes que saber cómo enseñar ese tema en particular [Maye2004]. Este capítulo resume algunos resultados de investigaciones sobre enseñanza de informática para añadir a tu colección de PCK.

Como con toda investigación, se requiere cierta precaución al interpretar los resultados:

Las teorías cambian a medida que se obtienen más datos. La investigación en educación en informática (CER, por Computing education research en inglés) es una disciplina nueva: la Sociedad Americana de Educación en Ingeniería fue fundada en 1893 y el Consejo Nacional de Profesores de Matemática en 1920, pero la Asociación de Profesores de Informática no se creó hasta 2005. Mientras que existe un flujo constante de nuevo conocimiento en conferencias como SIGCSE[1], ITiCSE[2], y ICER[3], simplemente no sabemos tanto sobre como aprender a programar como sí sabemos sobre aprender a leer, jugar un deporte o resolver cálculos simples.

La mayoría de las personas en estos estudios son WEIRD: viven en sociedades occidentales, democráticas, industrializadas y con alto nivel de riqueza y educación (WEIRD por Western, Education, Industrialized, Rich, and Democratic en

[1] http://sigcse.org/
[2] http://iticse.acm.org/
[3] https://icer.hosting.acm.org

inglés) [Henr2010]. Además, son principalmente jóvenes en contextos formales, ya que es la población a la que la mayoría de las personas que investigan tienen fácil acceso. Sabemos mucho menos sobre adultos, grupos marginados y estudiantes en contextos no formales o **usuario final programador**, aún cuando son la mayoría.

Si esto fuera un ensayo académico, empezaría la mayoría de oraciones con frases como, "Algunas investigaciones parece indicar que…" Pero dado que los y las docentes en las aulas tienen que tomar decisiones independientemente de si las investigaciones tienen respuestas claras o no, este capítulo presenta las mejores conjeturas prácticas en lugar de sutiles posibilidades.

Jerga

Como cualquier especialidad, CER tiene jerga. ***CS1*** *refiere a un curso introductorio de un semestre de duración, donde los estudiantes aprenden variables, bucles y funciones por primera vez, mientras que* ***CS2*** *refiere a un segundo curso que cubre las estructuras de datos básicas como pilas y colas, y* ***CS0*** *se refiere a un curso introductorio para personas sin experiencia previa y que no tienen intención de continuar con computación de inmediato. Las definiciones completas de estos términos se encuentran en Lineamientos del programa ACM*[4].

7.1. ¿Qué les estamos enseñando ahora?

Se sabe muy poco sobre qué enseñan en coding bootcamps e iniciativas no formales, en parte porque muchas personas son reticentes a compartir los programas. Sabemos más sobre lo que se enseña en instituciones [Luxt2017]:

Temas	**%**	**Temas**	**%**
Proceso de programación	87 %	Tipos de datos	23 %
Pensamiento abstracto para programación	63 %	Entrada /Salida	17 %
Estructuras de datos	40 %	Librerías	15 %
Conceptos orientados a objetos	36 %	Variables y Asignación	14 %
Estructuras de control	33 %	Recursión	10 %
Operaciones y Funciones	26 %	Punteros y administración de memoria	5 %

Títulos de temas de alto nivel como estos pueden esconder una gran cantidad de fallas. Un resultado más tangible surge de [Rich2017], donde revisa cien artículos para encontrar trayectorias de aprendizaje para clases de computación en escuelas primarias y secundarias. Sus resultados para la secuenciación, la repetición y los condicionales son esencialmente mapas conceptuales colectivos que combinan y ra-

[4] https://www.acm.org/education/curricula-recommendations

cionalizan el pensamiento implícito y explícito de gran cantidad de docentes. (Figure 7.1).

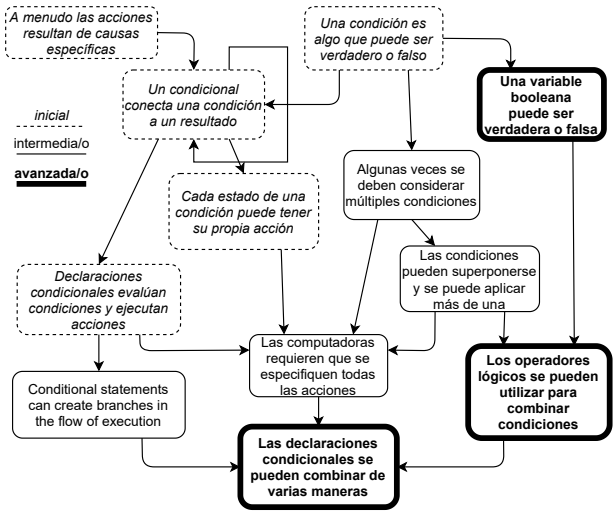

Figura 7.1: Trayectoria de aprendizaje para condiciones (extraído de [Rich2017])

7.2. ¿Cuánto están aprendiendo?

Puede haber un mundo de distancia entre lo que enseñan las/los docentes y cuánto aprenden las personas. Para explorar lo último, debemos usar otras medidas o hacer estudios directos. Tomando el primer enfoque, aproximadamente dos tercios de las/los estudiantes de nivel superior aprueban su primer curso de informática, con algunas variaciones dependiendo del tamaño de la clase, pero sin diferencias significativas a lo largo del tiempo o basadas en el lenguaje [Benn2007a; Wats2014].

¿Cómo afecta la experiencia previa a estos resultados? Para responder esto, [Wilc2018] compara el desempeño y la confianza de personas novatas con y sin experiencia previa en programación en CS1 y CS2 (ver más abajo). Encontraron que personas novatas con experiencia previa superaron a personas sin experiencia en un 10% en CS1, pero esas diferencias desaparecieron hacia el final de CS2. También encontraron que las mujeres con experiencia previa superaron a sus pares masculinos en todas las áreas, pero siempre tenían menos confianza en sus habilidades. (Section 10.4).

En cuanto a estudios sobre cuanto aprenden las personas novatas, [McCr2001] presenta un estudio internacional en múltiples espacios que luego fue replicado por [Utti2013]. De acuerdo al primer estudio, "...los decepcionantes resultados sugieren que muchas/os estudiantes no saben cómo programar al final de los cursos introductorios". Más específicamente, "Para una muestra combinada de 216 estudiantes de cuatro universidades, la puntuación media fue de 22,89 sobre 110 puntos en los criterios generales de evaluación desarrollados para este estudio" Este resul-

tado puede hablar tanto de las expectativas de profesores como de la habilidad de las/los estudiantes, pero de cualquier manera, nuestra primera recomendación es *medir y hacer un seguimiento de los resultados* de tal manera que se puedan comparar a través del tiempo para que puedas saber si tus lecciones se están volviendo más o menos efectivas.

7.3. ¿Qué conceptos erróneos tienen las personas novatas?

Chapter 2 explicó por qué aclarar los conceptos erróneos a las personas novatas es tan importante, como enseñarles estrategias para resolver problemas. La mayor confusión de las personas novatas —a veces llamada el "superbug" en programación —es la creencia de que las computadoras entienden lo que las personas quieren decir de la misma manera que cualquier ser humano lo haría [Pea1986]. Nuestra segunda recomendación es entonces *enseñar a las personas novatas que las computadoras no entienden los programas*, es decir, que llamar a una variable "precio" no garantiza que su valor sea realmente un precio.

[Sorv2018] muestra más de cuarenta conceptos erróneos que docentes también pueden intentar aclarar, muchos de las cuales también se discuten en el estudio de [Qian2017]. Una es la creencia de que las variables en los programas funcionan de la misma manera que en planillas de cálculo, es decir, que luego de ejecutar:

```
nota = 65
total = nota + 10
nota = 80
print(total)
```

el valor de `total` será 90 en vez de 75 [Kohn2017]. Este es un ejemplo de la forma en que las personas novatas construyen un modelo mental plausible pero erróneo haciendo analogías; otras confusiones incluyen:

- Una variable guarda la historia de los valores que le fueron asignados, es decir, recuerda cuál solía ser su valor.
- Está garantizado que dos objetos con el mismo valor para sus atributos `nombre` o identificación `id` son el mismo objeto.
- Las funciones son ejecutadas cuando se las define, o son ejecutadas en el orden en que son definidas.
- La condición de un bucle `while` se evalúa constantemente, y el bucle se detiene tan pronto como se vuelve falso. Por el contrario, la condición de las sentencias`if` es constantemente evaluada, y sus declaraciones son ejecutadas tan pronto como la condición se vuelve verdadera, independientemente de dónde se encuentre el flujo de control en ese momento.

- Las asignaciones modifican valores, es decir, después de `a = b`, la variable `b` queda vacía.

7.4. ¿Qué errores cometen las personas novatas?

Los errores que cometen las personas novatas nos indican que priorizar cuando enseñamos, pero resulta que la mayoría de las personas que enseñan no saben cuán comunes son los diferentes tipos de errores. El el estudio más importante es el de [Brow2017], que encontró que la falta de paréntesis y comillas son el tipo de error más común en programas Java escritos por personas novatas, pero además son los más simples de resolver, mientras que algunos errores (como poner la condición de un `if` en `{}` en vez de `()`) se cometen solo una vez. No extraña que los errores que producen problemas de compilación son resueltos mucho más rápido que aquellos que no lo hacen. Algunos errores, en cambio, se repiten muchas veces, como llamar métodos con los argumentos incorrectos (ej. pasar una cadena de caracteres en vez de un número entero).

No es correcto vs. No está resuelto

Una dificultad en una investigación como esta es distinguir los errores del trabajo en proceso. Por ejemplo, una estructura `if` *vacía o un método que se define pero aún no se ha usado puede ser señal de que el código está incompleto y que no es un error.*

[Brow2017] también comparó los errores que las personas novatas realmente cometen con los que sus docentes pensaron que cometieron. Descubrieron que, "…las/los docentes llegaron a un escaso consenso sobre cuales son los errores más frecuentes, sus clasificaciones tenían solo una correspondencia moderada con la de las/los estudiantes en los…datos, y la experiencia de las/los docentes no tuvo ningún efecto en este nivel de acuerdo." Por ejemplo, confundir = (asignación) y == (igualdad) no eran tan común como la mayoría de las/los docentes creían.

No solo por el código

[Park2015] recopiló datos de un editor HTML en línea, durante un curso introductorio de desarrollo web y encontró que casi todas/os las estudiantes cometieron errores de sintaxis que permanecieron sin ser resueltos por semanas durante el curso. El 20% de esos errores están relacionados con reglas relativamente complejas que indican cuándo *es válido que los elementos HTML estén anidados entre sí, pero el 35% está relacionado a sintaxis de etiquetas más simples que determinan* cómo *los elementos HTML están anidados. La tendencia de muchas/os docentes a decir, "Pero las reglas son simples," es un buen ejemplo del punto ciego de las personas expertas que se analiza en el Chapter 3…*

7.5. ¿Cómo programan las personas novatas?

[Solo1984; Solo1986] fue pionero en la exploración de las estrategias de programación de personas novatas y expertas. El hallazgo clave es que las personas expertas saben al mismo tiempo el "que" y el "como," es decir, entienden que poner en los programas *y* tienen un conjunto de patrones o plan para guiar su construcción. Las personas principiantes no tienen ninguna de las dos cosas, pero la mayoría de las/los docentes se enfocan únicamente en lo primero, a pesar de que los errores son usualmente causados por no tener una estrategia para resolver el problema, en lugar de falta de conocimiento sobre el lenguaje. Un trabajo reciente mostró la efectividad de enseñar cuatro habilidades distintas en un orden específico [Xie2019]:

	semántica del código	**plantillas asociadas a objetivos**
leyendo	1. Leer el código y predecir su comportamiento	3. reconocer plantillas y sus usos
escribiendo	2. Escribir la sintaxis correcta	4. usar las plantillas para alcanzar objetivos

Por lo tanto, nuestras recomendaciones son que *estudiantes lean código, luego lo modifiquen, luego lo escriban*, y practiquen usándolos. [Mull2007b] es uno de los tantos estudios que muestran los beneficios de enseñar patrones comunes de manera explícita y descomponer los problemas en patrones comunes crea oportunidades naturales para crear y etiquetar sub-objetivos [Marg2012; Marg2016].

7.6. ¿Cómo identifican y resuelven errores las personas novatas??

Una década atrás, [McCa2008] escribió, "Es sorprendente el poco espacio que se dedica a los errores y cómo resolverlos en la mayoría de los libros introductorios de programación."

e *introducir patrones comunes explícitamente y que las/los estudiantes.* Poco ha cambiado desde entonces: hay cientos de libros sobre compiladores y sistemas operativos, pero solo unos pocos sobre depuración de errores, y nunca he visto un curso de pregrado dedicado a este tema.

[List2004; List2009] encontró que a muchas personas novatas les cuesta predecir el resultado de pocas líneas de código y seleccionar la salida correcta del código a partir de un conjunto de posibilidades cuando les decían lo que se suponía que el código debía hacer. Más recientemente, [Harr2018] encontró que la brecha entre poder entender el código y poder escribirlo se ha cerrado en gran medida por CS2, pero que las personas novatas que aún tienen esa brecha es probable que les vaya mal.

Nuestra quinta recomendación es entonces *enseñar explícitamente a las personas principiantes como depurar errores*. [Fitz2008; Murp2008] encontraron que las

personas que pueden resolver errores son buenas programando, pero no todas las personas que son buenas programando son buenas resolviendo errores. Aquellas personas que usaron un depurador de errores simbólico para recorrer sus programas, rastrearon su ejecución manualmente, escribieron pruebas, y releyeron las especificaciones frecuentemente, todos hábitos que se pueden enseñar. Sin embargo, rastrear la ejecución paso a paso a veces se usa de manera ineficaz: por ejemplo, una persona novata podría poner la misma declaración `print` en ambas partes de una estructura `if-else`. Las personas novatas también comentarían líneas de código que en realidad eran correctas al tratar de aislar un problema; las/los docentes pueden cometer estos dos errores deliberadamente, señalarlos, y corregirlos para ayudar a las personas novatas a aprender a resolverlos.

Enseñar a principiantes cómo depurar errores también puede ayudar a que las clases sean más simples de manejar. [Alqa2017] encontró que estudiantes con mayor experiencia resolvieron problemas de depuración de errores significativamente más rápido, pero los tiempos variaron ampliamente: 4–10 minutos fue el rango típico para ejercicios individuales, lo cual significa que algunas/os estudiantes necesitan 2–3 más tiempo que otros para resolver el mismo ejercicio. Enseñar a estudiantes que hacen las cosas más lentamente, lo que las personas más rápidas están haciendo va a ayudar a que el progreso de todo el grupo en general sea más uniforme.

La depuración de errores depende de ser capaz de leer el código, algo quemúltiples estudios han demostrado, que es la forma más efectiva de encontrar errores [Basi1987; Keme2009; Bacc2013]. La rúbrica de calidad de código desarrollada por [Steg2014; Steg2016a] es una buena lista de verificación de elementos a revisar, aunque se presenta mejor de a partes en lugar de toda junta.

Hacer que las/los estudiantes lean código y resuman su comportamiento es un buen ejercicio (Section 5.1), pero frecuentemente lleva mucho tiempo practicarlo en clase. Hacer que las/los estudiantes predigan la salida de un programa justo antes de que sea ejecutado, por otro lado, ayuda a reforzar el aprendizaje (Section 9.11) y además les da la oportunidad ideal para hacer el tipo de preguntas "que pasaría si". Docentes o estudiantes pueden además rastrear los cambios en las variables a medida que avanzan, algo que [Cunn2017] encontró efectivo (Section 12.2).

7.7. ¿Qué pasa con las pruebas?

Programadoras/es novatas/os parecen tan reacias/os a testear software como aquellas personas profesionales. No hay duda de que hacerlo es valioso—[Cart2017] encontró que personas novatas con un alto rendimiento dedican mucho tiempo a testear el código, mientras que las personas novatas con un bajo rendimiento dedican mucho más tiempo a trabajar en el código con errores—y muchas/os docentes requieren que las/los estudiantes escriban pruebas en las actividades. ¿Pero qué tan bien lo hacen? Una posible respuesta proviene de [Bria2015], que calificó el código de las/los estudiantes según la cantidad de situaciones, definidas por la/el docente,

que pasaban correctamente las pruebas de los programas, y, a la inversa, calificó los casos de prueba escritos por estudiantes de acuerdo con la cantidad de errores detectados que habían sido sembrados deliberadamente. Ellos encontraron que las pruebas de personas novatas usualmente tienen una baja cobertura (es decir, no testean gran parte del código) y que usualmente testean muchas cosas al mismo tiempo, lo que dificulta determinar las causas de los errores.

Otra respuesta proviene de [Edwa2014b], que examinó todos los errores en el código presentado por personas novatas combinados e identificó aquellos detectados por el conjunto de pruebas de las/los novatos. Ellos encontraron que las pruebas de novatos solo detectaban un promedio de 13.6 % de los errores presentes en la totalidad de los programas. Además, el 90 % de las pruebas de las/los novatos fueron muy parecidas, lo que indica que las/los novatos escriben pruebas principalmente para confirmar que el código está haciendo lo que se supone que debe hacer en lugar de encontrar situaciones en las que no ocurre esto. Un enfoque para enseñar mejores prácticas para generar pruebas es definir un problema de programación proporcionando un conjunto de pruebas que deben ser pasadas en lugar de una descripción (Section 12.1). Sin embargo, antes de hacerlo tómate un momento para ver cuántas pruebas has escrito en tu propio código recientemente, y luego decide si estás enseñando lo que crees que las personas deberían hacer, o lo que ellos (y tú) realmente hacen.

7.8. ¿Importa el lenguaje?

La respuesta corta es "sí": Las personas novatas aprenden a programar más rápido y aprenden más usando herramientas basadas en bloques como Scratch (Figure 7.2) [Wein2017]. Una razón es que esos sistemas basados en bloques reduce la carga cognitiva al eliminar la posibilidad de cometer errores de sintaxis. Otra razón es que esas interfaces de bloques promueven la exploración de una manera que el texto no lo hace: como todas las buenas herramientas, Scratch se puede aprender accidentalmente [Malo2010].

Pero, ¿qué sucede *luego* de los bloques? [Chen2018] encontró que las personas cuyo primer lenguaje de programación fue gráfico, tenían calificaciones más altas en cursos introductorios de programación en comparación con las personas cuyo primer lenguaje era textual cuando los lenguajes se aprendieron durante o antes de los primeros años de adolescencia. Nuestra sexta recomendación es *introducir interfaces basadas bloques a niños, niñas y adolescentes* antes de avanzar a sistemas basados en texto. La calificación de edad corresponde a que Scratch deliberadamente parece destinado a usuarias y usuarios jóvenes, y puede ser difícil convencer a personas adultas para que lo tomen en serio.

Scratch probablemente ha sido estudiado más que cualquier otra herramienta de programación. Un ejemplo es [Aiva2016], que analizó más de 250.000 proyectos de Scratch y encontró (entre otras cosas) que alrededor del 28 % de esos proyectos

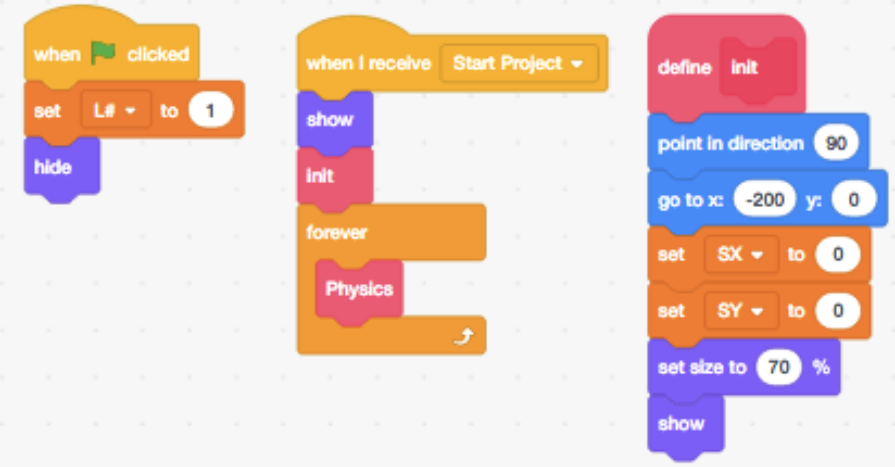

Figura 7.2: Scratch

tenían algunos bloques que nunca eran usados o activados. Como en la sección anterior sobre los programas en Java incompletos versus incorrectos, los autores plantean la hipótesis de que los usuarios pueden estar utilizando estos bloques como borrador para hacer un seguimiento de las partes del código que no quieren eliminar (todavía). Otros ejemplos son [Grov2017; Mlad2017], que estudiaron a personas novatas aprendiendo sobre bucles en Scratch, Logo, y Python. Encontraron que las ideas erróneas sobre bucles se minimizan cuando se usa un lenguaje basado en bloques en lugar de un lenguaje basado en texto. Además, a medida que las tareas se vuelven más complejas (como el uso de bucles anidados) las diferencias son mayores.

Más difícil de lo necesario

Las personas que crean lenguajes de programación hacen que esos lenguajes sean más difíciles de aprender al no realizar pruebas básicas de usabilidad. Por ejemplo, [Stef2013] encontró que, "...las tres palabras más comunes para generar bucles en ciencias de la computación, `for`*,* `while`*, and* `foreach`*, fueron calificadas como las tres opciones más antiintuitivas por personas que no programan." Su trabajo muestra que sintaxis al estilo de C (como se usa en Java y Perl) son tanto más difíciles de aprender para personas novatas como una sintaxis diseñada de manera aleatoria, pero que la sintaxis de lenguajes como Python y Ruby es significativamente más fácil de aprender, y la sintaxis de un lenguaje cuyas características son probadas antes de incorporarse es aún más fácil. [Stef2017] es un útil y breve resumen de lo que realmente sabemos sobre el diseño de lenguajes de programación y por qué creemos que es cierto, mientras que [Guzd2016] expone cinco principios que los lenguajes de programación para estudiantes deberían seguir.*

Programación orientada a objetos

Los objetos y clases son herramientas poderosas para personas con experiencia en programación, y muchas/os docentes promueven un enfoque de **primero objetos** para enseñar a programar (aunque a veces no concuerdan exactamente con lo que eso significa [Benn2007b]). [Sorv2014] describe y motiva este enfoque, y [Koll2015] describe tres generaciones de herramientas designadas para soportar programación de personas novatas en ambientes orientados a objetos.

Introducir objetos temprano tiene algunos desafíos. [Mill2016b] encontró que la mayoría de las personas novatas que usan Python tenían problemas para entender `self` (que refiere al objeto actual): las personas lo omitieron en las definiciones de métodos, no lo usaron cuando hacían referencia a los atributos del objeto, o ambas. [Rago2017] encontró algo similar en estudiantes de nivel secundario, y además encontró que docentes de secundaria usualmente tampoco tenían claro el concepto. En resumen, recomendamos que docentes *comiencen con funciones en vez de objetos*, es decir, que a las/los estudiantes no se les enseñe cómo definir clases hasta que comprendan las estructuras básicas de control y los tipos de datos.

Declaración de tipos

Las personas que programan han discutido durante décadas acerca de si los tipos de datos de variables deberían ser declarados o no, usualmente basándose en su experiencia personal como profesionales en lugar de tener en cuenta el tipo de datos. [Endr2014; Fisc2015] encontraron que pedir a personas novatas que declaren los tipos de variables suma cierta complejidad a los programas, pero se compensa rápidamente al actuar como documentación del uso de los métodos —en particular, evita preguntas sobre qué métodos están disponibles y cómo usarlos.

Nombrando variables

[Kern1999] escribió, "A menudo se alienta a la gente que programa a usar nombres de variables largos independientemente del contexto. Esto es un error: la claridad a menudo se logra siendo breves." Mucha gente que programa cree esto, pero [Hofm2017] encontró que usar palabras completas en nombres de variables condujo, en promedio, a una comprensión un 19 % más rápida en comparación con letras y abreviaturas. En contraste, [Beni2017] encontró que el uso de nombres de variables de una sola letra no afectaba la capacidad de las personas principiantes para modificar el código. Esto puede deberse a que sus programas son más cortos que los de profesionales o porque algunos nombres de variables de una sola letra tienen tipos y significados implícitos. Por ejemplo, la mayoría de las personas que programan asumen que `i`, `j` y `n` son enteros y que `s` es una cadena de caracteres, mientras que `x`, `y` y `z` son números de punto flotante o enteros más o menos equivalentemente.

¿Qué tan importante es esto? [Bink2012] reportó que leer y comprender el código es fundamentalmente distinto de leer prosa: "... la estructura más formal y la sintaxis del código fuente permite a la gente que programa, asimilar y comprender partes

del código rápidamente independientemente del estilo. En particular... las guías y los planes de programas juegan un papel importante en la comprensión." También encontró que a las/los desarrolladoras/es con experiencia no les afecta el estilo del identificador, entonces nuestra recomendación es utilizar un estilo coherente en todos los ejemplos. Dado que la mayoría de los lenguajes tienen guías de estilo (por ejemplo, PEP 8[5] para Python) y herramientas para verificar que el código siga esas pautas, nuestra recomendación completa es *usar herramientas para garantizar que todos los ejemplos de código se adhieran a un estilo consistente*.

7.9. ¿Ayudan mejores mensajes de error?

Los mensajes de error incomprensibles son una fuente importante de frustración para personas novatas (y también para personas experimentadas). Por lo tanto, varias/os investigadoras/es han explorado si mejores mensajes de error ayudan a evitar esto. Por ejemplo, [Beck2016] reescribió algunos de los mensajes del compilador de Java para que en lugar de:

```
C:\stj\Hola.java:2: error: cannot find symbol
        public static void main(string[ ] args)
^
1 error
Process terminated ... there were problems.
```

las personas vean:

```
Looks like a problem on line number 2.
If ``string'' refers to a datatype, capitalize the 's'!
```

Efectivamente, las personas novatas que recibieron estos mensajes repitieron menos errores y cometieron menos errores en general.

[Bari2017] fue más allá y usó seguimiento ocular para mostrar que a pesar de las quejas de las personas que escriben los compiladores, las personas realmente leen los mensajes de error—de hecho, pasan del 13 al 25 % de su tiempo haciendo eso. Sin embargo, leer mensajes de error resulta ser tan difícil como leer el código fuente, y la dificultad en leer los mensajes de error predice fuertemente el desempeño de la tarea. Por lo tanto, quienes enseñan *deben mostrar a sus estudiantes cómo leer e interpretar mensajes de error*. [Marc2011] tiene una rúbrica con respuestas a los mensajes de error que puede ser útil para calificar ese tipo de ejercicios.

[5] https://www.python.org/dev/peps/pep-0008/

¿Ayuda la visualización?

Visualizar la ejecución de un programa es una idea siempre popular, y herramientas como el Tutor de Python en línea [Guo2013] y Loupe[6] (que muestra como funciona un bucle de eventos de JavaScript) son útiles para enseñar. Sin embargo, las personas aprenden más al construir visualizaciones que al ver visualizaciones construidas por otras personas [Stas1998; Ceti2016], entonces, ¿las visualizaciones realmente ayudan al aprendizaje?

Para responder esto, [Cunn2017] replicó un estudio previo sobre los tipos de esquemas que realizan las/los estudiantes cuando rastrean la ejecución del código. Encontraron que no hacer esquemas se correlaciona con un menor éxito, mientras que el seguimiento de los cambios de valor de una variable escribiendo los nuevos valores cerca de su nombre a medida que cambian fue la estrategia más efectiva.

Un posible elemento de confusión que revisaron es el tiempo: dado que quienes hacen esquemas llevan significativamente más tiempo resolver los problemas, ¿lo hacen mejor solo porque piensan por más tiempo? La respuesta es no: no encontraron correlación entre el tiempo que se tomaron y el puntaje alcanzado. Nuestra recomendación es, por lo tanto, *enseñar a rastrear los valores que toman las variables al depurar errores*.

Diagramas de flujo

Un hallazgo que a menudo se pasa por alto sobre la visualización es que las personas entienden mejor los diagramas de flujo que el pseudocódigo si ambos están igualmente bien estructurados *[Scan1989]. Trabajos anteriores que muestran que el pseudocódigo supera a los diagramas de flujo usaron pseudocódigo estructurado pero diagramas de flujo desordenado; pero cuando se nivelaba el campo de juego, a las personas novatas les iba mejor con la representación gráfica.*

7.10. ¿Qué más podemos hacer para ayudar?

[Viha2014] examinó la mejora promedio en las tasas de aprobación de varios tipos de intervenciones en clases de programación. Señalan que hay muchas razones para tomar sus conclusiones con cautela: las prácticas de enseñanza previas al cambio rara vez se establecen claramente, la calidad del cambio no es juzgada, y solo el 8.3 % de los estudios reportaron resultados negativos; entonces, o hay un sesgo al reportar resultados, o la forma en la que enseñamos en este momento es la peor posible y cualquier cosa sería una mejora. Y como muchos otros estudios discutidos en este capítulo, en este trabajo sólo estaban observando clases universitarias, por lo que sus hallazgos pueden no ser generalizables a otros grupos.

[6] http://latentflip.com/loupe/

Con esas advertencias en mente, encontraron diez cosas que las/dos docentes puede hacer para mejorar los resultados (Figure 7.3):

Colaboración: Actividades que fomenten la colaboración entre las/los estudiantes, ya sea en aulas o en laboratorios.

Cambio del contenido: Se modificaron o actualizaron partes del material.

Contextualización: El contenido y las actividades del curso se alinearon con un contexto específico, como juegos o medios.

CS0: Creación de un curso preliminar a tomar antes del curso introductorio de programación; podría organizarse solo para algunas personas (por ejemplo, si están en riesgo).

Cómo jugando: Una componente de juego fue introducida en el curso.

Esquema de calificación: Un cambio en el sistema de calificación, como aumentar la importancia de las actividades de programación y reducir la del examen.

Trabajo en equipos: Actividades con un mayor compromiso de trabajo en grupo, como el aprendizaje en equipo y cooperativo.

Recursos multimedia: Actividades que usen explícitamente el uso de recursos multimedia (Chapter 10).

Apoyo de colegas: El apoyo de pares en forma de parejas, grupos, mentores contratados o tutoras/es.

Otro apoyo: Un término general para todas las actividades de apoyo, por ejemplo, mayor cantidad de horas docentes, canales de ayuda adicionales, etc.

Esta lista destaca la importancia del aprendizaje cooperativo. [Beck2013] analizó esto específicamente durante tres años académicos en cursos a cargo dos docentes diferentes y encontró beneficios significativos en general y para muchos subgrupos. Las personas que cooperaron obtuvieron calificaciones más altas y dejaron menos preguntas en blanco en el examen final, lo que indica una mayor autoeficacia y disposición para depurar cosas.

Computación sin código

Escribir código no es la única forma de enseñar a las personas a programar: hacer que las personas novatas trabajen en ejercicios de creatividad computacional mejora sus calificaciones en varios niveles [Shel2017]. Un ejercicio típico es describir un objeto cotidiano (como un clip o un cepillo de dientes) en términos de sus entradas, salidas y funciones. Este tipo de enseñanza a veces se le llama ***desconectado****; La página web CS Unplugged*[7] *tiene lecciones y ejercicios para esto.*

[7] https://csunplugged.org/en/

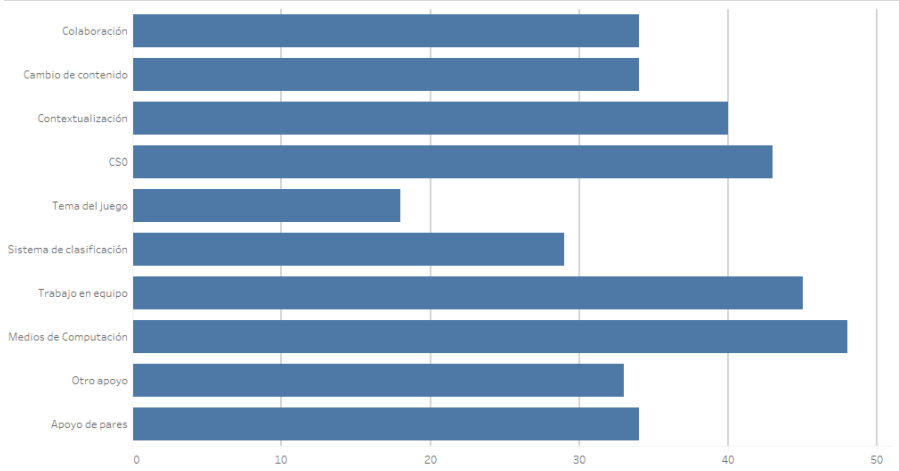

Figura 7.3: Efectividad de las intervenciones

7.11. ¿Qué sigue?

Para aquellas personas que quieran profundizar, [Finc2019] es un completo resumen de CER, [Ihan2016] resume los métodos que estos estudios usan más frecuentemente. Espero que algún día tengamos catálogos como [Ojos2015] y más materiales para capacitar docentes como [Hazz2014; Guzd2015a; Sent2018] para ayudarnos a todas y todos a enseñar mejor.

La mayor parte de la investigación mencionada en este capítulo fue financiada con fondos públicos pero tenemos que pagar para leerla: rompí la ley aproximadamente 250 veces para descargar trabajos científicos de páginas web como Sci-Hub[8] mientras escribía este libro. Espero que llegue el día en que nadie tenga que hacer esto; si investigas, por favor ayuda a que ese día se acerque publicando tu trabajo en lugares de acceso abierto.

7.12. Ejercicios

Conceptos erróneos de tus estudiantes (grupos pequeños/15')

Trabajando en grupos pequeños, vuelvan a leer la Section 7.3 y escriban una lista de conceptos erróneos que piensan que pueden tener sus estudiantes. ¿Qué tan específicos son? ¿Cómo verificarían que tan precisa es su lista?

[8] https://en.wikipedia.org/wiki/Sci-Hub

Revisando errores comunes (individual/20')

Estos errores comunes corresponden a una lista más larga en [Sirk2012]:

Asignación invertida: La/el estudiante asigna el valor del lado izquierdo de la variable al lado derecho de la variable en vez de hacerlo al revés.

Rama equivocada: La/el estudiante piensa que el código del cuerpo de un `if` corre, aun si la condición es falsa.

Ejecutar una función en vez de definirla: Las/los estudiantes creen que una función se ejecuta a medida que es definida.

Escribe un ejercicio para chequear que tus estudiantes *no* están cometiendo cada uno de estos errores.

Código desarmado (en parejas/15')

[Chen2017] describe ejercicios en los que las/los estudiantes reconstruyen el código que ha sido desarmado al eliminar comentarios, borrar o reemplazar líneas de código, moviendo líneas, etc. El rendimiento en estos ejercicios se correlaciona fuertemente con el rendimiento en las evaluaciones donde las personas deben escribir código, pero estas preguntas requieren menos trabajo para evaluar. Toma la solución de un ejercicio de programación que hayas creado en el pasado, desármalo de dos maneras distintas, intercambia el ejercicio con alguien más, y mira cuanto tiempo le lleva a cada persona responder correctamente tu problema.

El problema de la lluvia (en parejas/10')

[Solo1986] presentó el problema de la lluvia, que se ha usado en muchos estudios posteriores sobre programación [Fisl2014; Simo2013; Sepp2015]. Escribir un programa que repetidamente lee números enteros positivos hasta que llega a 99999. Después de llegar a 99999, el programa imprime el promedio de esos números leídos.

1. Resuelve el problema de la lluvia en el lenguaje de programación que prefieras.
2. Compara tu solución con las de tu colega. ¿Qué hace tu programa que no hace el de tu colega y viceversa?

Roles de variables (en parejas/15')

[Kuit2004; Byck2005; Saja2006] presentaron un conjunto de patrones de una sola variable que encuentro muy útil en la enseñanza de personas novatas:

Valor fijo: Un elemento que no toma un nuevo valor después de ser inicializado.

Paso a paso: Un elemento que pasa por una sucesión sistemática y predecible de valores.

Caminante: Un elemento que atraviesa una estructura de datos.

Elemento más reciente: Un elemento que guarda el último valor encontrado al pasar por una sucesión de valores.

Elemento más buscado: Un elemento que guarda el mejor o el valor más apropiado encontrado hasta el momento.

Recolector: Un elemento que acumula el efecto de valores individuales.

Seguidor: Un elemento que siempre obtiene su nuevo valor a partir del valor anterior de algún otro elemento.

Marca unidireccional: Un elemento de datos de dos valores que no puede obtener su valor inicial una vez que su valor se ha cambiado.

Temporal: Un elemento que contiene un valor solamente por poco tiempo.

Organizador: Una estructura de datos que guarda elementos que pueden ser reordenados.

Contenedor: Una estructura de datos que guarda elementos que pueden ser eliminados.

Elige un programa de 5 a 15 líneas y clasifica sus variables usando estas categorías. Compara tus clasificaciones con tu colega. En los casos en los que no estuvieron de acuerdo, ¿entendieron el punto de vista de la otra persona?

¿Qué estás enseñando? (individual/10')

Compara los temas que enseñas con la lista desarrollada en [Luxt2017] (Section 7.1). ¿Qué temas tratas? ¿Qué temas *no* incluyes? ¿Qué temas adicionales incluyes pero no están en la lista?

Actividades beneficiosas (individual/10')

Mira la lista de intervenciones desarrolladas por [Viha2014] (Section 7.10). ¿Cuáles de estas cosas ya haces en tus clases? ¿Cuáles podrías agregar fácilmente? ¿Cuáles son irrelevantes?

Conceptos erróneos y desafíos (grupos pequeños/15')

El sitio Desarrollo Profesional para la Enseñanza de Principios de CS[9] incluye una lista detallada de conceptos erróneos de estudiantes y ejercicios[10]. Trabajando en grupos pequeños, elige una sección (por ejemplo, estructura de datos o funciones) y revisa la lista. ¿Cuáles de estos conceptos erróneos recuerdas haber tenido cuando eras estudiante? ¿Cuáles tienes todavía? ¿Cuáles has visto en tus estudiantes?

[9] http://www.pd4cs.org/
[10] http://www.pd4cs.org/mc-index/

¿Qué es lo que más te importa? (toda la clase/15')

[Denn2019] le pidió a personas que propongan y califiquen varias preguntas de CER, y encontraron que no había superposición entre las preguntas que más les importaban a los investigadores a las que les interesaba a quienes no investigaban. Las preguntas favoritas de investigadores fueron:

1. ¿Qué conceptos fundamentales de programación son los más desafiantes para los estudiantes?
2. ¿Qué estrategias de enseñanza son más efectivas cuando se trata de una amplia gama de experiencias previas en clases introductorias de programación?
3. ¿Qué afecta la capacidad de los estudiantes para generalizar a partir de ejemplos de programación simples?
4. ¿Qué prácticas de enseñanza para enseñar computación a niñas y niños?
5. ¿Qué tipo de problemas encuentran las personas en las clases de programación más interesantes?
6. ¿Cuáles son las formas más efectivas de enseñar programación a varios grupos?
7. ¿Cuáles son las formas más efectivas de escalar la educación informática para llegar a la población general de estudiantes?

Mientras que las preguntas más importantes para las personas que no investigan fueron:

1. ¿Cómo y cuándo es mejor dar retroalimentación a las/los estudiantes sobre su código para mejorar el aprendizaje?
2. ¿Qué tipo de ejercicios de programación son más efectivos al enseñar a estudiantes de Ciencias de la Computación?
3. ¿Cuáles son los méritos relativos del aprendizaje basado en proyectos, las clases y el aprendizaje activo, para estudiantes de computación?
4. ¿Cuál es la forma más efectiva de hacer comentarios a los estudiantes en las clases de programación?
5. ¿Qué les resulta más difícil a las personas cuando dividen los problemas en tareas más pequeñas mientras programan?
6. ¿Cuáles son los conceptos clave que las/los estudiantes deben entender en las clases introductorias de computación?
7. ¿Cuáles son las formas más efectivas de desarrollar competencia en estudiantes de disciplinas no informáticas?

8. ¿Cuál es el mejor orden para enseñar conceptos y habilidades básicas en informática?

Haga que cada persona de la clase, independientemente, le otorgue un punto a las las ocho preguntas de las listas combinadas que más le interesan, luego calcule el puntaje promedio para cada pregunta. ¿Cuáles son las más populares en tu clase? ¿En qué grupo están las preguntas más populares?

8

Enseñar como un arte performativo

En *Darwin entre las máquinas*, George Dyson escribió, "En el juego de la vida y la evolución hay 3 jugadores en la mesa: los humanos, la naturaleza y las máquinas. Estoy firmemente del lado de la naturaleza. Pero la naturaleza, sospecho, está del lado de las máquinas..." De manera similar, ahora hay 3 jugadores en el juego de la educación: los libros de texto y otros materiales de lectura, las clases en vivo, y las clases en línea automatizadas. Podrías darle a tus estudiantes clases escritas y alguna combinación de videos grabados y ejercicios para que realicen a su propio ritmo, pero si vas a enseñar en persona tienes que ofrecer algo diferente (y con suerte mejor que) cualquiera de los anteriores. Por lo tanto, este capítulo se enfoca en cómo enseñar a programar programando.

8.1. Programar en vivo

> La enseñanza es teatro, no cine.
> — Neal Davis

La manera más efectiva de enseñar a programar es **programando en vivo** [Rubi2013; Haar2017; Raj2018]. En vez de presentar material previamente escrito, quien enseña escribe el código en frente de la clase mientras que los/las estudiantes lo siguen a la par, escribiendo y ejecutando el código a medida que avanzan. Programar en vivo funciona mejor que las presentaciones por varias razones:

- Permite la **enseñanza activa** al permitir a quienes están enseñando responder a los intereses y preguntas de los/las estudiantes en el momento. Una presentación de diapositivas es como una vía de ferrocarril: podrá ser un viaje suave, pero tienes que decidir hacia donde vas con anticipación. Programar en vivo es cómo manejar un vehículo todo terreno: podrá ser más accidentado, pero es mucho más fácil cambiar de dirección e ir hacia donde la gente quiere.
- Mirar como se va escribiendo un programa es más motivador que mirar a alguien pasar diapositivas.
- Facilita la transferencia de conocimiento de manera involuntaria: las personas aprenden más de lo que enseñamos conscientemente al observar *cómo* hacemos las cosas.

- Disminuye la velocidad de la persona que está enseñando: si tiene que escribir el programa medida que avanza, entonces solo puede ir el doble de rápido que sus estudiantes en vez de 10 veces más rápido como lo harían usando diapositivas.
- Ayuda a reducir la carga en la memoria de corto plazo porque hace que quien esté enseñando se más consciente de cuanto le está mostrando a sus estudiantes.
- Los/Las estudiantes pueden ver cómo diagnosticar y corregir errores. Van a dedicar mucho tiempo a esto; a menos que puedan tipear de manera perfecta, programar en vivo asegura que puedan ver como hacer esto.
- Ver a quienes enseñan cometer errores muestra a sus estudiantes que está bien que cometan errores. Si el/la docente no se avergüenza al cometer errores y habla sobre ellos, sus estudiantes también se sentirán más cómodos/as haciéndolo.

Otro beneficio de la programación en vivo es que demuestra el orden en que se deben escribir los programas. Cuando observaron cómo las personas resuelven problemas de Parson, [Ihan2011] encontraron que las personas con experiencia programando usualmente ubican la identificación del método al principio, luego agregan la mayor parte del control de flujo (es decir, bucles y condiciones), y solo luego de eso, agregan detalles como la inicialización de variables y el manejo de casos especiales. Es método "fuera de orden" es ajeno para las personas novatas, que leen y escriben código en el orden en que se presenta en la página; ver el código les ayuda a descomponer los problemas en submetas que pueden abordar una a la vez. La programación en vivo además les da quienes están enseñando la chance de enfatizar la importancia de los pequeños pasos con comentarios frecuentes [Blik2014] y la importancia de definir un plan en vez de hacer cambios más o menos aleatorios y esperar que las cosas mejoren [Spoh1985].

Sentirse cómodo/a al hablar mientras se escribe código en frente de una audiencia, requiere práctica, pero la mayoría de las personas indican que rápidamente se vuelve igual que difícil que hablar alrededor de una presentación de diapositivas. Las secciones que siguen ofrecen consejos sobre cómo mejorar la manera de programar en vivo.

Aprovecha tus errores

> Los errores de tipeo son la pedagogía.
> — Emily Jane McTavish

La regla más importante de la programación en vivo es aprovechar tus errores. No importa que tan bien te prepares, cometerás algunos errores; cuando lo hagas, piensa sobre ellos con tu audiencia. Si bien obtener los datos es difícil, programadores/as profesionales dedican del 25 % al 60 % de su tiempo identificando y resolviendo errores; las personas novatas dedican le dedican mucho más (Section 7.6), pero la mayoría de los libros de texto y tutoriales dedican poco tiempo a diagnosticar y corregir problemas. Si hablas en voz alta mientras intentas identificar que escribiste mal o dónde tomaste el camino equivocado, y explicas cómo lo corriges, les darás a

tus estudiantes un conjunto de herramientas que pueden usar cuando comentan sus propios errores.

Tropiezos deliberados

Una vez que hayas enseñado una lección varias veces, es poco probable que cometas nada más que errores básicos de tipeo (que de todas maneras pueden ser informativos). Puede intentar recordar errores pasados y cometerlos deliberadamente, pero usualmente eso se siente forzado. Un enfoque alternativo es ***sacudir la programación****: pide a tus estudiantes, uno a uno que te indiquen que escribir a continuación. Esto prácticamente garantiza que te encuentres en algún tipo de problemas.*

Pregunta por predicciones

Una manera de mantener a tus estudiantes motivados/as mientras estas programando en vivo es pedirles que hagan predicciones sobre que hará el código que ven en la pantalla. Luego, puedes escribir las primeras sugerencias que hagan, hacer que toda la clase vote sobre cual piensan que es la opción más probable, y finalmente ejecutar el código. Esta es una forma simple de instrucción entre pares, que discutiremos en la sección Section 9.2; además de mantener su atención en la actividad, les permite practicar cómo razona sobre el comportamiento del código.

Tomalo con calma

Cada vez que escribas un comando, agregues una lidera de código a un programa, o selecciones un elemento de un menú, di que estas haciendo en voz alta, luego señala lo que haz hecho y su resultado en la pantalla y repasalo una segunda vez. Esto ayuda a tus estudiantes a ponerse al día y a revisar que lo que acaban de hacer es correcto. Esto es particularmente importante cuando algunos de tus estudiantes tienen dificultades para ver o escuchar o no dominan el idioma en el que estás enseñando.

Hagas lo que hagas, *no* copies y pegues código: hacer eso prácticamente garantiza que iras mucho más rápido que tus estudiantes. Y si usas la tecla tab para autocompletar lo que estás escribiendo, decilo en vos alta para que tus estudiantes entiendan lo que estás haciendo: “Usemos turtle dot ‘r’ ‘i’ y tab para completar con ‘right’.”

Si la salida de tu comando o código hace que lo que acabas de escribir desaparezca de la vista, vuelve arriba para que tus estudiantes puedan verlo de nuevo. Si eso no es posible, ejecuta el mismo comando una segunda vez o copia y pega el último comando o comandos en las notas compartidas del taller.

Ser visto/a y escuchado/a

Cuando te sientas, es más probable que mires tu pantalla en vez de mirar a tu audiencia y puedes quedar fuera de la vista de tus estudiantes en las últimas filas del aula. Si eres físicamente capaz de pararte durante un par de horas, debes hacerlo mientras enseñas. Planifícalo y asegúrate de tener una mesa elevada, un escritorio

de pie, o un atril para tu computadora portátil para que no tengas que inclinarte al escribir.

Independientemente de si estás de pie o sentado/a, asegúrate de moverte lo más que puedas: acércate a la pantalla para señalar algo, dibuja algo en la pizarra, o simplemente aléjate de la computadora por un momento y háblale directamente a tu audiencia. Hacer esto aleja la atención de tus estudiantes de sus pantallas y les proporciona un momento natural para hacer preguntas.

Si vas a enseñar por más de un par de horas, vale la pena usar un micrófono incluso si la habitación es pequeña. Tu garganta se cansa tanto como cualquier otra parte de tu cuerpo; usar un micrófono no es diferente de usar zapatos cómodos (algo que también deberías usar). También puede marcar una gran diferencia para las personas que tienen discapacidad auditiva.

Copia la pantalla de tu estudiante

Es posible que hayas personalizado tu entorno de trabajo con una terminal de Unix shell elegante, un esquema de colores personalizado, o una gran cantidad de atajos de teclado. Tu estudiantes no tendrán nada de eso, así que intenta crear un entorno de trabajo que refleje lo que *sí* tienen. Algunos/as docentes crean un usuario distinto con configuración básica en sus computadoras o una cuenta específica para enseñar si están usando algún servicio online como Scratch o GitHub. Hacer esto también puede ayudar a evitar que los paquetes que instalaste ayer para trabajar rompan la lección que se supone que enseñes hoy.

Usa la pantalla de manera sabia

Por lo general, necesitaras agrandar el tamaño de la letra considerablemente para que las personas en el fondo de la sala puedan leer. Esto significa que podrás colocar muchas menos cosas en la pantalla de las que estás acostumbrado/a. En muchos casos, se reducirá a 60–70 columnas y 20–30 filas, por lo que estarás usando una super computadora del siglo 21 como si fuera una sencilla terminal de principios de la década de 1980.

Para organizar esto, maximiza la ventana que estás usando para enseñar y luego preguntale a todos si pueden leer lo que están en la pantalla o no. Usa una fuente de color negro sobre un fondo ligeramente coloreado en vez de una fuente de color claro sobre un fondo oscuro—el tono claro deslubrará menos que el blanco puro.

Presta atención a la iluminación de la sala: no debe estar completamente a oscuras, y no debe haber luces directamente o por encima de la pantalla de protección. Dedica algunos minitos para que tus estudiantes puedan reacomodar sus mesas para ver con claridad.

Cuando la parte inferior de la proyección de la pantalla está a la misma altura que las cabezas de tus estudiantes, las personas en el fondo no podrán ver lo que ocurre en esa sección de la pantalla. Puede elevar la parte inferior de la ventana para compensar esto, pero eso generará que tengas aún menos espacio para escribir.

Si puedes acceder a un segundo proyector y pantalla, úsalos: el espacio adicional

te permitirá mostrar el código de un lado y su resultado o comportamiento del otro lado. Si la segunda pantalla requiere su propia computadora, pídele a un/a ayudante que la controle en lugar de ir y venir entre los dos teclados.

Finalmente, si estás enseñando algo como la terminal de Unix shell en una consola, es importante decirle a las personas usas un editor de texto en la consola y cuando regresas a la consola propiamente dicha. La mayoría de las personas novatas no han visto nunca una ventana asumir múltiples personalidades de esta manera, y pueden confundirse rápidamente cuando estás interactuando en la terminal, cuando estás escribiendo en un editor de texto, y cuando estás trabajando de manera interactiva con Python u otro lenguaje. Puedes evitar este problema usando ventanas separadas para usar el editor de texto; si haces esto, siempre avísales a tus estudiantes cuando estás cambiando de una ventana a la otra.

Las herramientas de accesibilidad ayudan a todas las personas

Las herramientas como Mouseposé[1] (para Mac) y PointerFocus[2] (para Windows) resaltan la posición del cursor del mouse en la pantalla, y las herramientas de grabación de pantalla como Camtasia[3] y aplicaciones independientes como KeyCastr[4] muestran teclas invisibles como tab y Control-J a medida que las usas. Esto puede ser un poco molesto al comienzo, pero ayuda a tus estudiantes a descubrir lo que estás haciendo.

Dos dispositivos

Algunas personas usan dos dispositivos cuando enseñan: una computadora portátil conectada al proyector que los/as estudiantes vean y una tablet para que puedan ver sus propias notas y las notas que los/as estudiantes están tomando (Section 9.7). Esto es más confiable que pasar de un escritorio virtual al otro, aunque imprimir la lección sigue siendo la tecnología de respaldo más confiable.

Dibuja temprano, dibuja seguido

Los diagramas son siempre una buena idea. A veces tengo una presentación de diapositivas llena de diagramas preparada de antemano, pero construir los diagramas paso a paso ayuda a retenerlos más (Section 4.1) y te permite improvisar.

Evita las distracciones

Desactiva las notificaciones que usas en tu computadora, especialmente las de redes sociales. Ver mensajes parpadeando en la pantalla te distrae a vos y a tus estudiantes, y puede ser incómodo si aparece un mensaje que no te gustaría que otras

[1] https://boinx.com/mousepose/overview/
[2] http://www.pointerfocus.com/
[3] https://www.techsmith.com/video-editor.html
[4] https://github.com/keycastr/keycastr

personas vean. De nuevo, es posible que quieras crear una segunda cuenta en tu computadora que no tenga correo electrónico u otras herramientas configuradas.

Improvisa—luego de haber aprendido el material

No te alejes de la lección que planificaste o pediste prestada la primera vez que la enseñes. Puede ser tentador desviarse del material porque te gustaría mostrar un lindo truco o demostrar otra manera de hacer algo, pero existe la posibilidad de que te encuentres con algo inesperado que te lleve más tiempo del que tenés.

Sin embargo, una vez que el material te resulte más familiar, puedes y debes comenzar a improvisar en base a los antecedentes de tus estudiantes, sus preguntas durante la clase, y lo que personalmente te parezca más interesante. Esto es como tocar una nueva canción: sigues la partitura las primeras veces, pero después de que te sientes cómodo/a con los cambios de melodía y acordes, puedes comenzar a ponerle tu propio sello.

Cuando quieras usar algo nuevo, revísalo de antemano *usando la misma computadora que usarás cuando des la clase*: instalar cientos de megabytes de programas a través del WiFi de la escuela secundaria en frente de jóvenes de 16 años aburridos no es algo por lo que alguna vez quieras pasar.

Enseñanza directa

*La **Enseñanza directa** (ED) es un método de enseñanza centrando en el diseño meticuloso de la currícula dictado usando un guío predefinido. Es más como un actor recitando lineas que como el enfoque de improvisación que recomendamos. [Stoc2018] encontró que la ED tiene un efecto estadísticamente significativo positivo a pesar de que a veces pueda ser muy repetitivo. Yo prefiero improvisar porque la ED requiere más preparación inicial que lo que la mayoría de los grupos de estudiantes free-range pueden permitirse.*

Mira a la pantalla—de vez en cuando

Está bien enfrentar la pantalla donde estás proyectando ocasionalmente cuando estás mostrando una sección de código o dibujando un esquema: *no* mirar a la sala llena de personas que te están mirando a vos puede ayudarte a reducir tu nivel de ansiedad y darte un momento para pensar qué decir a continuación.

Sin embargo, no deberías acertó por más de unos segundos. Una buena regla general es tratar a la pantalla como a uno/a de tus estudiantes: si mirar a una persona durante el tiempo que miras a la pantalla te resulta incómodo, es hora de darte la vuelta y mirar a la clase nuevamente.

Inconvenientes

Programar en vivo tiene algunos inconvenientes, pero pueden evitarse o solucionarse con un poco de práctica. Si descubres que están cometiendo demasiados errores

de tipeo, reserva 5 minutos por día para practicar escribir con el teclado: también te ayudará en tu trabajo diario. Si crees que dependes demasiado de las notas de la clase, divídelas en partes más pequeñas para que solo tengas que pensar en un pequeño paso a la vez.

Y si sientes que estás pasando demasiado tiempo escribiendo código para importar librerías, encabezados de clases y código repetitivo, genera un esqueleto de código para que vos y tus estudiantes usen como punto de partida (Section 9.9). Hacer esto también reducirá su carga cognitiva, dado que centrarán su atención donde vos quieres.

8.2. Estudiar la lección

Desde políticos hasta investigador/as y docentes, quienes reforman la educación han diseñado sistemas para encontrar y apoyar a personas que pueden enseñar bien y eliminar a las personas no lo hacen. Pero la suposición de que algunas personas nacen como docentes es errónea: en cambio, como cualquier otra representación artística, las claves para enseñar mejor son práctica y colaboración. Cómo explica [Gree2014], En japonés este enfoque se llama **jugyokenkyu**, que significa "estudiar la lección":

> Para graduarse, los especialistas en educación [japoneses] no solo tenían que ver como trabaja el docente que le asignan, tenían que reemplazarlo efectivamente, participando en su aula primero como observadores/as y luego, a la tercera semana, como una aproximación... titubeante del propio maestro. Funcionó como una especie de relevo de docentes. Cada estudiante eligió una asignatura, preparando clases para 5 días... [y luego] cada uno enseñó un día. Para pasar la batuta, tenía que enseñar una clase de un día en cada asignatura: la que tenían planeada y las 4 que no... y tenían que hacerlo delante de su maestro. Después, todos—el docente, los estudiantes para ser docentes, y a veces, incluso un observador externo—se sentaban alrededor de una mesa para hablar de lo que observaron.

Poner el trabajo bajo un microscopio para mejorarlo es común en áreas tan diversas como la fabricación[5] y la musica. Un/a músico/o profesional, por ejemplo, analizará media docena de grabaciones de "Body and Soul" o "Smells Like Teen Spirit" antes de interpretarlas. También recibirán comentarios de colegas músicos durante la práctica y después de las actuaciones.

Pero la retroalimentación continua no es parte de la cultura de la enseñanza en la mayoría de los países de habla inglesa. Allí, lo que sucede en el aula se queda en el aula: quienes enseñan no miran las clases de sus colegas de manera regular, por lo que no pueden tomar prestadas las buenas ideas de las demás personas. Los/as docentes podrán acceder a los planes de clases y tareas de otros colegas, la junta

[5] https://en.wikipedia.org/wiki/W._Edwards_Deming

escolar o una editorial de libros de texto, o revisar MOOCs en Internet, pero cada persona tiene que descubrir como dar las clases específicas en aulas específicas para estudiantes específicos/as. Esto es particularmente cierto para personas voluntarias y docentes free-range que participan en talleres y actividades fuera de la escuela.

Desarrollar nuevas técnicas y dar **lecciones de demostración** (en las que una persona enseña a estudiantes reales mientras otras personas observan) no son la solución. Por ejemplo, [Finc2007; Finc2012] encontraron que de los 99 historiales analizados, quienes enseñan solo buscaron activamente nuevas prácticas o materiales en 3 casos, y solo consultaron material publicado en 8. La mayoría de los cambios se dieron localmente, sin aportes de fuentes externas, o solo involucraron comunicación personal con otros/as docentes. [Bark2015] encontró algo similar:

> La adopción no es una "acción racional"... sino una serie iterativa de decisiones tomadas en un contexto social, basado en tradiciones normativas, señales sociales, y procesos emocionales o intuitivos... No es probable que profesores utilicen los resultados de investigaciones en educación como base para tomar decisiones... La retroalimentación positiva de los/as estudiantes se toma como fuerte evidencia por parte de los/las docentes de que deben continuar con una práctica.

Jugyokenkyu funciona porque maximiza la oportunidad de transferencia de conocimiento no planificada entre docentes: alguien se propone demostrar X, pero mientras lo miran, su audiencia también (o en su lugar) aprende Y. Por ejemplo, quien enseña podría tener la intención de demostrar a sus estudiantes como buscar direcciones de correo electrónico en un archivo de texto, pero lo que su audiencia podría terminar aprendiendo son algunos atajos de teclado.

8.3. Dando y recibiendo retroalimentación al enseñar

Observar a alguien te ayuda, y darle una devolución ayuda a esa persona, pero puede ser difícil recibir devoluciones, especialmente cuando son negativas (Figure 8.1).

La retroalimentación es más fácil de dar y recibir cuando ambas partes comparten expectativas sobre lo que está y no está al alcance y sobre cómo se deben expresar los comentarios. Si solicitas una devolución:

Inicia la devolución. Es mejor pedir la retroalimentación que recibirla de de mala gana.

Elige tus preguntas, es decir, pide comentarios específicos. Es mucho más difícil para alguien responder, "¿Qué te pareció?" que responder, "¿Hablé demasiado rápido?" o , "¿Qué cosa de esta lección debería seguir haciendo?" Direccionar la devolución de esta manera además es más útil para vos. Siempre es mejor de mejorar una cosa a la vez que cambiar todo y esperar que sea para mejor. Direccionar

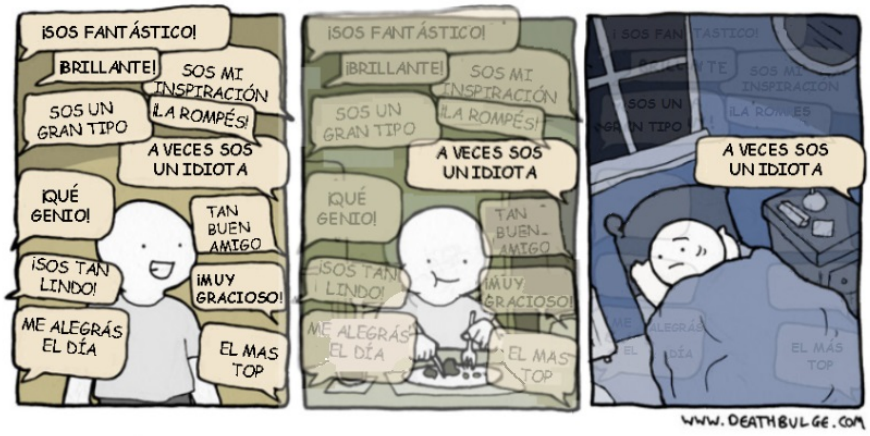

Figura 8.1: Feedback feelings (copyright © Deathbulge 2013)

los comentarios hacia algo que elegiste trabajar te ayuda a mantenerte enfocado/a, lo que a su vez aumenta las chances de que veas un progreso.

Usa un traductor de retroalimentación. Pídele a alguien que lea todas las devoluciones y te haga un resumen. Puede ser más fácil escuchar, "Varias personas piensan que podrías acelerar un poco," que leer varias notas que digan, "Esto es demasiado lento" o, "Esto es aburrido."

Se amable con vos. Muchos/as de nosotros/as somos muy críticos/as de nosotros/as mismos/as, por lo que siempre es útil anotar lo que pensamos de nosotros/as *antes* de recibir retroalimentación de otras personas. Eso nos permite comprar lo que pensamos de nuestro desempeño con lo que otras personas piensan, lo que a su vez nos permite evaluar esto último con mayor precisión. Por ejemplo, es muy común que las personas piensen que están diciendo "mmm" and "ehh" con demasiada frecuencia cuando tu audiencia en realidad no lo nota. Recibir esa devolución una vez permite a los/as docentes ajustar la evaluación sobre si mismos/As la próxima vez que se sientan así.

También puedes dar retroalimentación a otras personas de manera más efectiva:

Interactúa. Mirar fijamente a alguien es una buena manera de hacer que se sienta incómodo/a, por lo que si deseas darle una devolución sobre como enseña normalmente, necesitas tranquilizarlo/a. Interactuar con la persona como si fueras estudiante es una buena manera de hacer esto, así que haz preguntas o (pretende) escribir su ejemplo. Si sos parte de un grupo, haz que una o dos personas desempeñen el papel de estudiantes mientras que el resto toma notas.

Balancea la retroalimentación positiva y negativa. El "sándwich de cumplidos" compuesto por un comentario positivo, uno negativo, y un segundo positivo se

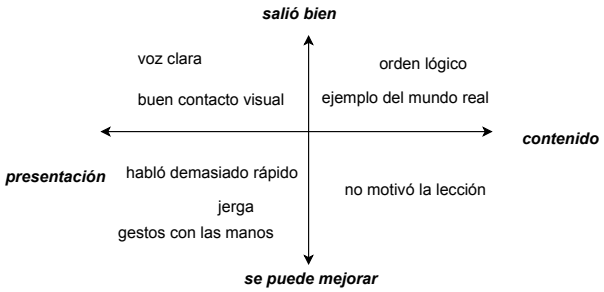

Figura 8.2: Rubrica de enseñanza

vuelve cansador rápidamente, pero es importante decirle a las personas qué deben seguir haciendo y qué deben cambiar[6].

Toma notas. No vas a recordar todo lo que notaste si la presentación dura más de unos pocos segundos, y definitivamente no vas a recordar con qué frecuencia lo notaste. Escribe una nota la primera vez que algo sucede y luego agrega una marca o cruz cuando vuelva a ocurrir para que puedas ordenar tus comentarios por frecuencia.

Tomar notas es más eficiente cuando tienes algún tipo de rúbrica para que tengas que apurarte a escribir tus observaciones mientras la persona que estás observando todavía está hablando. La rúbrica más simple para dar comentarios de forma libre en grupo es la grilla de 2x2 cuyo eje vertical tiene las etiquetas “lo que salió bien” y “lo que se puede mejorar”, y en el eje horizontal “contenido” (lo que se dijo) y “presentación” (como se dijo). Quienes observan escriben sus comentarios en notas autoadhesivas mientras miran la demostración, luego las pegan en los cuadrantes de la grilla dibujada en una pizarra (Figure 8.2).

Rúbricas e inventario de preguntas

La Section F.1 contiene una rúbrica de ejemplo para evaluar 5–10 minutos de enseñanza de programación. Presenta elementos más o menos en el orden en que es probable que aparezcan, por ejemplo, preguntas sobe la introducción aparecen antes que las preguntas sobre la conclusión.

Rúbricas como esta tienden a crecer con el tiempo a medida que las personas piensan en cosas les gustaría agregar. Una buena manera de mantener las rúbricas manejables es insistir en que la longitud total permanezca constante: si alguien quiere agregar una pregunta, debe identificar una que sea menos importante y que pueda sacarse.

Si te interesa dar y recibir retroalimentación, [Gorm2014] tiene buenos consejos que puedes usar para hacer que la retroalimentación entre pares sea parte del proceso de enseñanza, mientras que [Gawa2011] analiza el valor de tener un/a tutor/a.

[6] Por un tiempo, me preocupe tanto por la afinación que perdí completamente mi sentido del tiempo

Clases de estudio

Las escuelas de arquitectura a menudo incluyen clases de estudio en las que estudiantes resuelven pequeños problemas de diseño y reciben devoluciones de sus pares en ese mismo momento. Estas cases son más efectivas cuando el/la docente evalúa las devoluciones de pares para que quienes participan aprendan no solo cómo construir edificios sino también como dar y recibir retroalimentación [Scho1984]. Las clases magistrales de música tienen un propósito similar, y he descubierto que dar retroalimentación sobre la retroalimentación ayuda a las personas a mejorar su manera de enseñar también.

8.4. Cómo practicar como enseñamos

La mejor manera de mejorar la manera que damos clases en persona es observarse a si mismo hacerlo:

- Trabaja en grupos de a tres personas.
- Cada persona rota entre los roles de docente, audiencia y quien graba. La persona en el rol docente tiene 2 minutos para explicar algo. La persona que pretende ser la audiencia está allí para prestar atención, mientras que la persona la tercera persona graba la sesión con un teléfono o otro dispositivo portátil.
- Luego de que todas las personas tuvieron la oportunidad de enseñar, ¿el grupo mira todos los videos. Cada persona da una retroalimentación sobre los 3 videos, es decir, las personas dan una devolución sobre si mismas y sobre las demás.
- Después de que se discutieron los videos, se borran. (Muchas personas se sienten justificadamente incómodas por su en Internet.)
- Finalmente, toda la clase vuelve a reunirse y agrega las devoluciones a una grilla 2x2 compartida como la que se describió previamente *sin decir* de quién se trata cada comentario.

Para que este ejercicio funcione bien:

- Graben los 3 videos y luego miren los 3. Si el ciclo es enseñar-revisar-enseñar-revisar, la última persona en enseñar siempre se queda sin tiempo (a veces a propósito). Hacer todas las revisiones luego de que todas las personas enseñaron también ayuda a poner un poco de distancia entre los dos momentos, lo que hace que el ejercicio sea un poco menos incómodo.
- Avísales a las personas al comienzo de la clase que les pedirás que enseñen algo para que tengan tiempo de elegir un tema. Avisarles con mucha anticipación puede ser contraproducente, ya que algunas personas se preocuparan por cuán deben prepararse.

- Los grupos deben estar físicamente separados para reducir el ruido en sus grabaciones. En la práctica, esto significa 2–3 grupos en un aula de tamaño normal, y el resto usando espacios de descanso cercanos, oficinas o (en una ocasión) el armario de conserje.
- Las personas deben dar retroalimentación sobre sí mismas y entre sí para que puedan calibrar sus impresiones de la manera en que enseñan contra las de otras personas. La mayoría de las personas son más críticas sobre ellas mismas de lo que deberían ser, y es importante que se den cuenta de esto.

El anuncio de este ejercicio usualmente es recibido con quejidos y aprensión, ya que pocas personas disfrutan verse o escucharse a sí mismas. Sin embargo, esas mismas personas lo califican constantemente como un de las partes más valiosas de los talleres de enseñanza. También es una buena preparación para enseñar de a pares (Section 9.3): a las personas que enseñan les resulta mucho más fácil intercambiar devoluciones informales si han tenido algo de práctica y tienen una rúbrica compartida para definir expectativas.

Y hablando de rúbricas: una vez que la clase haya puesto todos sus comentarios en una grilla compartida, elige algunos comentarios positivos y negativos. haz una lista, y pídeles que hagan el ejercicio nuevamente. La mayoría de las personas se sienten más cómodas la segunda vez, y la evaluación sobre lo que han decidido que es importante aumenta su sentido de autodeterminación (Chapter 10).

Tics

Todas las personas tenemos hábitos nerviosos: hablamos más rápido y con voz más aguda de lo normal cuando estamos en el escenario, jugamos con nuestro pelo, o hacemos sonar los nudillos. Las personas que juegan llaman a esto "tics," y las personas a menudo no se dan cuenta de que se mueven, se miran los zapatos, o juegan con lo que tienen en los bolsillos cuando en realidad no saben la respuesta a una pregunta.

No puedes deshacerte de los tics completamente, e intentar hacerlo puede hacer que te obsesiones con ellos. Una mejor estrategia es tratar de desplazarlos, por ejemplo, entrenarse para apretar los dedos de los pies dentro de los zapatos cuando tienes nervios en vez de limpiarte la oreja con el dedo meñique.

8.5. Ejercicios

Dar devolución sobre una mala enseñanza (toda la clase/20)

En grupo, miren este video de mala enseñanza[7] y den retroalimentación sobre dos ejes: positivo versus negativo y contenido versus presentación. Que cada persona en

[7] https://www.youtube.com/watch?v=-ApVt04rB4U

la clase agregue un comentario a la grilla 2x2 en una pizarra en las notas compartidas sin duplicar ningún comentario. ¿Qué vieron otras personas y vos no? ¿Con qué comentarios estás totalmente de acuerdo o en desacuerdo?

Practicar dar devoluciones (grupos pequeños/45)

Usa el proceso descripto en la Section 8.4 para enseñar en grupos de 3 y recolectar las devoluciones.

Lo malo y lo bueno (toda la clase/20)

Mira los videos de programación en vivo mal desarrollada[8] y bien desarrollada[9] y resume tu devolución sobre las diferencias usando la grilla 2x2 habitual. ¿De qué manera la segunda sesión de enseñanza es mejor que la primera? ¿Hay algo que es mejor en el primer video que en el segundo?

Observa, luego hace (de a pares/30)

Enseña 3–4 minutos de una lección usando programación en vivo a un/a colega, luego intercambia lugares y observa a esa persona programar en vivo. No te preocupes por grabar estas sesiones—es difícil capturar tanto a la persona como a la pantalla con un dispositivo portátil—pero da una devolución de la misma manera que lo hiciste antes. Explícales a tus colegas que vas a enseñar y con qué esperas que estén familiarizados/as.

- ¿Qué se siente diferente entre programar en vivo y dar una clase tradicional? ¿Cuál fue más fácil y más difícil?
- ¿Cometiste algún error? Si lo hiciste, ¿cómo lo manejaste?
- ¿Hablaste y escribiste al mismo tiempo o alternativamente?
- ¿Con qué frecuencia señalaste algo en la pantalla? ¿Con qué frecuencia usaste el cursor para resaltar algo?
- ¿Qué intentarás seguir haciendo la próxima vez? ¿Qué intentarás hacer diferente?

Tics (grupos pequeños/15)

1. Toma notas sobre los tics que piensas que tienes, pero no las compartas con otras personas.
2. Enseña una clase corta (de 2–3 minutos).
3. Preguntale a tu audiencia cómo creen que te traicionan los nervios. ¿Coinciden con lo que pensaste de vos?

[8] https://youtu.be/bXxBeNkKmJE
[9] https://youtu.be/SkPmwe_WjeY

Consejos para enseñar (grupos pequeños/15)

El sitio CS Teaching Tips[10] tiene una gran cantidad de consejos prácticos sobre como enseñar computación, así como una colección de hojas de consejos descargables. Revisa las hojas de consejos en grupos pequeños y clasifica cada uno de acuerdo a si lo usas todo el tiempo, ocasionalmente, o nunca lo usaste. ¿En qué se diferencia tu práctica y la práctica de tus compañeros? ¿Hay algún consejo con el que no estés de acuerdo o creas que sería ineficaz?

Resumen

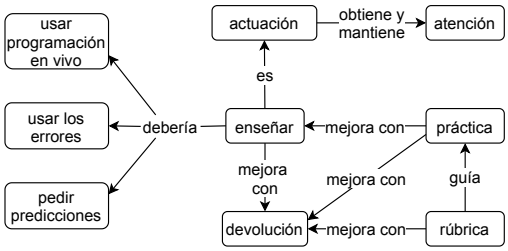

Figura 8.3: Conceptos: Retroalimentación

[10] http://csteachingtips.org/

9

En el salón de clase

El capítulo anterior describió cómo practicar la entrega de lecciones y describió un método — live coding — que permite a las/los educadores adaptarse al ritmo y los intereses de sus estudiantes. Este capítulo describe otras prácticas que también son útiles en clases de programación.

Antes de describirlos, vale la pena detenerse un momento para establecer expectativas. El mejor método de enseñanza que conocemos es la tutoría individual: [Bloo1984] descubrió que los/las estudiantes a quienes se les enseñó uno a uno hicieron dos desviaciones estándar mejor que los/las que aprendieron mediante una clase convencional, es decir que los/las estudiantes con tutoría individual superaron al 98 % de estudiantes a los que se les dio clases. Aunque, si bien la tutoría y el aprendizaje fueron las formas más comunes de transmitir conocimientos a lo largo de la mayor parte de la historia, la industrialización de la educación formal la ha convertido en la excepción en la actualidad. A pesar de la explosión en torno a la inteligencia artificial, esto no va a cerrar este círculo pronto, por lo que cada método que se describe a continuación es esencialmente un intento de abordar la efectividad de la tutoría individual a escala.

9.1. Hacer cumplir el código de conducta

Lo más importante que he aprendido sobre la enseñanza en los últimos 30 años es lo importante que es para todos/as tratar a los/las demás con respeto, tanto dentro como fuera de clase. Si usas este material de alguna manera, por favor adopta un Código de Conducta como el de Appendix B y exige a todos/as los/las que participan en tus clases que lo respeten. No puedes evitar que las personas sean ofensivas, como tampoco las leyes contra el robo evitan que las personas roben, pero *puedes* aclarar las expectativas y las consecuencias, y señalar que estás tratando de que tu clase sea acogedora para todos/as.

Pero un Código de Conducta sólo es útil si se hace cumplir. Si crees que alguien ha violado el tuyo, puedes advertirles, pedirles que se disculpen, y / o expulsarlos/las, dependiendo de la gravedad de la infracción y si crees o no que fue intencional. Hagas lo que hagas:

Hazlo frente a testigos/as. La mayoría de las personas bajarán el tono de su lenguaje y hostilidad frente a una audiencia, y tener a alguien más presente asegura

que la discusión posterior no degenere en afirmaciones contradictorias sobre quién dijo qué.

Si expulsas a alguien, comunícalo al resto de la clase y explica por qué. Esto ayuda a evitar que los rumores se difundan y muestra que tu Código de Conducta realmente significa algo.

Envía un correo electrónico a la persona infractora tan pronto como puedas para resumir lo que sucedió y los pasos que tomaste, y copia el mensaje a los/las anfitriones/as del taller o a alguno de tus colegas educadores para que haya un registro contemporáneo de la conversación. Si la persona infractora responde, no participes en un debate largo: nunca es productivo.

Lo que sucede fuera de la clase importa al menos tanto como lo que sucede dentro de ella [Part2011], por lo que debes proporcionar una forma para que los/las estudiantes informen sobre los problemas que tú mismo no puedes ver. Un paso es pedirle a alguien que no sea parte de tu grupo que sea el primer punto de contacto; de esta manera, si alguien quiere presentar una queja sobre ti o alguno de tus colegas educadores, tiene cierta garantía de confidencialidad y acción independiente. [Auro2019] tiene muchos otros consejos y es a la vez breve y práctico.

9.2. Instrucción de pares

No importa lo buena que sea una persona que enseña, solo puede decir una cosa a la vez. Entonces, ¿cómo puede aclarar muchos conceptos erróneos diferentes en un tiempo razonable? La mejor solución desarrollada hasta ahora es una técnica llamada **instrucción de pares**. Originalmete creada por Eric Mazur en Harvard [Mazu1996], se ha estudiado extensamente en una amplia variedad de contextos, incluida la programación [Crou2001; Port2013], y [Port2016] descubrió que los/las estudiantes valoran la instrucción de sus pares incluso en el primer contacto.

La instrucción entre pares intenta proporcionar instrucción individualizada de manera escalable intercalando la evaluación formativa con la discusión del estudiante:

1. Haz una breve introducción al tema.
2. Ofrece a los estudiantes una pregunta de opción múltiple que investigue sus conceptos erróneos. (en lugar de probar el simple conocimiento de hechos).
3. Haz que todos los estudiantes voten por sus respuestas al MCQ.
    - Si todos los estudiantes tienen la respuesta correcta, continúa.
    - Si todos tienen la misma respuesta incorrecta, aborda ese error específico.

- Si tienen una combinación de respuestas correctas e incorrectas, Dales varios minutos para discutir entre ellos en grupos de 2 a 4, luego que vuelvan a votar.

Asi como este video[1] muestra, la discusión en grupo mejora significativamente la comprensión de los estudiantes porque descubre lagunas en su razonamiento y los obliga a aclarar su pensamiento. Volver a sondear a la clase le permite al maestro saber si pueden seguir adelante o si se necesitan más explicaciones. Una ronda final de explicación adicional después de que se presenta la respuesta correcta les da a los estudiantes una oportunidad más para solidificar su comprensión.

Pero, ¿podría ser esto un falso positivo? ¿Los resultados están mejorando debido a una mayor comprensión durante la discusión o simplemente por un efecto de seguimiento del líder ("vota como Jane, ella siempre tiene la razón")? [Smit2009] probó esto siguiendo la primera pregunta con una segunda que los estudiantes respondieron individualmente. Descubrieron que la discusión entre pares mejora la comprensión, incluso cuando ninguno de los estudiantes de un grupo de discusión sabía originalmente la respuesta correcta. Siempre que exista diversidad de opiniones dentro del grupo, sus conceptos erróneos se anulan.

[1] https://www.youtube.com/watch?v=2LbuoxAy56o

Tomar una posición

Es importante que tus estudiantes voten públicamente para que no puedan cambiar de opinión después y racionalizarlo con excusas como: "Acabo de interpretar mal la pregunta." Gran parte del valor de la instrucción entre pares proviene de la hipercorrección de obtener la respuesta incorrecta y tener que pensar en las razones del porque de esta respuesta (Section 5.1).

9.3. Teach Together

Co-enseñar describe cualquier situación en la que dos educadores trabajan juntos en la misma clase. [Frie2016] describe muchas formas de hacer esto:

Enseñar en equipo: Ambos educadores entregan un único flujo de contenido en conjunto, turnándose como músicos haciendo solos.

Enseñar y ayudar: El educador A enseña mientras el educador B se mueve por el aula para ayudar a los estudiantes con dificultades.

Enseñanza alternativa: El educador A proporciona a un pequeño grupo de estudiantes una instrucción más intensiva o especializada mientras que el educador B ofrece una lección general al grupo principal.

Enseñar y observar: El educador A enseña mientras el educador B observa a los estudiantes y recopila datos sobre su comprensión para ayudar a planificar lecciones futuras.

Enseñanza paralela: La clase se divide en dos y los educadores presentan el mismo material simultáneamente a cada grupo.

Station teaching: Los estudiantes se dividen en pequeños grupos que rotan de una estación o actividad a la siguiente mientras los educadores supervisan donde sea necesario.

Todos estos modelos crean más oportunidades para la transferencia involuntaria de conocimientos que la enseñanza por sí sola. Enseñar en equipo es particularmente beneficiosa en talleres de un día: le da a la voz de cada educador la oportunidad de descansar y reduce el riesgo de que estén tan cansados al final del día que comiencen a hablar bruscamente con sus estudiantes o a manipular el teclado.

Ayudar

Muchas personas que no se sienten cómodas enseñando están dispuestas y son capaces de brindar asistencia técnica en clase. Pueden ayudar a los estudiantes con la configuración e instalación, responder preguntas técnicas durante los ejercicios, supervisar la sala para detectar personas que puedan necesitar ayuda o poner atención a las notas compartidas.

(Section 9.7), y responder preguntas o recuerdar al educador que lo haga durante los descansos.

Los ayudantes a veces son personas que se están capacitando para convertirse en educadores (es decir, son el educador B en el modelo de enseñar y apoyar), pero también pueden ser miembros del personal de apoyo técnico de la institución anfitriona, exestudiantes de la clase o estudiantes avanzados que ya conocen el material bien. Usar a estos últimos como ayudantes es doblemente efectivo: no solo es más probable que comprendan los problemas que tienen sus compañeros, sino que también evita que se aburran. Esto ayuda a que toda la clase se mantenga comprometida porque el aburrimiento es contagioso: si un puñado de personas comienza a pagar, las personas que los rodean seguirán su ejemplo.

Si tu y un compañero están co-enseñando:

- Toma de 2 a 3 minutos antes del comienzo de cada clase para confirmar quién está enseñando qué. Si tienen tiempo, intenten dibujar o revisar juntos un mapa conceptual.
- Usa ese tiempo para hacer también un par de señales con las manos. " Vas demasiado rápido ", " habla ", " ese estudiante necesita ayuda " y " es hora de ir al baño " son todos útiles.
- Cada persona debe enseñar durante al menos 10 a 15 minutos seguidos, ya que los estudiantes se distraerán con cambios más frecuentes.
- La persona que no está enseñando no debe interrumpir, ofrecer correcciones o elaboraciones, o hacer cualquier otra cosa que distrae de lo que está haciendo o diciendo la persona que enseña. La única excepción es hacer preguntas guía si los estudiantes parecen pasivos o inseguros de sí mismos.
- Cada persona debería tomar un par de minutos antes de empezar a enseñar para ver lo que su compañero va a enseñar después de acabar, y luego *no* presentar nada de ese material.
- La persona que no está enseñando debe mantenerse comprometida con la clase, no ponerse al día con su correo electrónico. Supervisar las notas compartidas (Section 9.7), vigilar a los estudiantes para ver quién tiene dificultades, anotar algunos comentarios para dárselos a su compañero de enseñanza en el próximo receso—cualquier cosa que contribuya a la lección es mejor que cualquier cosa que no lo haga.

Lo más importante es que, cuando termine la clase, tomen unos minutos para felicitarse o compadecerse unos de otros: tanto en la enseñanza como en la vida, la miseria compartida disminuye y la alegría compartida aumenta.

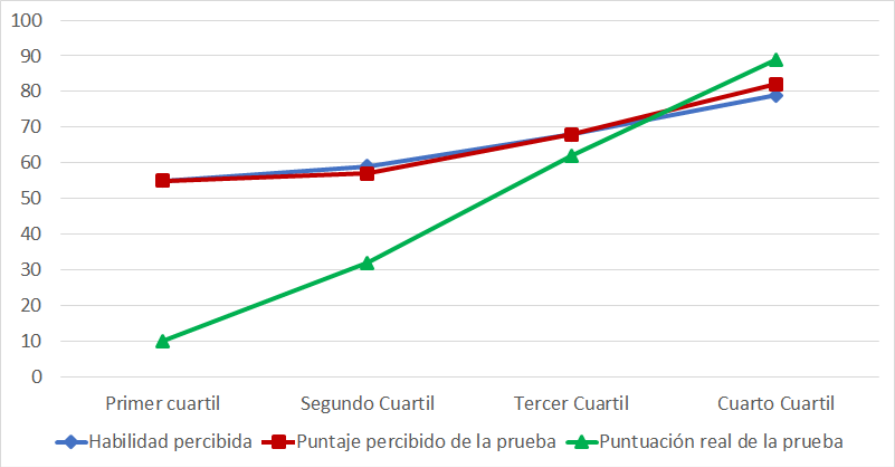

Figura 9.1: The Dunning-Kruger Effect

9.4. Evaluar conocimientos previos

Cuanto más sepas sobre tus estudiantes antes de comenzar a enseñar, más podrás ayudarles. Dentro de un sistema escolar formal, los pre-requisitos de tu curso te dirán algo sobre lo que probablemente ya sepan. Sin embargo, en un entorno de campo libre, tus estudiantes pueden ser mucho más diversos, por lo que es posible que quieras darles una breve encuesta o cuestionario antes de tu clase para averiguar qué conocimientos y habilidades tienen.

Pedir a las personas que se califiquen a sí mismas en una escala del 1 al 5 no tiene sentido porque cuanto menos saben las personas sobre un tema, con menor precisión pueden estimar sus conocimientos (Figure 9.1, de https://theness.com/neurologicablog/index.php/misunderstanding-dunning-kruger/[2]), un fenómeno llamado **Dunning-Kruger effect** [Krug1999]. Por el contrario, las personas que son miembros de grupos subrepresentados a menudo subestiman sus habilidades.

En lugar de pedirles a las personas que se autoevalúen, puedes preguntarles con qué facilidad podrían completar algunas tareas específicas. Sin embargo, hacer esto es arriesgado porque la escuela entrena a las personas para que traten cualquier cosa que parezca un examen como algo que tienen que aprobar en lugar de una oportunidad para dar forma a la instrucción. Si alguien responde "No sé" incluso a un par de preguntas en su pre-evaluación, podría concluir que tu clase es demasiado avanzada para ellos. Por lo tanto, podrías asustar a muchas de las personas que más deseas ayudar.

Section F.5 presenta un breve cuestionario de preevaluación que es poco proba-

[2] Neurologica

ble que la mayoría de los potenciales estudiantes encuentren intimidante. Si lo usas o algo parecido, trata de hacer un seguimiento con las personas que *no* responden para averiguar por qué no y compara tu evaluación de tus estudiantes con su autoevaluación para mejorar tus preguntas.

9.5. Plan for Mixed Abilities

Si tus estudiantes tienen niveles muy diversos de conocimientos previos, puedes terminar fácilmente en una situación en la que un tercio de tu clase se pierde y un tercio se aburre. Eso es poco satisfactorio para todos, pero hay algunas estrategias que puedes utilizar para manejar la situación:

- Antes de realizar un taller, comunica claramente su nivel a todos mostrando algunos ejemplos de ejercicios que se les pedirá que completen. Esto ayuda a los potenciales participantes a medir el nivel de la clase de manera mucho más efectiva que una lista de temas en forma de puntos.
- Proporciona ejercicios adicionales a su propio ritmo para que los estudiantes más avanzados no terminen temprano y se aburran.
- Pon atención a los estudiantes que se están quedando atrás e actúa temprano para que no se frustren ni se den por vencidos.
- Pide a tus estudiantes más avanzados ayudar a las personas que están a su lado (vea Section 9.6 debajo).

Otra forma de adaptarse a las habilidades mixtas es hacer que todos trabajen en el material por su cuenta a su propio ritmo, como lo harían en un curso en línea, pero hacerlo simultáneamente y lado a lado con los ayudantes que deambulan por la sala para despejar a las personas. Algunas personas llegarán tres o cuatro veces más lejos que otras cuando los talleres se realicen de esta manera, pero todos habrán tenido un día gratificante y desafiante.

Falsos principiantes

*Un **falso principiante** es alguien que ha estudiado un lenguaje antes pero está aprendiéndolo de nuevo. Pueden ser indistinguibles de **principiantes absolutos** en las pruebas de pre-evaluación, pero pueden moverse mucho más rápido una vez que comienza la clase porque están volviendo a aprender en lugar de aprender por primera vez.*

*Ser un falso principiante es a menudo una señal de **privilegio preparatorio** [Marg2010], y los falsos principiantes son comunes en las clases de programación de rango abierto. Por ejemplo, un niño cuya familia es lo suficientemente acomodada como para haberlos enviado a un campamento de verano de robótica puede tener un desempeño pobre en una pre-evaluación de conocimientos de programación porque el material no está*

> *fresco en su mente, pero aún tiene una ventaja sobre un niño de un trasfondo menos afortunado. Las estrategias descritas anteriormente pueden ayudar a nivelar el campo de juego en casos como este, pero nuevamente, la solución real es usar tu propio privilegio para abordar factores fuera de clase más importantes [Part2011].*

Lo más importante es aceptar que no puedes ayudar a todos todo el tiempo. Si reduces la velocidad para acomodar a dos personas que les está costando, estás fallando a las otras dieciocho. Del mismo modo, si dedicas unos minutos a hablar sobre un tema avanzado con un estudiante que está aburrido, el resto de la clase se sentirá excluida.

9.6. Pair Programming

Pair programming es una practica del desarrollo de software en la que dos programadores trabajan juntos en una computadora[3]. Una persona (el conductor) escribe mientras que la otra (el navegador) ofrece comentarios y sugerencias, y los dos cambian de función varias veces por hora.

El Pair programming es una práctica eficaz en el trabajo profesional [Hann2009] y también es una buena forma de enseñar: los beneficios incluyen una mayor tasa de éxito en los cursos introductorios, un mejor software y una mayor confianza de los estudiantes en sus soluciones. También hay evidencia de que estudiantes de grupos subrepresentados se benefician incluso más que otros [McDo2006; Hank2011; Port2013; Cele2018]. Los compañeros pueden ayudarse mutuamente durante los ejercicios prácticos, aclarar los conceptos erróneos de los demás cuando se presenta la solución y discutir intereses comunes durante los descansos. Lo he encontrado particularmente útil con las clases de habilidades mixtas, ya que las parejas son más homogéneas que los individuos.

Cuando utilices el pair programming, coloca a *todos* en parejas, no solo a los estudiantes que tienen dificultades, para que nadie se sienta seleccionado. También es útil que las personas se sienten en lugares nuevos (y, por lo tanto, se emparejen con diferentes compañeros) de forma regular, y que las personas cambien de rol dentro de cada pareja tres o cuatro veces por hora para que la personalidad más fuerte de cada pareja no domine la sesión.

Si tus estudiantes son nuevos en el pair programming, toma unos minutos para demostrar cómo se ve realmente para que comprendan que la persona que no tiene las manos en el teclado no debe sentarse y mirar. Finalmente, díles que las personas que se enfocan en tratar de completar la tarea lo más rápido posible son menos justas al compartir [Lewi2015].

Cambio de parejas

[3] https://www.youtube.com/watch?v=vgkahOzFH2Q

Los educadores tienen opiniones encontradas sobre si se debería exigir a las personas que cambien de pareja a intervalos regulares. Por un lado, les da a todos la oportunidad de obtener nuevos conocimientos y hacer nuevos amigos. Por otro lado, trasladar las computadoras y los adaptadores de corriente a escritorios nuevos varias veces al día es desgastante y la combinación puede resultar incómoda para los introvertidos. Dicho esto, [Hann2010] encontró una correlación débil entre los rasgos de personalidad de los "Cinco Grandes" y el rendimiento en la programación de parejas, aunque un estudio anterior [Wall2009] encontró que las parejas cuyos miembros tenían diferentes niveles de rasgos de personalidad se comunicaban con más frecuencia.

9.7. Take Notes... Together?

La toma de notas es una forma de elaboración en tiempo real (Section 5.1): te obliga a organizar y reflexionar sobre el material a medida que ingresa, lo que a su vez aumenta la probabilidad de que lo transfieras a la memoria a largo plazo. Muchos estudios han demostrado que tomar notas mientras se aprende mejora la retención [Aike1975; Boha2011]. Si bien aún no se ha estudiado ampliamente [Ornd2015; Yang2015], He descubierto que hacer que los estudiantes tomen notas juntos en una página en línea compartida también es eficaz:

- Permite a las personas comparar lo que creen que están escuchando con lo que otras personas están escuchando, lo que les ayuda a llenar los vacíos y corregir conceptos erróneos de inmediato.
- Les da a los estudiantes más avanzados de la clase algo útil que hacer. En lugar de aburrirse y revisar Instagram durante la clase, pueden tomar la iniciativa para registrar lo que se dice, lo que los mantiene comprometidos y permite que los estudiantes menos avanzados concentren más su atención en el nuevo material.
- Las notas que toman los estudiantes suelen ser más útiles para *ellos* que las que el educador prepararía de antemano, ya que es más probable que los estudiantes escriban lo que realmente encontraron nuevo en lugar de lo que el educador predijo que sería nuevo.
- Mirar notas recientes mientras los estudiantes están trabajando en un ejercicio ayuda al educador a descubrir que la clase se perdió o entendió algo mal.

¿Es el lápiz más poderoso que el teclado?

[Muel2014] informó que tomar notas en una computadora es generalmente menos efectivo que tomar notas con lápiz y papel. Si bien su resultado fue ampliamente compartido, [More2019] no pudo replicarlo.

Si los estudiantes toman notas juntos, también puedes hacer que peguen fragmentos cortos de código y respuestas en forma de puntos o de oraciones a las preguntas de evaluación formativa. Para evitar que todos intenten editar el mismo par de líneas al mismo tiempo, haz una lista con el nombre de todos y pégala en el documento siempre que quieras que cada persona responda una pregunta.

Los estudiantes a menudo descubren que tomar notas juntos les distrae la primera vez que lo intentan porque tienen que dividir su atención entre lo que dice el educador y lo que escriben sus compañeros (Section 4.1). Si estás trabajando con un grupo en particular solo una vez, debes prestar atención a los consejos en Section 9.12 y pedirles que tomen notas individualmente.

Puntos para Mejorar

Una forma de demostrarles a los estudiantes que están aprendiendo con*tigo, no solo* de *ti, es permitirles que tomen notas editando (una copia de) su lección. En lugar de publicar archivos PDF para que los descarguen, crea copias editables de tus diapositivas, notas y ejercicios en una wiki, un documento de Google o cualquier otra cosa que te permita revisar y comentar los cambios. Dale crédito a las personas por corregir errores, aclarar explicaciones, agregar nuevos ejemplos y escribir nuevos ejercicios no reduce tu carga de trabajo, pero aumenta el compromiso y la duración de la lección. (Section 6.3).*

9.8. Notas adhesivas

Las notas adhesivas son una de mis herramientas de enseñanza favoritas, y no soy el único que ama su versatilidad, portabilidad, adherencia, capacidad de plegado y aroma sutil pero atractivo [Ward2015].

Como indicadores de estado

Dale a cada estudiante dos notas adhesivas de diferentes colores, como naranja y verde. Estos se pueden sostener para votar, pero su uso real es como indicadores de estado. Si alguien ha completado un ejercicio y quiere que lo revisen, coloca la nota adhesiva verde en su laptop; si tienen un problema y necesitan ayuda, colocan el naranja. Esto funciona mucho mejor a que la gente levante la mano: es más discreto (lo que significa que es más probable que lo hagan), pueden seguir trabajando mientras su bandera está levantada en lugar de intentar escribir con una sola mano, y el educador puede ver rápidamente desde el frente del salón en qué estado se encuentra la clase. Los indicadores de estado son particularmente útiles cuando las personas en clases con habilidades mixtas están trabajando en el material a su propio ritmo (Section 9.5).

Una vez que sus estudiantes se sientan cómodos con dos adhesivos, dáles un

tercero que puedan usar cuando tengan el cerebro lleno o necesiten un descanso para ir al baño [4]. Nuevamente, es más probable que los adultos publiquen una nota adhesiva a que levanten la mano, y una vez que una nota adhesiva azul es levantada, generalmente sigue una ráfaga de otras.

Para distribuir atención

También se pueden usar notas adhesivas para garantizar que la atención del educador se distribuya de manera justa. Haz que cada estudiante escriba su nombre en una nota adhesiva y que lo ponga en su computadora portátil. Cada vez que el maestro los llama o responde una de sus preguntas, ponen su nota adhesiva debajo. Una vez que todas las notas adhesivas están abajo, todos vuelven a poner las suyas arriba.

Esta técnica hace que sea fácil para el educador ver con quién no ha hablado recientemente, lo que a su vez ayuda a evitar prejuicios inconscientes e interactuar preferentemente con sus estudiantes más extrovertidos. Sin una verificación como esta, es muy fácil crear un ciclo de retroalimentación en el que los extrovertidos reciben más atención, lo que los lleva a mejorar, lo que a su vez los lleva a recibir más atención, mientras que los estudiantes más introvertidos, menos seguros o marginados quedan detrás [Alvi1999; Juss2005].

También muestra a tus estudiantes que la atención se distribuye de manera justa, de modo que cuando se les llame, no se sientan molestados. Cuando trabajo con un grupo nuevo, permito que las personas tomen sus propias notas adhesivas durante la primera o la segunda hora de clase si prefieren que no los llamen. Si continúan haciendo esto a medida que pasa el tiempo, trato de tener una conversación tranquila con ellos para averiguar por qué y para ver si hay algo que pueda hacer para que se sientan más cómodos.

Como tarjetas de actas

También puedes usar notas adhesivas como **tarjetas de actas**. Antes de cada receso, los estudiantes se toman un minuto para escribir una cosa en la nota adhesiva verde que creen que será útil y una cosa en la nota naranja que encontraron demasiado rápido, demasiado lento, confuso o irrelevante. Mientras disfrutan de su café o almuerzo, revisa sus notas y busca patrones. Se necesitan menos de cinco minutos para ver qué disfrutan los estudiantes de una clase de 40 personas, en qué se sienten confundidos, qué problemas tienen y qué preguntas aún no has respondido.

Los estudiantes no deben firmar sus tarjetas de actas: están pensadas como comentarios anónimos. La técnica de uno arriba/uno abajo descrita en Section 9.11 es una oportunidad para una retroalimentación colectiva.

[4] Una colega me dijo una vez que la unidad básica de enseñanza es la vejiga. Cuando dije yo nunca había pensado en eso, dijo: "Obviamente nunca has estado embarazada"

9.9. Nunca una página en blanco

Los talleres de programación y otros tipos de clases se pueden construir en torno a un conjunto de ejercicios independientes, desarrollar un solo ejemplo extendido en etapas o utilizar una estrategia mixta. Las dos ventajas principales de los ejercicios independientes son que las personas que se retrasan pueden volver a sincronizar fácilmente y que los desarrolladores de lecciones pueden agregar, eliminar y reorganizar el material a voluntad (Section 6.3). Un solo ejemplo extendido, por otro lado, mostrará a los estudiantes cómo encajan las partes y piezas que están aprendiendo: en el lenguaje educativo, les brinda más oportunidades para integrar sus conocimientos.

Independientemente del enfoque que adopte, los principiantes nunca deben comenzar a hacer ejercicios con una página o pantalla en blanco, ya que a menudo les resulta intimidante o desconcertante. Si te han seguido mientras realizas la codificación en vivo, pídales que agreguen algunas líneas más o que modifiquen el ejemplo que creaste. Alternativamente, si están tomando notas juntos, pega algunas líneas de código de inicio en el documento para que lo amplíen o modifiquen.

Modificar el código existente en lugar de escribir código nuevo desde cero no solo proporciona a los estudiantes una estructura: también está más cerca de lo que harán en la vida real. Sin embargo, ten en cuenta que los estudiantes pueden distraerse tratando de comprender todo el código de inicio en lugar de hacer su propio trabajo. El `public static void main()` de Java o un puñado de sentencias `import` al inicio de un programa en Python podría tener sentido para ti, pero es carga extraña para ellas. (Chapter 4).

9.10. Configuración de tus estudiantes

Los estudiantes de rangos abiertos a menudo quieren traer sus propias computadoras y dejar la clase con esas máquinas configuradas para hacer un trabajo real. Por lo tanto, los educadores de rangos abiertos deberían prepararse para enseñar tanto en Windows como en MacOS [5], aunque sería más sencillo exigir a los estudiantes que utilicen solo uno.

Denominadores comunes

Si tus participantes utilizan diferentes sistemas operativos, trata de evitar el uso de funciones que sean específicas de uno solo y señala las que utilices*. Por ejemplo, los controles y el comportamiento de "minimizar ventana" en Windows son diferentes a los de MacOS.*

No importa cuántas plataformas tengas que manejar, coloca instrucciones de configuración detalladas en el sitio web de su curso y envía un correo electrónico a los

[5] "¡Y Linux!", grita alguien desde el fondo del salón.

estudiantes un par de días antes de que comience el taller para recordarles que realicen la configuración. Algunas personas seguirán apareciendo sin el software requerido porque tuvieron problemas, no pudieron encontrar tiempo para completar todos los pasos o simplemente son el tipo de persona que nunca sigue las instrucciones por adelantado. Para detectar esto, haz que todos ejecuten un comando simple tan pronto como lleguen y muestren el resultado a los educadores, luego busque ayudantes y otros estudiantes para ayudar a las personas que se han encontrado con problemas.

Máquinas Virtuales

Algunas personas usan herramientas como Docker[6] para poner máquinas virtuales en las computadoras de sus estudiantes para que todos trabajen exactamente con las mismas herramientas, pero esto presenta un nuevo conjunto de problemas. Las máquinas más antiguas o más pequeñas simplemente no son lo suficientemente rápidas para ejecutarlas, los estudiantes luchan por alternar entre dos conjuntos diferentes de atajos de teclado para cosas como copiar y pegar, e incluso los profesionales competentes se confundirán sobre qué está sucediendo exactamente y dónde.

La configuración es tan complicada que muchos educadores prefieren que los estudiantes usen herramientas basadas en navegador. Sin embargo, esto hace que la clase dependa del WiFi (que puede ser de calidad muy variable) y no satisface el deseo de los estudiantes de irse con sus propias máquinas listas para su uso en el mundo real. Mientras herramientas basadas en la nube como Glitch[7] y RStudio Cloud[8] se vuelven más robustas, esta última consideración se está volviendo menos importante.

Una última forma de abordar los problemas de configuración es dividir la clase en varios días y hacer que las personas instalen lo que se requiere para cada día antes de dejar la clase el día anterior. Dividir el trabajo en partes hace que cada una sea menos intimidante, es más probable que los estudiantes realmente lo hagan y garantiza que puedas comenzar a tiempo para cada lección, excepto la primera.

9.11. Otras prácticas de enseñanza

Ninguna de las prácticas más pequeñas que se describen a continuación es esenciales, pero todas mejorarán la entrega de lecciones. Como ocurre con el ajedrez y el matrimonio, el éxito en la enseñanza suele ser una cuestión de progreso lento y constante.

[6] http://docker.com
[7] https://glitch.com/
[8] http://rstudio.cloud

Comience con introducciones

Comienza tu clase presentándote. Si eres un experto, cuéntales un poco cómo llegaste a donde estás; si solo estás dos pasos por delante de ellos, enfatiza lo que usted y ellos tienen en común. Digas lo que digas, tus metas son hacerte más accesible y fomentar su creencia de que pueden tener éxito.

Los estudiantes también deben presentarse entre sí. En una clase de una docena, pueden hacer esto verbalmente; en una clase más grande o si son desconocidos entre sí, creo que es mejor que cada uno escriba una o dos líneas sobre sí mismos en las notas compartidas (Section 9.7).

Configura tu propio entorno

Configurar tu entorno es tan importante como configurar el de tus estudiantes, pero más involucrado. Además de tener acceso a la red y todo el software que vas a utilizar, también debes tomar un vaso de agua o una taza de té o café. Esto ayuda a mantener tu garganta lubricada, pero su propósito real es darte una excusa para hacer una pausa y pensar durante un par de segundos cuando alguien te hace una pregunta difícil o cuando pierdes la noción de lo que ibas a decir a continuación. Probablemente también querrá algunos marcadores de pizarra y algunas de las otras cosas que se describen en Section F.3.

One way to keep your day-to-day work from getting in the way of your teaching is to create a separate account on your computer for the latter. Use system defaults for everything in this second account, along with a larger font and a blank screen background, and turn off notifications so that your teaching isn't interrupted by pop-ups.

Evite la tarea en formatos de todo el día

Los estudiantes que hayan pasado todo un día programando estarán cansados. Si les das tarea para hacer fuera del horario de clase, también empezarán cansados al día siguiente, así que no lo hagas.

No toques el teclado de tu estudiante

A menudo es tentador arreglar las cosas para los estudiantes, pero incluso si narras cada paso, es probable que los desmotives al enfatizar la brecha entre sus conocimientos y los tuyos. En su lugar, mantenga sus manos fuera del teclado y hable con sus estudiantes sobre lo que tengan que hacer: llevará más tiempo, pero es más probable que se quede.

Repite la Pregunta

Siempre que alguien haga una pregunta en clase, repítasela antes de responder para comprobar que la ha entendido y para que las personas que quizás no la ha-

yan escuchado tengan la oportunidad de hacerlo. Esto es particularmente importante cuando se graban o transmiten presentaciones, ya que tu micrófono generalmente no captará lo que otras personas están diciendo. Repetir las preguntas también te da la oportunidad de redirigir la pregunta a algo con lo que se sienta más cómodo respondiendo...

Uno arriba, uno abajo

Un complemento de las tarjetas de actas es solicitar comentarios resumidos al final de cada día. Los estudiantes dan alternativamente un punto positivo o negativo sobre el día sin repetir nada de lo que ya se ha dicho. La prohibición de las repeticiones obliga a las personas a decir cosas que de otro modo no harían: una vez que se hayan dado todos los comentarios "seguros", los participantes comenzarán a decir lo que realmente piensan.

Diferentes modos, diferentes respuestas

Las tarjetas de actas (Section 9.8) son anónimas; la retroalimentación alterna de arriba a abajo no lo es. Debes usar los dos juntos porque el anonimato permite tanto la honestidad como la ofensa.

Haz que los estudiantes hagan predicciones

Las investigaciones han demostrado que las personas aprenden más de las demostraciones si se les pide que predigan lo que sucederá [Mill2013]. Hacer esto encaja naturalmente en la codificación en vivo: después de agregar o cambiar algunas líneas de un programa, pregunta a la clase qué sucederá cuando se ejecute. Si el ejemplo es incluso moderadamente complejo, la predicción puede servir como una pregunta motivadora para una ronda de instrucción entre pares.

Configuración de Mesas

Es posible que no tengas ningún control sobre la distribución de los escritorios o mesas en la sala en la que enseñas, pero si lo tienes, creemos que es mejor tener asientos planos (estilo cena) en lugar de asientos bancarios (estilo teatro), así puedes llegar a los estudiantes que necesitan ayuda más fácilmente y que sea más fácil para los estudiantes emparejarse entre sí (Section 9.5). Las tomas de corriente en el piso para que no tengas que pasar cables de alimentación por el piso hacen la vida más fácil y segura, pero siguen siendo poco comunes.

Independientemente del diseño que tengas, trata de asegurarte de que cada asiento tenga una vista sin obstáculos de la pantalla. Un buen soporte para la espalda también es importante, ya que las personas estarán en ellos durante un período prolongado. Al igual que los tomacorrientes en el piso, los buenos asientos en el salón de clases aún son infrecuentes.

Pastillas para la tos

Si hablas todo el día en una habitación llena de gente, se te irrita la garganta porque está irritando las células epiteliales de la laringe y la faringe. Esto no solo te vuelve ronco, sino que también te hace más vulnerable a las infecciones (que es parte de la razón por la que las personas a menudo se resfrían después de enseñar).

La mejor manera de protegerse contra esto es mantener la garganta alineada, y la mejor manera de hacerlo es usar pastillas para la tos temprano y con frecuencia. Los buenos también enmascararán la aparición del aliento a café, por lo que tus estudiantes probablemente estarán agradecidos.

Piensa, empareja, comparte

Piensa, empareja, comparte es una técnica ligera que ayuda a las personas a mejorar sus ideas mediante la discusión con sus compañeros. Cada persona comienza pensando individualmente sobre una pregunta o problema y anotando algunas notas. Luego, se explican sus ideas por parejas, fusionándolas o seleccionando las más prometedoras. Finalmente, algunas parejas presentan sus ideas a todo el grupo. Piensa, empareja, comparte funciona porque obliga a las personas a exteriorizar su cognición (Section 3.1). También les da la oportunidad de detectar y resolver brechas o contradicciones en sus ideas *antes* de exponerlas a un grupo más grande, lo que puede hacer que tus estudiantes menos extrovertidos estén un poco menos nerviosos por parecer tontos.

Mañana, mediodía y noche

[Smar2018] descubrió que a los estudiantes les va peor si sus clases y otros trabajos se programan en horarios que no coinciden con sus relojes corporales naturales, es decir, que si una persona matutina toma clases nocturnas o viceversa, sus calificaciones se ven afectadas. Por lo general, no es posible acomodar esto en grupos pequeños, pero los más grandes deben intentar escalonar las horas de inicio de las sesiones paralelas. Esto también puede ayudar a las personas a hacer malabarismos con las responsabilidades del cuidado de los niños y otras limitaciones, y reducir la duración de las filas en las pausas para el café y en los baños.

Humor

El humor debe usarse con moderación al enseñar: la mayoría de los chistes son menos divertidos cuando se escriben y se vuelven aún menos divertidos con cada relectura. Ser espontáneamente divertido mientras enseñas generalmente funciona mejor, pero puede salir mal fácilmente: lo que es una broma para tu círculo de amigos puede convertirse en un problema político serio para tu audiencia. Si haces bromas cuando enseñas, no las hagas a expensas de ningún grupo o de ningún individuo excepto posiblemente de ti mismo.

9.12. Limita la innovación

Cada una de las técnicas presentadas en este capítulo mejorará tus clases, pero no debes intentar adoptarlas todas a la vez. La razón es que cada nueva práctica aumenta *tu* carga cognitiva, así como la de tus estudiantes, ya que ahora todos están tratando de aprender una nueva forma de aprender, así como el tema de la lección. Si trabajas con un grupo repetidamente, puedes introducir una técnica nueva cada pocas lecciones; si tú solo los tienes para un taller de un día, es mejor elegir solo un método que no hayan visto antes y que se sientan cómodos con eso.

9.13. Ejercicios

Crea un cuestionario (individual/20)

Utiliza el cuestionario de Section F.5 como plantilla, crea un breve cuestionario que puedas entregar a tus estudiantes antes de impartir una clase propia. ¿Qué es lo que más deseas saber sobre sus antecedentes y cómo pueden ambas partes estar seguras de que están de acuerdo en qué nivel de comprensión está preguntando?

Uno de los tuyos (Toda la clase/15)

Piensa en una práctica de enseñanza que no se haya descrito hasta ahora. Presenta tu idea a un compañero, escucha la de ellos y seleccione una para presentarlo al grupo en general. (Este ejercicio es un ejemplo de pensar-emparejar-compartir).

¿Puedo conducir? (pares/10)

Intercambia computadoras con un compañero (preferiblemente alguien que use un sistema operativo diferente al tuyo) y trabaje con un simple ejercicio de programación. ¿Qué tan frustrante es? ¿Cuánta información te da sobre lo que los novatos tienen que pasar todo el tiempo?

Emparejar (parejas/15)

Mira este video[9] de pair programming y luego practica hacerlo con un compañero. Recuerda cambiar los roles entre conductor y navegador cada pocos minutos. ¿Cuánto tiempo tardas en adoptar un ritmo de trabajo?

[9] https://www.youtube.com/watch?v=vgkahOzFH2Q

Compara Notas (grupos pequeños/15)

Forma grupos de 3 a 4 personas y compare las notas que ha tomado en este capítulo. ¿Qué le pareció digno de mención que sus compañeros perdieron y viceversa? ¿Qué entendiste diferente?

Credibilidad (individual/15)

[Fink2013] describe tres cosas que hacen que los educadores sean creíbles a los ojos de sus estudiantes:

Competencia: conocimiento del tema como lo demuestra la capacidad para explicar ideas complejas o hacer referencia al trabajo de otros.

Integridad: teniendo en cuenta los mejores intereses de los estudiantes. Esto se puede demostrar dando retroalimentación individualizada, ofreciendo una explicación racional para las decisiones de calificación y tratando a todos los estudiantes por igual.

Dinamismo: entusiasmo por el tema (Chapter 8).

Describe una cosa que haces al enseñar que se ajuste a cada categoría y luego describe una cosa que no haces pero que deberías hacer.

Medir la eficacia (individual/15)

[Kirk1994] define cuatro niveles en los cuales evaluar la formación:

Reacción: ¿Cómo se sintieron los estudiantes con respecto a la formación?

Aprendizaje: ¿Cuánto aprendieron realmente?

Comportamiento: ¿Cuánto han cambiado su comportamiento como resultado?

Resultados: ¿Cómo han afectado esos cambios de comportamiento su resultado o el resultado de su grupo?

¿Qué estás haciendo en cada nivel para evaluar qué y cómo enseñas? ¿Qué podrías hacer que no estés haciendo?

Objeciones y contraobjeciones (piensa-empareja-comparte/15)

Has decidido no preguntarles a tus estudiantes si su clase fue útil porque sabes que no existe una correlación entre sus respuestas y cuánto aprenden realmente (Section 7.1). En cambio, has presentado cuatro propuestas, cada una de las cuales tus colegas han rechazado:

Ve si recomiendan la clase a sus amigos. ¿Por qué esto sería más significativo que preguntarles cómo se sienten acerca de la clase?

Hazles un examen al final. Pero cuánto saben los estudiantes al final del día es un mal predictor de cuánto recordarán dos o tres meses después, y cualquier tipo de examen final hará que la clase sea mucho más estresante.

Hazles un examen dos o tres meses después. Eso es prácticamente imposible con los estudiantes de rango abierto, y las personas que no obtuvieron nada del taller probablemente tengan menos probabilidades de participar en el seguimiento, por lo que los comentarios recopilados de esta manera serán sesgados.

Revisa si siguen usando lo que aprendieron. La instalación de software espía en las computadoras de los estudiantes está mal vista, entonces, ¿cómo se implementará?

Trabajando por tu cuenta, encuentra respuestas a estas objeciones, luego comparte tus respuestas con un compañero y discute los enfoques que ha ideado. Cuando hayas terminado, comparte tu enfoque favorito con la clase.

10

Motivation and Demotivation

Estudiantes necesitan ánimo para enfrentar terreno desconocido, asi que este capítulo analiza maneras en la que docentes pueden motivarlas/los. Y más importante, el capítulo habla de como docentes pueden *desmotivarlas/los* and how to avoid doing that. y como evitar lo mismo.

Nuestro punto de comienzo es la diferencia entre **motivación extrínseca**, lo que sentimos cuando hacemos algo para evitar castigo o ganarse una recompensa,

y **motivación intrínseca**, lo que sentimos cuando conseguimos algo que nos satisface personalmente. Ambos afectan la mayoría de situaciones—por ejemplo, personas enseñan porque les gusta y porque les pagana—pero aprendemos mejor cuando estamos motivados intrínsecamente citeWlod2017. De acuerdo a self-determination theory (*teoría de autodeterminación*)[1], los tres impulsores de motivación intrínseca son:

Competencia: el sensación de que sabes lo que haces.

Autonomía: la sensación de estar en control de tu propio destino.

Relación: la sensación de estar conectada/o con las/los demás.

Una lección bien diseñada anima a los tres. Por ejemplo, un ejercicio de programación puede dejar que las/los estudiantes practiquen con las herramientas que necesitan usar para resolver un problema mayor (competencia), dejarlas/los abordar las partes del problema en el orden que quieran (autonomía), y dejarlas/los conversar con sus compañeras/os (relación).

El problema de las notas

Yo nunca he tenido una audiencia en mi vida. My audiencia es una rúbrica. – citado por Matt Tierney[2]

Califaciones y la manera en que distorsionan el aprendizaje se utilizan con frecuencia como ejemplo de motivación extrínseca, pero como observa [Mill2016a], no van a desaparecer en el corto plazo, así que no tiene sentido intentar construir un sistema que los ignora. En vez, [Lang2013] explora como cursos que enfatizan calificaciones pueden incentivar a los estudiantes a que hagan trampa y ofrece algunos consejos de cómo disminuir este efecto, mientras [Covi2017] observa el problema más grande de

[1] https://en.wikipedia.org/wiki/Self-determination_theory
[2] https://twitter.com/figuralities/status/987330064571387906

*balancear motivación intrínseca y extrínseca en educación institucional, y el enfoque de constructive alignment (*alineación constructiva*)*[3] *defendido en [Bigg2011] busca traer actividades y resultados de aprendizaje en línea unos con otros.*

[Ambr2010] contiene una lista de métodos basados en evidencia para motivar a las estudiantes. Ninguno es sorprendiente—es difícil imaginar alguien diciendo que *no debemos* identificar y recompensar lo que valoramos—pero es útil revisar lecciones para asegurar de que estén haciendo al menos algunas de estas cosas. Una estrategia en particular que me gusta es tener estudiantes quienes hayan luchado y tenido éxito llegar y echar sus cuentos al resto de la clase. Es más probable que estudiantes crean cuentos de personas como ellas, y personas quienes hayan pasado por tu curso siempre tendrán consejos en los que nunca hubieras pensado.

No solo para estudiantes

Discusiones de motivación en educación con frecuencia sobrepasan la necesidad de motivar la docente*. Estudiantes responden al entusiasmo de un docente, y docentes (particularmente las voluntarias) necesitan valorar un tema para seguir enseñándolo Esta es otra razón poderosa para coenseñar (Section 9.3): al igual que tener una compañera para correr hace que sea más probable que sigas corriendo, tener una compañera docente ayuda a ponerte en marcha esos días que tienes una gripe y la bombilla del proyector se ha fundido y nadie sabe donde conseguir un reemplazo y en serio, ¿están haciendo construcción* otra vez*?*

Docentes también pueden hacer otras cosas positivas. [Bark2014] consiguió tres cosas que impulsaron la retención para todos los estudiantes: asignaciones significativas, interacción del profesorado con los estudiantes y colaboración estudiantil en las asignaciones. El ritmo y la carga de trabajo relativo a las expectativas tambien fueron impulsores significativos, pero primariamente para estudiantes varones. Cosas que *no* impulsaron retención fueron interacciones con asistentes de lección e interacciones con compañeras en actividades extracurriculares. Estos resultados parecen obvios, pero lo contrario también parecería obvio: si el estudio hubiera encontrado que actividades extracurriculares impulsan retención, también pensaríamos que tiene sentido. Notablemente, dos de los cuatro impulsores de retención (interacción del profesorado y colaboración estudiantil) toman esfuerzo adicional para replicar en línea (Chapter 11).

10.1. Tareas auténticas

Como Dylan Wiliam menciona en [Hend2017], motivación no siempre conduce al logro pero el logro casi siempre conduce a la motivación: el éxito de las estudian-

[3] https://en.wikipedia.org/wiki/Constructive_alignment

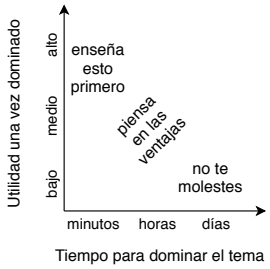

Figura 10.1: Que enseñar

tes las motiva mucho más a que les digan lo maravillosas que son. Podemos usar esta idea en la docencia creando una cuadrícula cuyos ejes son "tiempo medio para dominar" y "utilidad una vez dominado" (Figure 10.1).

Cosas que se dominan rápido y son útil de inmediato se deben enseñar primero, incluso si no se consideran fundamental por personas que ya son practicantes competentes porque unas pocas victorias tempranas fortalecerán la confianza de las estudiantes en sí mismas y en su docente. Por el contrario. cosas que son difíciles de aprender y no son útiles a tus estudiantes en su etapa actual de desarollo deben omitirse por completo, mientras que temas en la diagonal deben sopesarse entre si.

¿Útil para quién?

Si alguien quiere construir sitios web, conceptos de ciencia de computación fundamentales como recursión y computabilidad pueden habitar la esquina inferior derecha de esta cuadrícula. Eso no quiere decir que no valen la pena aprendiendo, pero si nuestro objetivo es motivar a las personas, pueden ser y deben ser deferidos. Por lo contrario, un estudiante de ultimo ano tomando un clase de programacion para estimular su mente puede preferir explorar estas ideas grandes en vez de hacer algo práctico. Cuando estas creando tu cuadricula, debes hacerlo con tus estudiante tipos en mente (Section 6.1). Si temas terminan en muy diferentes lugares para diferentes personas, debes pensar en crear cursos diferentes.

Un ejemplo bien estudiado de priorizar lo que es útil sin sacrificar lo que es fundamental es el enfoque de computación de medios desarrollado en Georgia Tech [Guzd2013]. En vez de imprimir "hello world" o sumando los primeros diez enteros, el primer programa de un/a estudiante podría abrir una imagen, cambiarle el tamaño para crear una versión miniatura, y guardar el resultado. Es es una **tarea auténtica** es decir, algo que los estudiantes creen que harían en la vida real. Tambien tiene un **artefacto tangible**: si la imagen sale el tamaño incorrecto, estudiantes tienen algo en mano que puede guiar su depuración. [Lee2013] describe una adaptación de este enfoque de Python a MATLAB, mientras que otros están construyendo cursos similares acerca de ciencia de datos, procesamiento de imágenes y biología [Dahl2018; Meys2018; Ritz2018]

Siempre habrá tensión entre dándole a los estudiantes problemas auténticos y ejercitando las habilidades individuales que requieren para resolver esos problemas: al fin y al cabo, programadores no contestan preguntas de opción múltiple en el trabajo como tampoco los músicos tocan escalas una y otra vez en frente de una audiencia. Consiguiendo el balance es difícil, pero un primer paso es eliminar cualquier cosa arbitraria o sin sentido. Por ejemplo, ejemplos de programación no deben utilizar variables llamadas `foo` y `bar`, y si vas a hacer que tus estudiantes ordenen una lista, hazla una lista de canciones en vez de cadenas de caracteres como "aaa" y "bbb".

10.2. Desmotivación

> Las mujeres no van de la computación porque no saben como es; se van porque *si saben*.
> — atribuido a varias personas

Si enseñas en un ambiente de rango libre probablemente tus estudiantes son voluntarias/os, y probablemente quieren estar en tu aula. Por lo tanto, motivarlos es menos una preocupante que no desmotivarlos. Desafortunadamente, es fácil desmotivar a la personas accidentalmente. Por ejemplo, [Cher2009] reportó cuatro estudios

mostrando que las pistas ambientales sutiles tienen una diferencia medible en el interés que personas de diferentes géneros tienen en computación: cambiando objetos en un aula de ciencias de la computación de aquellos considerados estereotipados de ciencias de la computación (p.ej. afiches y video juegos de Star Trek) a objetos no considerados estereotipados (p.ej. afiches de la naturaleza y guías telefónicas impulsó el interés de estudiantes universitarias al nivel de sus pares masculinos. De forma similar, [Gauc2011] reporta un trío de estudios mostrando que la redacción de género comúnmente empleada en materiales de contratación laboral pueden mantener desigualdad de género en ocupaciones tradicionalmente dominadas por hombres.

Hay tres desmotivadores principales para estudiantes adultos:

Imprevisibilidad desmotiva a las personas porque si no hay una conexión confiable entre lo que hacen y el resultado que logran, no hay razón para que intenten hacer nada.

Indiferencia desmotiva porque estudiantes que creen que la docente o el sistema educacional no se preocupan por ellas o por el material tampoco se preocuparán por ellos.

Injusticia desmotiva a las personas desfavorecidas por razones obvias. Lo sorprendente es que tambien desmotiva a las personas que se benefician de la injusticia: conciente o inconcientemente, les preocupa que algun dia se encuentren en el grupo desfavorecido [Wilk2011].

En situaciones extremas, estudiantes pueden desarrollar **indefensión aprendida**: sometidas repetidamente a comentarios negativos en una situación que no pueden cambiar, pueden aprender a ni si quiera intentar cambiar las cosas que podrían.

Una de las maneras más rápidas y seguras de desmotivar estudiantes es usar lenguaje que sugiere que algunas personas son programadores naturales y otras no. Guzdial ha llamado esto el mito mas grande de enseñar ciencias de la computación[4], y [Pati2016] respaldó esto mostrando que la gente ve evidencia de un "gen geek" donde no existe uno. Analizaron distribuciones de cualificaciones de 778 cursos universitarios y encontraron que solo 5.8 % mostraba señas de ser multimodal, es decir, solo una clase de veinte mostró señas de tener dos poblaciones distintas de estudiantes, Luego le mostraron a 53 profesores de ciencias de la computación histogramas de distribuciones ambiguas de calificaciones; aquellos que creían que algunas personas tienen una predisposición innata a ser mejor en las ciencias de la computación eran más propensos de verlas como bimodal que aquellos que no.

Estas creencias son importantes porque las docentes actúan sobre ellas [Brop1983]. Si un docente cree que es probable que a un estudiante le vaya bien naturalmente (a menudo inconscientemente) se enfoca en ese estudiante, que luego cumple con las expectativas de la docente debido a la mayor atención, que a su vez parece confirmar la creencia de la docente. Lamentablemente, hay pocas señales de que la mera evidencia del tipo presentado en [Pati2016] es suficiente para romper este círculo vicioso...

[4] https://cacm.acm.org/blogs/blog-cacm/189498-top-10-myths-about-teaching-computer-science/fulltext

Aquí hay algunas otras cosas específicas que desmotivarán a tus estudiantes:

Una actitud más-santa-que-tú o desdeñosa de un/una docente o un/una compañero/a estudiante.

Diciendo que sus habilidades existentes son tonterías. Personas usuarias de Unix se burlan de Windows, programadoras/es de todo tipo hacen chistes sobre Excel, y sin importar qué marco de aplicación web conoces, alguna programadora te dirá que está desactualizado. Estudiantes a menudo han invertido mucho tiempo y esfuerzo para adquirir las habilidades que tienen; menospreciarlos es una buena manera de garantizar que no escucharán más nada de lo que tengas que decir.

Sumergirse en discusiones técnicas complejas o detalladas con los estudiantes más avanzados de la clase.

Fingiendo que sabes más de lo que sabes. Las/os estudiantes confiarán más en ti si eres franca/o acerca de los límites de tu conocimiento, y será más probable que hagan preguntas y pidan ayuda.

Usando la letra S ("solo") o fingiendo sorpresa. Como se discutió en Chapter 3, diciendo cosas como "no puedo creer que no sabes X" o "¿nunca has oído de Y?" le señala a los/las estudiantes que el/la docente piensa que su problema es trivial y que deben ser estúpidos/as por no poder resolverlo.

Dolores de cabeza de instalación de software. El primer contacto de las personas con programación o con herramientas nuevas de programación suele a ser desmoralizador, y creer que algo es difícil de aprender es una profecía autocumplida. No es solamente el tiempo que toma en configurar o el sentimiento que es injusto tener que depurar algo que depende de precisamente el conocimiento que aún no tienen. El problema real es que cada falla de este tipo refuerza su creencia de que tendrán un mejor chance de cumplir con la fecha límite del próximo jueves si siguen haciendo las cosas como siempre las han hecho.

Es incluso más fácil desmotivar a las personas en línea que en persona, pero ahora hay estrategias basadas en evidencia para lidiar con esto. [Ford2016] consiguió que cinco barreras para contribuir en Stack Overflow[5] se consideran significativamente más problemáticas por mujeres que hombres: falta de conocimiento de las características del sitio, sentirse incapaz de contestar preguntas, un tamaño de comunidad intimidante, malestar interactuando con extraños o depender de ellos, y la sensación de que buscando cosas en línea no era "trabajo real." El medio de comentarios negativos no llegó a esta lista, pero de seguro sería la próxima agregada si los autores no fueran tan estrictos con sus límites estadísticos. Todos estos factores pueden y deben abordarse tanto en persona como en línea usando métodos como los de Section 10.4, y hacerlo mejora los resultados para todos [Sved2016].

Fracaso productivo y privilegio

[5] https://stackoverflow.com/

> *Algunos trabajos recientes han explorado **fracaso productivo**, donde a los/las estudiantes les dan deliberadamente problemas que no pueden ser resueltos con el conocimiento que tienen y tienen que ir y adquirir información nueva para poder avanzar [Kapu2016]. Fracaso productivo recuerda superficialmente la mantra del sector tecnológico "fracasa rápido, fracasa con frecuencia" pero este último es más un indicador de privilegio que de comprensión. Las personas solo pueden darse el lujo de celebrar el fracaso sí es seguro que tendrán una oportunidad de volver a intentarlo; muchos de tus estudiantes, y muchas personas de grupos marginados o desfavorecidos, no pueden estar seguro de esto, y asumir que el fracaso es una opción es una buena manera de desmotivarlas.*

Síndrome del impostor

Síndrome del impostor es la creencia de que tus logros son casualidades suertudas y viene con miedo de que alguien finalmente se dará cuenta. Es muy común entre las grandes triunfadoras que realizan un trabajo visible públicamente, pero afecta de manera desproporcionada a miembros de grupos subrepresentados: como se discutió en Section 7.1, [Wilc2018] consiguió que estudiantes expuestas previamente a la computación superaron a sus compañeros masculinos en todas las áreas en los cursos de introducción a la programación pero constantemente tenían menos confianza en sus habilidades, en parte porque la sociedad sigue señalando en maneras sutiles y no-tan-sutiles que realmente no pertenecen.

Aulas tradicionales pueden alimentar al síndrome del impostor. El trabajo escolar se realiza con frequencia solo o en grupos pequeños, pero los resultados se comparten y critican públicamente. Como resultado, raramente vemos como otras luchan por teminar su trabajo, lo que puede alimentar la creencia que esto le viene fácil a todos los demás. Miembros de grupos subrepresentados que ya sienten una presión adicional para demostrar su valía pueden ser particularmente afectados.

The Ada Initiative ha creado unas guías[6] para luchar con tu propio síndrome del impostor, que incluyen:

Habla del problema con personas de tu confianza. Cuando escuchas de otras personas que el síndrome del impostor es un problema común, se vuelve más difícil creer que tus sentimientos de fraude son reales.

Vaya a una sesión en persona sobre el síndrome del impostor. No hay nada como estar en un salón lleno de personas que respetas y descubriendo que el 90 % de ellas tienen síndrome del impostor.

Cuida tus palabras, porque influyen tu forma de pensar. Diciendo cosas como, "No soy un experto en esto, pero…" resta valor del conocimiento que realmente posees.

Enseña a otras personas de tu campo. Ganarás confianza en tu propio conoci-

[6] https://www.usenix.org/blog/impostor-syndrome-proof-yourself-and-your-community

miento y habilidad y ayudarás a evitar algunos cardúmenes del síndrome del impostor.

Haga preguntas. Haciendo preguntas puede ser intimidante si piensas que debes saber la respuesta, pero obtener respuestas elimina la prolongada agonía de la incertidumbre y el miedo al fracaso.

Construya alianzas. Tranquiliza y fortalece a tus amistades, quienes te reconfortan y fortalecen a cambio. (Y si no, tal vez quieras pensar en conseguir amistades nuevas...)

Posea tus logros. Siga registrando y revisando activamente lo que has hecho, lo que has construido, y los éxitos que has tenido.

Como docente, tu puedes ayudar a personas con su síndrome del impostor compartiendo cuentos de errores que has cometido o cosas que te costaron aprender. Esto le asegura a la clase que está bien encontrar temas difíciles. Ser abierta con el grupo también genera confianza y le da confianza para hacer preguntas. (La codificación en vivo es excelente para esto: como se indicó en Section 8.1, tus errores tipográficos le muestran a tu clase que eres humano/a.) Las evaluaciones formativas frecuentes también ayudan, en particular si estudiantes te ven ajustando lo que enseñas o tu velocidad basado en sus resultados.

Mentalidad y amenaza de estereotipo

Carol Dweck y otros han estudiado las diferencias de la **mentalidad fija** y la **mentalidad de crecimiento** en resultados de aprendizaje. Si personas creen que la competencia en alguna área es intrínseca (es decir, que tienes "el gen" para ella o no), a *todos* les va peor, incluyendo a los/las supuestamente aventajados/as La razón es que si a alguien no le va bien al principio, asumen que les falta esa aptitud, lo que predispone su rendimiento en el futuro. Por otro lado, si las personas creen que una habilidad se aprende y se puede mejorar, les irá mejor en promedio.

Hay preocupaciones[7] que la mentalidad de crecimiento se ha exagerado, o que es mucho más difícil traducir investigaciónes al respeto a la práctica de lo que sus defensores más entusiastas han insinuado [Sisk2018]. Sin embargo, si parece que estudiantes de un nivel socioeconomico bajo o que estan en riesgo académico podrían beneficiarse de las intervenciones de mentalidad.

Otro efecto discutido ampliamente es **amenaza de estereotipo** [Stee2011]. Recordando a personas de estereotipos negativos, incluso en formas sutiles, puede hacerlas sentirse ansiosas por el riesgo de confirmar esos estereotipos, lo que a su vez puede reducir su rendimiento. Otra vez, hay unas preocupaciones sobre la replicabilidad de estudios claves[8], y el problema se complica aún más por el hecho de que el

[7] https://educhatter.wordpress.com/2017/03/26/growth-mindset-is-the-theory-flawed-or-has-gm-been-debased-in-the-classroom/

[8] https://www.psychologytoday.com/blog/rabble-rouser/201512/is-stereotype-threat-overcooked-overstated-and-oversold

término se ha utilizado de muchas formas [Shap2007], pero nadie diría que mencionar los estereotipos en clase ayudaría a los/las estudiantes.

10.3. Accesibilidad

Colocar las lecciones y los ejercicios fuera del alcance de alguien es tan desmotivador como parece, y es muy fácil hacerlo sin darse cuenta. Por ejemplo, las primeras lecciones de programación en línea que escribí tenían una transcripción de la narración al lado de las diapositivas, pero no incluyó el código fuente: eso estaba en capturas de pantalla de diapositivas PowerPoint. Alguien utilizando un lector de pantalla[9] podía entonces oír lo que se decía sobre el programa, pero no sabía lo que era realmente el programa. No siempre es factible adaptarse a las necesidades de cada estudiante, pero agregando títulos de descripción a las imágenes y hacer que los controles de navegación sean accesible a personas que no pueden usar el mouse puede hacer una gran diferencia.

Bordillos

Hacer que el material sea accesible ayuda a todos, no solamente a las personas con dificultades. Bordillos[10]*—las pequenas rampas inclinadas que unen una acera a la calle—fueron fueron creados originalmente para facilitar el movimiento de personas con discapacitación fisica, pero resultó ser igual de útil para personas con cochecitos y carritos de supermercado. De manera similar, subtitular images no solamente ayuda a las personas con discapacitación visual: tambien hace que las imagenes sean mas faciles de encontrar e indexar para los motores de búsqueda*

El primer paso y el más importante para hacer lecciones accesibles es involucrar a las personas con discapacitaciones en el procesos de hacer decisiones: el eslogan *nihil de nobis, sine nobis*[11] (literalmente, "nada sobre nosotros sin nosotros") es anterior a los derechos de accesibilidad, pero siempre es un punto adecuado de partida. Unas recomendaciones específicas son:

Descubra lo que debes hacer. Cada uno de estos afiches[12] ofrece lo que debe y no se debe hacer para personas con autismo, usuarios/as de lectores de pantalla, y personas con baja visión, discapacidades físicas o motoras, ejercicios de escucha, y dislexia.

No hagas todo a la vez. Las mejoras descritas en el punto anterior pueden parecer bastante abrumadoras, así que haz un cambio a la vez.

[9] https://en.wikipedia.org/wiki/Screen_reader
[10] https://es.wikipedia.org/wiki/Bordillo
[11] https://es.wikipedia.org/wiki/Nada_sobre_nosotros_sin_nosotros
[12] https://accessibility.blog.gov.uk/2016/09/02/dos-and-donts-on-designing-for-accessibility/

Primero haz las cosas fáciles. Tamaño de fuente, usando un micrófono de clip para que las personas te puedan oír más fácilmente, y revisando tu selección de colores son buenos puntos de partida.

Sepa lo bien que lo está haciendo. Sitios como WebAIM[13] permiten que revises cuán accesibles son tus materiales en línea para usuarios con discapacidad visual.

[Coom2012; Burg2015] son buenas guías de diseño visual para la accesibilidad. Sus recomendaciones incluyen:

Formatea documentos con encabezados reales y otros puntos de referencias en vez de simplemente cambiar los tamaños y estilos de fuente.

Evita usar solamente el color para transmitir significado en texto o gráficos. En su lugar, use el color en combinación con diferentes patrones de rayado cruzado (que también hace que el material sea comprensible cuando se imprime en blanco y negro).

Elimina elementos innecesarios en lugar de hacerlos invisibles, porque los lectores de pantalla todavía los dirán en voz alta.

Permita auto-ritmo y repetición para las personas con problemas de lectura o audición.

Incluya narración de la acción de pantalla en los videos (y habla mientras escribes cuando codificas en vivo).

Cucharas

En el 2003, Christine Miserandino comenzó a usar cucharas[14] como una forma de explicar cómo es vivir con una enfermedad crónica. Personas sanas comienzan cada día con una cantidad ilimitada de cucharas, pero aquellas con lupus u otras condiciones debilitantes solo tienen unas pocas, y cada cosa que hacen les cuesta una. ¿Levantarse de la cama? Esa es una cuchara. ¿Preparando una comida? Esa es otra cuchara, y pronto se te acaban.

> No puedes simplemente ponerte la ropa cuando estás enfermo/a... Pero si mis manos duelen ese dia, los botones están fuera de discusión. Si tengo moretones ese dia, Necesito usar mangas largas, y si tengo una fiebre necesito un abrigo para mantenerme abrigado/a, y así sucesivamente. Si se me cae el cabello necesito pasar más tiempo para lucir presentable, y luego tienes que tomar en cuenta otros 5 minutos para sentirte mal de que te tomó 2 horas hacer todo esto.

Como Elizabeth Patitsas has argumentado[15], las personas que tienen muchas cucharas pueden acumular más, pero las personas cuya cantidad es limitada pueden

[13] http://webaim.org/
[14] https://butyoudontlooksick.com/articles/written-by-christine/the-spoon-theory/
[15] https://patitsas.blogspot.com/2018/03/spoons-are-form-of-capital.html

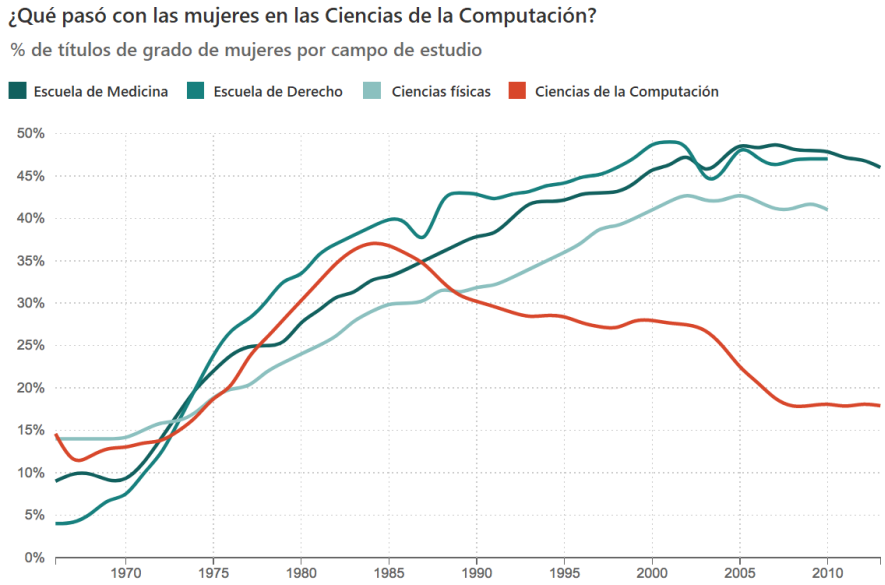

Figura 10.2: Estudiantes universitarias de ciencias de la computación en los EEUU

tener dificultades para salir adelante. Al diseñar clases y ejercicios, recuerda que algunos/as de tus estudiantes pueden tener obstáculos físicos o mentales que no son obvios. En caso de duda, pregunta: es casi seguro que tengan más experiencia con lo que funciona y lo que no que cualquier otra persona.

10.4. Inclusivity

Inclusión es una póliza para incluir a las personas que de otro modo pueden quedar excluidas o marginadas. En la computación, requiere hacer un esfuerzo positivo para ser más acogedor con las mujeres, grupos raciales o étnicos subrepresentados, personas con diversas orientaciones sexuales, los/as adultos mayores, aquellos con dificultades físicas, los anteriormente encarcelados, los desfavorecidos económicamente, y todos/as las demás que no encajan en el grupo demográfico de hombres blancos/asiáticos prósperos de Silicon Valley. Figure 10.2 (de NPR[16]) ilustra gráficamente los efectos de la cultura excluyente hacia mujeres en la computación.

[Lee2017] es una guía breve y práctica de como hacer eso con referencias a la literatura de investigación. Las prácticas que describe ayudan a estudiantes que pertenecen a uno o más grupos marginados o excluidos, pero también ayudan a motivar

[16]https://www.npr.org/sections/money/2014/10/21/357629765/when-women-stopped-coding

a todos los demás. Están redactadas en términos de cursos a largo plazo, pero muchas pueden ser aplicadas en talleres y otros ambientes de rango libre:

Pida a estudiantes que te envíen un correo electrónico antes del taller para explicar cómo creen que el entrenamiento los/las puede ayudar a lograr sus metas.

Revisa tus notas para asegurarte de que sean libres de pronombres de género, incluyan nombres diversos culturalmente, etc.

Enfatiza que lo que importa es la velocidad a la que están aprendiendo, no las ventajas o desventajas que tenían cuando comenzaron.

Fomenta la programación en pareja, pero demuestralo primero para que los/las estudiantes entiendan las funciones de conductor/a y navegador/a.

Mitiga activamente el comportamiento que puede resultar intimidante para algunos/as estudiantes por ejemplo uso de jerga o "preguntas" que se hacen para mostrar conocimiento.

Una forma de apoyar a estudiantes de grupos marginados es que las personas se inscriban en los talleres en grupos en lugar de individualmente. En esa manera, todos en el sala saben por adelantado que estarán con personas en la que confían, lo que aumenta la probabilidad de que realmente vengan. También ayuda después del taller: si las personas vienen con sus amistades o colegas, pueden trabajar juntos para utilizar lo que aprendieron.

Más fundamentalmente, los autores de lecciones deben considerar la situación completa de cada persona. Por ejemplo, [DiSa2014a] consiguió que el 65 % de los hombres afroamericanos en un programa de prueba de juegos pasó a estudiar computación. en parte porque el aspecto de juego del programa era algo que sus compañeros respetaban. [Lach2018] exploró dos estrategias generales para crear contenido inclusivo y los riesgos asociados con ellas:

Representación comunitaria resalta las identidades sociales, las historias, y las redes comunitarias de los/las estudiantes utilizando mentores extracurriculares o modelos a seguir de los vecindarios de los/las estudiantes, o actividades que utilizan narrativas e historias comunitarias como base para un proyecto de computación. El riesgo mas grande de este enfoque es la poca profundidad, p.ej. utilizando computadoras para construir presentaciones con diapositivas en lugar de hacer cualquier computación real.

Integración computacional incorpora ideas de la comunidad de los/las estudiantes, tales como reproducir diseños de gráficos indígenas en un ambiente de programación visual. El riesgo más grande aquí es apropiación cultural, p.ej. utilizando prácticas sin reconocer los orígenes.

En caso de duda, pregunta a tus estudiantes y miembros de la comunidad que creen que deberías hacer. Volvemos a esto en Chapter 13.

Conducta como accesibilidad

Dijimos en Section 9.1 que las clases deberían hacer un cumplir un Código

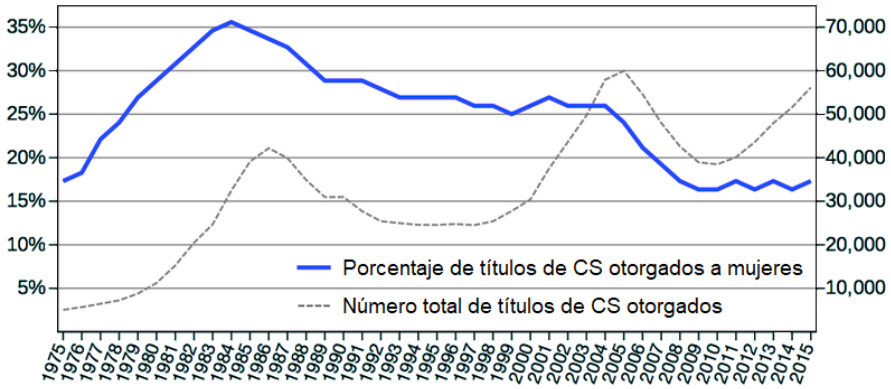

Figura 10.3: Títulos otorgados y matrícula femenina

de Conducta como el de Appendix B. Esta es una forma de accesibilidad: mientras que los subtitulos hacen que el video sea accesible a personas con discapacidades auditivas, un Código de Conducta hace que lecciones sean accesibles a personas que de otro modo serían marginadas.

Pasando el modelo del déficit

Dependiendo de en quien confies, solo 12–18 % de las personas que obtienen un título en ciencias de la computación son mujeres, que es menos de la mitad del porcentaje observado a mediados de la década de 1980 (Figure 10.3, de [Robe2017]). Y los países occidentales son los únicos que tienen un porcentaje tan bajo de mujeres en la computación; las mujeres siguen siendo a menudo 30–40 % de los estudiantes de ciencias de la computación en otras partes [Galp2002; Varm2015].

Dado que es poco probable que las mujeres hayan cambiado drásticamente en los últimos 30 años, tenemos que buscar causas estructurales para comprender que salió mal y cómo solucionarlo. Una explicación es la manera en que computadoras domésticas se comercializaron como "juguetes para niños" a partir de la década de 1980 [Marg2003]; otra es la manera en la que los departamentos de ciencias de la computación respondieron al crecimiento explosivo de la matrícula en la década de 1980 y nuevamente en la de 2000 cambiando los requisitos de admisión [Robe2017]. Ninguno de estos factores puede parecer dramático para las personas que se ven afectadas por ellos, pero actúan como el goteo constante del agua sobre una piedra: a medida que pasa el tiempo, erosionan la motivación y, con ella, la participación.

El primer paso y el más importante para solucionar esto es dejar de pensar en términos de un "tubería con fugas" [Mill2015]. Más generalmente, tenemos que superar el **modelo deficitario**, es decir, dejar de pensar que los miembros de un grupo subrepresentado carecen de algo y por lo tanto son responsables de no salir adelante. Creyendo esto coloca la carga en las personas que ya tienen que hacer un trabajo adi-

cional para superar las desigualdades estructurales y (no por casualidad) da a quienes benefician de los acuerdos actuales una excusa para no mirarse con mucho cuidado.

Reescritura del historial

[Abba2012] describe las carreras y logros de las mujeres que le dieron forma a la historia temprana de la computación, pero que con demasiada frecuencia han sido eliminadas de ella [Ensm2003; Ensm2012] describe como la programación pasó de ser una profesión femenina a una masculina en la década de 1960. mientras [Hick2018] examina como Gran Bretaña perdió su dominio inicial en la computación discriminando sistemáticamente en contra de sus trabajadores más cualificados: las mujeres. (Vea [Milt2018] para obtener una reseña de los tres libros.) Hablar de esta historia hace que algunos hombres en computación se sientan incómodos; en mi opinión, esa es una buena razón para hacerlo.

Misoginia en los videojuegos, el uso de "encaje cultural" en la contratación para excusar prejuicios conscientes o inconscientes, una cultura de silencio en torno al acoso, y la creciente desigualdad en la sociedad que produce privilegios preparatorios no son culpa de una persona en particular, pero solucionarlos es responsabilidad de todos. Como docente, tienes mas poder que la mayoría; este taller[17] tiene excelente consejos prácticos sobre cómo ser un/a buen/a aliado/a, y su consejo probablemente es más importante que cualquier cosa que te enseñe este libro sobre la enseñanza.

10.5. Ejercicios

Tareas auténticas (pares/15')

1. En pares, enumeren media docena de cosas que hicieron esta semana que utilizan las habilidades que enseñan.

2. Coloquen sus artículos en una cuadrícula de 2x2 de "tiempo para dominar" y "utilidad". ¿Dónde están de acuerdo y en desacuerdo?

Necesidades básicas (toda la clase/10')

Paloma Medina identifica seis necesidades básicas[18] para las personas en el trabajo: pertenencia, mejoramiento (es decir progresando), elección, igualdad, previsibilidad, e importancia. Luego de leer sus descripciones de cada una, ordénalas de mayor a menor importancia para ti personalmente, luego compara la clasificación con tus compañeros/as ¿Cómo crees que tu clasificación se compara con la de tus estudiantes?

[17] https://frameshiftconsulting.com/ally-skills-workshop/
[18] https://www.palomamedina.com/biceps

Implementa una estrategia para la inclusión (individual/5')

Escoja una actividad o cambio en práctica de [Lee2017] en la que te gustaría trabajar. Pon un recordatorio en tu calendario de aqui a tres meses para preguntarte si has hecho algo al respecto.

Después de los hechos (piensa-empareja-comparta/20)

1. Piensa en un curso que has tomado en el pasado e identifica una cosa que el/la docente hizo que te desmotivó. Toma notas de lo que se pudo hacer después para corregir la situación.

2. Emparéjate con tu vecino y compara cuentos, y luego agrega tus comentarios a un conjunto de notas compartidas por toda la clase.

3. Revisa lo comentarios en el conjunto de notas como grupo. Resalta y analiza algunas de las cosas que podrían haberse hecho de manera diferente.

4. ¿Crees que haciendo esto te ayudará a manejar situaciones parecidas en el futuro?

Camina la ruta (toda la clase/15')

Encuentra punto de partida de transporte público más cercano a tu edificio y camina desde allí a tu oficina y luego al baño más cercano, tomando notas de las cosas que crees que serían difíciles para alguien con dificultades de movilidad. Ahora toma prestada una silla de ruedas. ¿Qué tan completa fue tu lista de ejercicios? ¿Y te diste cuenta que la primera oración de este ejercicio asumía que podías caminar?

¿Quién decide? (toda la clase/15')

En [Litt2004], Kenneth Wesson escribió, "Si los niños de los barrios marginales pobres superan sistemáticamente a los niños de hogares suburbanos ricos en las pruebas estandarizadas, ¿alguien es lo suficientemente ingenuo como para creer que todavía insistiríamos en usar estas pruebas como indicadores de éxito?" Lea este articulo[19] de Cameron Cottrill, y luego describa un ejemplo de tu propia experiencia de evaluaciones "objetivas" que reforzaron el status quo.

Estereotipos comunes (pares/10')

Algunas personas todavía dicen, "Es tan simple que incluso tu abuela podría usarlo." En pares, enumera otras dos o tres frases que refuerzan estereotipos sobre la computación.

[19] https://mobile.nytimes.com/2016/04/10/upshot/why-talented-black-and-hispanic-students-can-go-undiscovered.html

No ser un idiota (individual/15')

Este articulo corto[20]

de Gary Bernhardt reescribe un mensaje innecesariamente hostil para ser menos grosero. Utilizándolo como un modelo, consiga algo desagradable en Stack Overflow[21] o en algún otro foro público de discusión. y reescríbelo para que sea más inclusivo.

Salvar la cara (individual/10')

¿Alguno de tus estudiantes esperados se avergonzaría de admitir que todavía no saben algunas de las cosas que quieres enseñar? Si es asi, ¿cómo puedes ayudarles a salvar la cara?

Juguetes infantiles (toda la clase/15')

[Cutt2017] encuestó a usuarios adultos de computadoras acerca de sus actividades infantiles y consiguió que la correlación más fuerte entre la confianza y el uso de la computadora se basa en leer por uno/a mismo/a y jugar con juguetes de construcción como Lego que no tienen partes móviles. Encuesta la clase y observe en qué otras actividades participan las personas, luego busque estas actividades en línea. ¿Qué grado de género tienen las descripciones y las propagandas para ellas? ¿Qué efecto crees que tiene esto?

Accesibilidad de la lección (pares/30')

En pares, escoja una lección cuyos materiales están disponibles en línea e independientemente clasifíquela de acuerdo a lo que se debe y no se debe hacer en estos afiches[22]. ¿Dónde estuvieron de acuerdo tú y tu pareja? ¿Dónde estuvieron en desacuerdo? ¿Qué tan bien fue la lección para cada una de las seis categorías de usuarios?

Siguiendo el ciclo (grupos pequeños/15')

[Coco2018] sigue un patrón deprimentemente común en que las buenas intenciones se ven socavadas por el liderazgo de una organización que no está dispuesto a cambiar realmente. Trabajando en grupos de 4–6, escriba textos o correos electrónicos breves que imaginas que cada una de las partes involucradas enviaría a la otra en cada etapa de este ciclo.

[20] https://www.destroyallsoftware.com/blog/2018/a-case-study-in-not-being-a-jerk-in-open-source
[21] https://stackoverflow.com/
[22] https://accessibility.blog.gov.uk/2016/09/02/dos-and-donts-on-designing-for-accessibility/

¿Qué es lo peor que podría pasar? (grupos pequeños/5')

A través de los años, Se me prendió fuego un proyector, una estudiante empezó el trabajo de parto, y una pelea estalló en clase. Me he caído del escenario dos veces, me he quedado dormido en una de mis propias charlas, y he tenido muchos chistes que fracasaron.

En grupos pequeños, haga una lista de las peores cosas que te han pasado mientras enseñabas, y luego compártelas con la clase. Guarda la lista para recordarte más tarde que no importa que tan mala sea la clase, al menos nada de *eso* sucedió.

Revisa

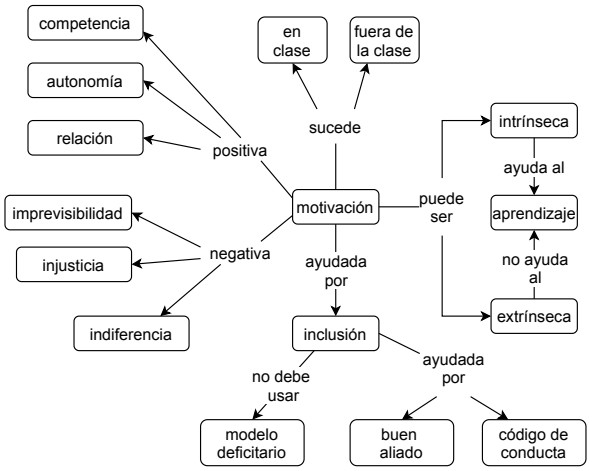

Figura 10.4: Concepts: Motivación

11

Enseñar Online

> Si usas robots para enseñar, les enseñas a las personas a ser robots.
> — atribuido a varias personas

La tecnología ha cambiado la enseñanza y el aprendizaje muchas veces. Antes de que se introdujeran los pizarrones en las escuelas a principios del siglo XIX, no había forma de que las/os docentes compartieran un ejemplo improvisado, un diagrama, o hacer ejercicio con toda una clase a la vez. Barato, de confianza, fácil de usar, y flexible, los pizarrones permitieron a las/os docentes hacer cosas rápidamente y a gran escala que antes solo habían podido hacer lentamente y poco a poco. De forma similar, las cámaras de video portátiles revolucionaron el entrenamiento atlético; al igual que las grabadoras revolucionaron la enseñanza musical una década antes.

Muchas de las personas que introducen Internet en las aulas no conocen esta historia, y no se dan cuenta de que el suyo es solo el último de una larga serie de intentos[1] de utilizar máquinas para enseñar [Watt2014]. Desde la imprenta pasando por la radio y la televisión a computadoras de escritorio y dispositivos móviles, cada nueva forma de compartir conocimientos ha producido una ola de optimistas agresivos que creen que la educación no funciona y que la tecnología puede arreglarlo.

Sin embargo, las/os defensoras/es más acérrimos de la tecnología de la educación a menudo han sabido menos sobre "educación" que sobre "tecnología", y detrás de su retórica, muchos han sido impulsados más por la perspectiva de ganancias que por el deseo de empoderar a las/os estudiantes.

El debate actual a menudo se enturbia al confundir "en línea" con "automatizado". Corre bien, una docena de personas que resuelven un problema en un chat de video se siente como cualquier otra discusión en grupos pequeños. Por el contrario, un escuadrón de ayudantes de enseñanza que califican cientos de trabajos con una rúbrica inflexible bien podría ser una colección de scripts de Perl. Por lo tanto, este capítulo comienza con la instrucción en línea totalmente automatizada, usando videos grabados y ejercicios calificados automáticamente, luego explora algunos modelos híbridos alternativos.

[1] http://teachingmachin.es/timeline.html

11.1. MOOCs

El esfuerzo de más alto perfil para reinventar la educación usando Internet son los **Cursos masivos en línea (en inglés *Massive Open Online Course***), o *MOOC*, por sus siglas en Inglés. El término fue inventado por David Cormier en 2008 para describir un curso organizado por George Siemens y Stephen Downes. Ese curso se basó en una visión **conectivista** del aprendizaje, que sostiene que el conocimiento se distribuye y el aprendizaje es el proceso de encontrar, crear y podar conexiones.

El término "MOOC" fue rápidamente co-optado por las/os creadores de cursos que se parecían más al modelo de disertación de un aula tradicional, con la/el docente como el centro definiendo los objetivos y las/os estudiantes vistos como recipientes o replicadores de conocimientos. Las clases que utilizan el modelo conectivista original se suelen denominar "cMOOCs," mientras que las clases que centralizan el control se llaman "xMOOCs." (A este último también se le llama a veces un " MESS " (la palabra *mess* significa *lío* en Inglés), por las siglas Sabio Masivamente Realzado en el Escenario (*Massively Enhanced Sage on the Stage*, por sus siglas en Inglés.))

Cinco años atrás, no se podía caminar por los campus de las universidades más grandes sin escuchar a alguien hablando sobre como los MOOCs revolucionarian la educación, la destruirían, o posiblemente ambas cosas. Los MOOCs le darían a las/os estudiantes acceso a un gran abanico de cursos y les permitirían trabajar cuando les fuera conveniente en lugar de acomodar su aprendizaje a la agenda de otra persona.

Pero los MOOC no han sido tan efectivos como predijeron sus defensores más entusiastas [Ubel2017]. Una razón es que el contenido grabado es ineficaz para muchas/os principiantes porque no puede aclarar sus conceptos erróneos individuales(Chapter 2): si no entienden una explicación la primera vez, por lo general, no hay otra diferente para ofrecer. Otra razón es que la evaluación automatizada necesaria para lograr lo "masivo" en MOOC solo funciona bien en los niveles más bajos de la taxonomía de Bloom(Section 6.2). Ahora también está claro que las/os estudiantes tienen que soportar mucho más la carga de mantenerse concentrados en un MOOC, que la impersonalidad de trabajar en línea puede fomentar un comportamiento descortés y desmotivar a las personas, y que "disponible para todos" significa en realidad "disponible para todo el mundo lo suficientemente pudiente como para tener Internet de alta velocidad y mucho tiempo libre".

[Marg2015] examinó 76 MOOCs sobre varios temas y descubrió que si bien la organización y presentación del material fue buena, la calidad del diseño de las lecciones fue deficiente. Mas cerca de casa, [Kim2017] estudió treinta tutoriales on-line y populares sobre programación, y descubrió que en gran medida enseñaban el mismo contenido de la misma manera: de abajo hacia arriba, comenzando con conceptos de programación de bajo nivel y avanzando hacia metas de alto nivel. La mayoría requirió que las/os estudiantes escribieran programas y proporcionaron algún tipo de retroalimentación inmediata, pero esta retroalimentación fue típicamente muy superficial. Pocos explican cuándo y por qué los conceptos son útiles (es decir, no mostraron cómo transferir conocimientos), o proporcionaron orientación para erro-

res comunes y aparte de una diferenciación rudimentaria basada en la edad, ninguna lección se personalizaba basada en la experiencia previa en programación o en los objetivos de la/el estudiante.

Aprendizaje personalizado

Pocos términos han sido utilizados y abusados de tantas formas como ***aprendizaje personalizado****. Para la mayoría de los defensores de la educacuón con tecnología, significa ajustar dinámicamente el ritmo de las lecciones en función del rendimiento de la/el estudiante, de modo que si alguien responde varias preguntas seguidas correctamente, la computadora omitirá algunas de las preguntas siguientes.*

Hacer esto puede producir modestas mejoras[2]*, pero se puede hacer mejor. Por ejemplo, si muchas/os estudiantes encuentran difícil un tema en particular, la/el docente puede preparar múltiples explicaciones alternativas de ese punto en lugar de acelerar un solo camino. De esa manera, si una explicación no resuena, otras están disponibles. Sin embargo, esto requiere mucho más trabajo de diseño por parte de la/el docente, que puede ser la razón por la que no ha demostrado su popularidad. Incluso si funciona, es probable que los efectos sean mucho menores de lo que creen algunos de sus defensores. Una/un buena/buen docente hace una diferencia de 0.1 a 0.15 desviaciones estándar en el desempeño de fin de año en la escuela primaria [Chet2014] (ver este artículo*[3] *para un breve resumen). No es realista creer que cualquier tipo de automatización pueda superar esto en el corto plazo.*

Entonces, ¿cómo *debe* ser usada la internet para enseñar y aprender habilidades tecnológicas? Sus pros y contras son:

Los y las estudiantes pueden acceder a más lecciones y más rapido que nunca antes Previsto, por supuesto, que un motor de búsqueda considere que vale la pena indexar esas lecciones, que su proveedor de servicios de Internet y el gobierno no lo bloqueen, y que la verdad no se ahoga en un mar de desinformación que agota la atención.

Los y las estudiantes pueden acceder a *mejores* lecciones que nunca antes, a menos que estén siendo dirigidas/os hacia material de segunda categoría, para redistribuir la riqueza de las personas que no tienen, a las personas que si tienen [McMi2017]. También vale la pena recordar que la escasez aumenta el valor percibido, para que la educación en línea sea más barata se convertirá cada vez más en lo que todo el mundo quiere para las/os hijas/os de otra persona.

Los estudiantes también pueden acceder a muchos más contactos que nunca. Pero solo si esas/os estudiantes realmente tienen acceso a la tecnología requerida, puede permitirse usarlo, y no están fuera de línea por acoso o marginadas/os porque no se ajustan a las normas sociales de cualquier grupo que lleve la voz cantante.

[2] https://www.rand.org/pubs/research_briefs/RB9994.html

[3] http://educationnext.org/in-schools-teacher-quality-matters-most-coleman/

En la práctica, la mayoría de las/os usuarias/os de MOOC provienen de entornos seguros y acomodados [Hans2015].

Los profesores pueden obtener información mucho más detallada sobre cómo trabajan las/os estudiantes. Siempre que las/os estudiantes hagan cosas que sean susceptibles de análisis automatizado a gran escala y no se opongan a la vigilancia en el aula o no son lo suficientemente poderosas/os como para que sus objeciones importen.

[Marg2015; Mill2016a; Nils2017] describe formas de acentuar los aspectos positivos en la lista anterior evitando los negativos:

Hacer que los plazos sean frecuentes y bien publicitados, y aplíquelos para que las/os estudiantes entren en ritmo de trabajo.

Mantener al mínimo las actividades sincronizadas de todas las clases, como conferencias en vivo para que las personas no se pierdan cosas debido a conflictos de agenda.

Hacer que las/os estudiantes contribuyan al conocimiento colectivo, ej. tomar notas compartidas (Section 9.7), servir como escribas en el aula, o contribuir con problemas a conjuntos de problemas compartidos (Section 5.3).

Animar o solicita a las/os estudiantes que realicen parte de su trabajo en grupos pequeños que *si* tienen actividades sincrónicas en linea como una discusión semanal. Esto ayuda a las/os estudiantes a mantenerse comprometidos/as y motivados/as sin crear demasiados problemas de agenda. (Ver Appendix E para obtener algunos consejos sobre cómo hacer que estas discusiones sean justas y productivas.)

Crear, publicitar y hacer cumplir un código de conducta. para que todos puedan participar en los debates en línea (Section 9.1).

Utilizar muchas lecciones breves en lugar de pocos fragmentos largos al estilo conferencias. para minimizar la carga cognitiva y brindar muchas oportunidades para la evaluación formativa. Esto también ayuda con el mantenimiento: si todos tus videos son cortos, simplemente puedes volver a grabar cualquiera que necesite actualización, lo que a menudo es más barato que intentar arreglar los más largos.

Utilizar el video para fomentar la participación en lugar de instruir. Dejando de lado las discapacidades (Section 10.3), las/os estudiantes pueden leer más rápido de lo que tú puedes hablar. La excepción a esta regla es que el video es la mejor manera de enseñar verbos(acciones): videos de pantallas cortos que muestran cómo usar un editor, cómo recorrer el código en un depurador, y así sucesivamente, son más eficaces que las capturas de pantalla con texto.

Identificar y aclarar tempranamente conceptos erróneos. Si los datos muestran que las/os estudiantes tienen dificultades con algunas partes de una lección, crear explicaciones alternativas de esos partes y ejercicios adicionales para que practiquen.

Todo esto tiene que ser implementado de alguna manera, lo que significa que

necesitas alguna clase de plataforma de enseñanza. Puedes utilizar tanto un **sistema de gestión del aprendizaje (*learning management system*, en inglés)** todo en uno como Moodle[4] o Sakai[5], o generar uno por ti mismo/a usando Slack[6] o Zulip[7] para el chat, Google Hangouts[8] o appear.in[9] para videoconferencias, y WordPress[10], Google Docs[11], o cualquier número de sistemas wiki para la autoría colaborativa. Si recién estás comenzando, elije lo que sea más fácil de configurar y administrar y lo que sea más familiar para tus estudiantes. Si te enfrentas a una elección, la segunda consideración es más importante que la primera: esperas que las personas aprendan mucho en tu clase, por lo que es justo que tu aprendas a manejar las herramientas con las que se sientan más cómodas.

Montar una plataforma para el aprendizaje es necesario pero no suficiente: si quieres que tus estudiantes prosperen, necesitas crear una comunidad. Cientos de libros y presentaciones hablan sobre cómo hacer esto, pero la mayoría se basan en las experiencias personales de sus autores. [Krau2016] es una excepción bienvenida: si bien es anterior al descenso acelerado de Twitter y Facebook hacia el abuso y la desinformación, la mayoría de sus hallazgos siguen siendo relevantes. [Foge2005] también está lleno de consejo útiles sobre las comunidadesd de práctica a las que las/os estudiantes pueden esperar unirse; exploramos algunas de sus ideas en Chapter 13.

Libertad para, libertad de

El ensayo de 1958 de Isaiah Berlin "Dos conceptos de libertad[12]*" hizo una distinción entre libertad positiva, que es la capacidad de hacer algo, y libertad negativa, que es la ausencia de reglas que digan que no puedes hacerlo. Las discusiones en línea generalmente ofrecen libertad negativa (nadie te impide decir lo que piensas) pero no libertad positiva (muchas personas no pueden ser escuchadas, en realidad). Una forma de abordar esto es introducir algún tipo de limitación, como permitir que cada estudiante contribuya con un mensaje por hilo de discusión al día. Hacer esto les da a aquellas/os que tienen algo que decir la oportunidad de decirlo, mientras deja espacio para que otras/os también digan cosas.*

Otra preocupación que se tiene sobre la enseñanza en línea es la posibilidad de que las/os estudiantes hagan trampa. La deshonestidad del día a día no es más común en las clases en línea que en los entornos presenciales [Beck2014], pero la tentación de que otra persona escriba el examen final, y la dificultad de comprobar si esto realmente sucedió, es una de las razones por las que las instituciones educativas se han mostrado reacias a ofrecer créditos para las clases solamente en línea. Es posible su-

[4] http://moodle.org
[5] https://www.sakaiproject.org/
[6] http://slack.com
[7] https://zulipchat.com/
[8] http://hangouts.google.com
[9] https://appear.in/
[10] https://wordpress.org/
[11] http://docs.google.com
[12] https://en.wikipedia.org/wiki/Two_Concepts_of_Liberty

pervisar el examen a distancia, pero antes de invertir en esto, lee [Lang2013]: explora por qué y cómo las/os estudiantes hacen trampa, y cómo se pueden estructurar los cursos para evitar darles una razón para hacerlo.

11.2. Video

Una característica destacada de la mayoría de los MOOC es el uso de clases grabadas en video. Estos videos pueden ser efectivos: como se menciona en Chapter 8, una técnica de enseñanza llamada instrucción directa basado en la entrega precisa de un guión bien diseñado ha demostrado repetidamente su eficacia [Stoc2018]. Sin embargo, deben diseñarse guiones para la instrucción directa, probando, y refinado con mucho cuidado, lo que es una inversión que muchos MOOC no han querido o no han podido hacer. Hacer un pequeño cambio en una página web o en una plataforma de diapositivas solo toma unos minutos; hacer incluso un pequeño cambio en un video corto lleva una hora o más, por lo que el costo de actuar sobre la base de los comentarios puede ser insoportable para la/el docente. E incluso cuando están bien hechos los videos deben combinarse con actividades para que sean beneficiosos: [Koed2015] estima, "…el aprendizaje el beneficio de hacer … es más que seis veces que mirar o leer."

Si estás enseñanzo programación, puede usar grabaciones de pantallas en lugar de diapositivas, ya que ofrecen algunas de las mismas ventajas que la codificación en vivo (Section 8.1). [Chen2009] ofrece consejos útiles para crear y criticar grabaciones de pantallas y otros videos; Figure 11.1 (de [Chen2009]) reproduce los patrones que presenta el papel y las relaciones entre ellos. (También es un buen ejemplo de mapa conceptual (Section 3.1).)

Entonces, ¿qué hace que un video instructivo sea efectivo? [Guo2014] midió el compromiso y participación, observando cuánto tiempo las/os estudiantes vieron los videos de MOOCs, y encontró que:

- Los videos más cortos son mucho más atractivos; los videos no deben durar más de seis minutos.
- Una cabeza parlante superpuesta a las diapositivas es más atractiva que una sola voz en off.
- Los videos que se sienten personales pueden ser más atractivos que las grabaciones de estudio de alta calidad, por lo que filmar en entornos informales podría funcionar mejor que filmar en un estudio profesional por un costo menor.
- Dibujar en una tableta es más atractivo que las diapositivas de PowerPoint o las grabaciones de pantalla con código, aunque no está claro si esto se debe al movimiento y la informalidad o porque reduce la cantidad de texto en la pantalla.
- Está bien que los profesores hablen bastante rápido siempre que estén entusiasmados.

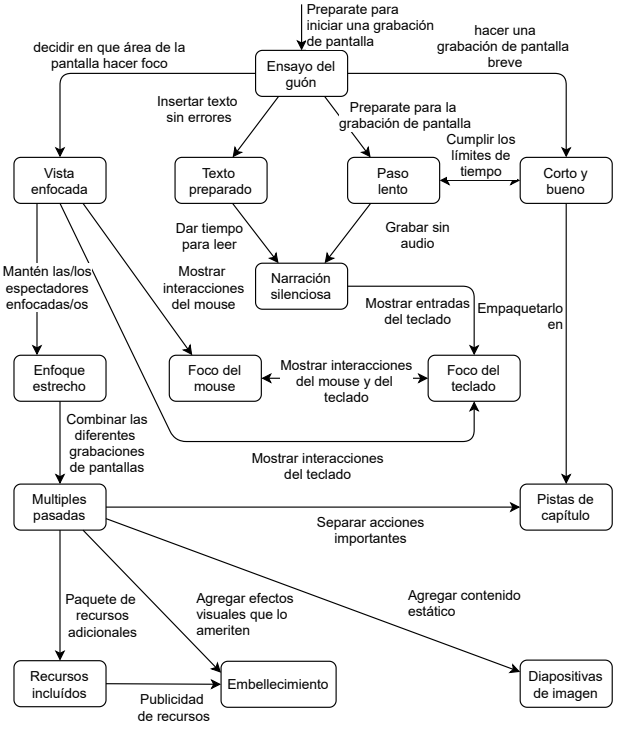

Figura 11.1: Patrones para grabaciones de pantalla

Una cosa [Guo2014] que no se abordó es el problema del huevo y la gallina: ¿las/os estudiantes encuentran cierto tipo de video atractivo porque están acostumbrados? Entonces, ¿producir más videos de ese tipo aumentará la participación simplemente debido a un ciclo de retroalimentación? ¿O estas recomendaciones reflejan algunos procesos cognitivos más profundos? Otra cosa que este documento no analizó son los resultados del aprendizaje: sabemos que las evaluaciones de las/os estudiantes de los cursos no se correlacionan con el aprendizaje [Star2014; Uttl2017], y si bien es plausible que las/os estudiantes no aprendan de las cosas que no ven, queda por demostrar que *aprenden* de las cosas que *ven*.

Estoy un poco incómodo/a

La investigación de [Guo2014]'s fue aprobada por una junta universitaria de ética en investigación, las/os estudiantes cuyos hábitos de visualización fueron monitoreados casi con certeza hicieron clic en "aceptar" en un acuerdo de términos de servicio en algún momento, y me alegra tener estos nuevos conocimientos. Por otra parte, la palabra "privacidad" no apareció en el título o en el resumen de ninguno *de las decenas de artículos o posters en la conferencia donde se presentaron estos resultados. Si puedo elegir, prefiero no saber qué tan comprometidos están los estudiantes que fomentar la vigilancia ubicua en el aula.*

Hay muchas formas diferentes de grabar lecciones en video; para saber cuáles son más eficaces, [Mull2007a] asignó 364 estudiantes de física de primer año a los tratamientos multimedia en línea de la Primera y Segunda Ley de Newton en uno de cuatro estilos:

Exposición: presentación concisa al estilo de una clase magistral.

Exposición extendida: igual que la anterior, pero con información adicional interesante.

Refutación: Exposición con conceptos erróneos comunes explícitamente declarados y refutados.

Diálogo: Discusión estudiante-docente del mismo material que en la Refutación.

Refutación y diálogo produjeron los mayores beneficios de aprendizaje en comparación con la exposición; las/os estudiantes con pocos conocimientos previos se beneficiaron más, y aquellas/os con un alto conocimiento previo no estaban en desventaja. De nuevo, esto destaca la importancia de abordar directamente los conceptos erróneos de las/os estudiantes. No sólo, le digas a las personas lo que *es*: díles también qué *no es* y por qué no.

11.3. Modelos híbridos

La enseñanza totalmente automatizada es solo una forma de utilizar la web en la enseñanza. En la práctica, casi todo el aprendizaje en las sociedades prósperas tiene actualmente un componente en línea, ya sea oficialmente o a través de canales de comunicación persona a persona y búsquedas subrepticias de respuestas a preguntas sobre la tarea. La combinación de instrucción en vivo y automatizada permite a los maestros usar las fortalezas de ambos. En un aula tradicional, la/el docente puede responder preguntas de inmediato, pero las/os estudiantes necesitan días o semanas para recibir comentarios sobre sus ejercicios de programación. En línea, un/a estudiante puede tardar más en obtener una respuesta, pero pueden obtener comentarios inmediatos sobre el código que programó (al menos para ese tipo de ejercicios podemos calificar automáticamente).

Otra diferencia es que los ejercicios en línea deben ser más detallados porque tienen que anticiparse a las preguntas de las/os estudiantes. Encuentro que las lecciones en persona comienzan con la intersección de lo que todos necesitan saber y se expanden a pedido, mientras que las lecciones en línea deben incluir la unión de lo que todas las personas necesitan saber porque la/el docente no está ahí para hacer esta expansión.

En realidad, la distinción entre online y presencial ahora es menos importante para la mayoría de las personas que la distinción entre síncrono y asíncrono: ¿las/os docentes y las/os estudiantes interactúan en tiempo real? ¿O su comunicación se extiende e intercala con otras actividades? En persona casi siempre será sincrónico, pero en línea es, cada vez más, una mezcla de ambos:

> Creo que nuestros nietos probablemente considerarán, la distinción que hacemos entre lo que llamamos el mundo real y lo que ellos consideran simplemente el mundo, como la cosa más pintoresca e incomprensible sobre nosotros.
> — William Gibson

La implementación más popular de este futuro combinado hoy es el **aula invertida**, en el que las/os estudiantes ven lecciones grabadas por su cuenta y el tiempo de la clase se utiliza para discutir y resolver conjuntos de problemas. Descrito originalmente en [King1993], la idea se popularizó como parte de la instrucción entre pares (Section 9.2) y se ha estudiado intensamente durante la última década. Por ejemplo, [Camp2016] comparó a las/os estudiantes que tomaron una clase de introducción a la informática en línea con los que la tomaron en un aula invertida. La finalización de ejercicios de práctica (sin marcar) se correlacionó con los puntajes de los exámenes para ambos, pero la tasa de finalización de los ejercicios de ensayo por parte de los estudiantes en línea fue significativamente más baja que las tasas de asistencia a clase para los estudiantes presenciales.

Pero si hay grabaciones disponibles, ¿Seguirán asisitiendo las/os estudiantes a las clases para hacer ejercicios de práctica? [Nord2017] examinó el impacto de las grabaciones en la asistencia a las clases y al desempeño de las/os estudiantes en diferentes

niveles. En la mayoría de los casos, el estudio no encontró consecuencias negativas al hecho de hacer disponibles las grabaciones; en particular, las/os estudiantes no se saltaron las clases cuando las grabaciones estaban disponibles (al menos, no más de lo que suelen faltar). Los beneficios de proporcionar grabaciones fueron mayores para las/os estudiantes al principio de sus carreras, pero disminuye a medida que las/os estudiantes maduran.

Otro modelo híbrido lleva la vida en línea al aula. Tomar notas juntos es un primer paso (Section 9.7); agrupando respuestas a preguntas de opción múltiple en tiempo real usando herramientas como Pear Deck[13] y Socrative[14] es otra. Si la clase es pequeña — digamos, de una docena a quince personas — también puedes hacer que todos/as las/os estudiantes se unan a una videoconferencia para que puedan compartir la pantalla con la/el docente. Esto les permite mostrar su trabajo (o sus problemas) a toda la clase. sin tener que conectar su computadora portátil al proyector. las/os estudiantes también pueden usar el chat en la videollamada para hacer preguntas para la/el docente; en mi experiencia, la mayoría de las preguntas serán respondidas por sus compañeros/as, y la/el docente puede encargarse del resto cuando lleguen a un momento natural de descanso. Este modelo ayuda a nivelar "la cancha´´ para las/os estudiantes remotos: si alguien no puede asistir a clase por razones de salud o por compromisos familiares o laborales, todavía puede participar casi en pie de igualdad, si todo el mundo está acostumbrado a colaborar online y en tiempo real.

También he impartido clases utilizando instrucción remota en tiempo real, en el que las/os estudiantes comparten la ubicación en 2 a 6 sitios difrentes, con ayudantes presentes mientras enseñaba vía videoconferencia (Section C.1). Esto escala bien, ahorra en gastos de viaje, y permite el uso de técnicas como la programación por pares (Section 9.6). Lo que *no* funciona, es tener un grupo en persona y uno o más grupos de forma remota: aún con la mejor voluntad del mundo, las/os participantes en persona reciben mucha más atención.

11.4. Participación on-line

[Nuth2007] descubrió que hay tres mundos superpuestos en cada aula: lo público (lo que dice y hace la/el docente), lo social (interacciones entre pares, entre los estudiantes), y lo privado (dentro de la cabeza de cada estudiante). De estos, el más importante suele ser el social: las/os estudiantes captan tanto a través de las señales de sus compañeros/as como de la instrucción formal.

Por lo tanto, la clave para hacer efectiva cualquier forma de enseñanza en línea es facilitar las interacciones entre pares. Para ayudar a lograr esto, los cursos casi siempre tienen algún tipo de foro de discusión. [Mill2016a] observó que las/os estudiantes los utilizan de formas muy diferentes:

[13] https://www.peardeck.com/
[14] https://socrative.com/

> . . . es muy poco probable que las personas procrastinadoras participen en foros de discusión en línea, y esta participación reducida, a su vez, se correlaciona con peores calificaciones. Una posible explicación de esta correlación es que las personas procrastinadoras son especialmente reacias a unirse, una vez que la discusión está en curso, quizás porque les preocupa ser percibidas como recién llegadas en una conversación establecida. Esta aversión a participar tarde provoca que se pierdan los beneficios importantes de aprendizaje y motivación de la interacción entre pares.

[Vell2017] analiza publicaciones en foros de discusión de 395 estudiantes de CS2 en dos universidades dividiéndolos en cuatro categorías:

Activos/as: solicitud de ayuda que no muestra razonamiento y no muestra lo que la/el estudiante ya ha probado o ya sabe.

Constructivo/a: refleja el razonamiento de las/os estudiantes o intenta construir una solución al problema.

Logístico/a: políticas del curso, horarios, envío de tareas, etc.

Aclaración de contenido: solicitud de información adicional que no revela el propio pensamiento dla/el estudiante.

Descubrieron que dominaban las preguntas constructivas y logísticas, y que las preguntas constructivas se correlacionan con las calificaciones. También encontraron que las/os estudiantes rara vez hacen más de una pregunta activa en un curso, y que estas *no* se correlacionan con las calificaciones. Si bien esto es decepcionante, saberlo ayuda a establecer las expectativas de las/os docentes: si bien es posible que todos deseemos que nuestros cursos tengan comunidades en línea animadas, tenemos que aceptar que la mayoría no lo hará, o que la mayor parte de la discusión de estudiante a estudiante se llevará a cabo a través de canales que ya están usando de la que no podemos ser parte.

Cooperación

[Gull2004] describe un concurso de codificación en línea que combina colaboración y competencia. El concurso comienza cuando se publica una descripción del problema junto con una solución correcta pero ineficiente. Cuando acaba, la persona ganadora es quien ha hecho la mayor contribución global para mejorar el rendimiento de la solución global. Todas las presentaciones están abiertas, para que los participantes puedan ver el trabajo de los demás y tomar prestadas ideas entre ellos. Como muestra el trabajo, la solución final es casi siempre un híbrido que utiliza ideas de muchas personas.

[Batt2018] describió una variación a pequeña escala de esto en una clase de introducción a la informática. En la etapa uno, cada estudiante presentó un proyecto de programación de forma individual. En la etapa dos, las/os estudiantes trabajaron en parejas para crear una solución mejorada al mismo problema. La evaluación indica que los proyectos de dos

etapas tienden a mejorar la comprensión de las/os estudiantes y que disfrutaron del proceso. Proyectos como estos, no solo mejoran la participación, también brindan a las/os estudiantes más experiencia sobre la base del código de otra persona.

La discusión no es la única forma de lograr que las/os estudiantes trabajen juntos en línea. [Pare2008] y [Kulk2013] reportan experimentos en el que las/os estudiantes califican el trabajo de las/os demás, y las calificaciones que asignan se comparan con las calificaciones otorgadas por profesores/as, asistentes de posgrado u otros expertos/as. Ambos trabajos encontraron que las calificaciones asignadas por las/os estudiantes coincidían con las calificaciones asignadas por las/os expertos/as con tanta frecuencia como las calificaciones de las/os expertos/as coincidían entre sí, y que unos sencillos pasos (como filtrar respuestas obviamente no consideradas o estructurar rúbricas) disminuyó aún más el desacuerdo. Como se discute en Section 5.3, la colusión y el sesgo *no* son factores importantes en la calificación de pares.

Confía, pero educa

La forma más común de medir la validez de los comentarios es comparar las calificaciones de las/os estudiantes con las calificaciones de las/os expertas/os, pero revisión por pares calibrada (Section 5.3) puede ser igualmente efectiva. Antes de pedir a las/os estudiantes que califiquen el trabajo del resto, se les pide que califiquen muestras y comparen sus resultados con las calificaciones asignadas por la/el docente. Una vez que ambas calificaciones se alinean, se le permite al/la estudiante comenzar a calificar a sus compañeros/as. Dado que la lectura crítica es una forma eficaz de aprender, este resultado puede apuntar a un futuro en el que las/os estudiantes utilicen la tecnología para emitir juicios, en lugar de ser juzgado por la tecnología.

Una técnica que definitivamente vamos a ver más en los próximos años es la transmisión en línea de sesiones de programación en vivo [Raj2018; Haar2017]. Esto tiene la mayoría de los beneficios discutidos en Section 8.1, y cuando se combina con la toma de notas colaborativa (Section 9.7) puede ser una aproximación cercana a una experiencia en clase.

Mirando aún más adelante, [Ijss2000] identificó cuatro niveles de presencia en línea, desde el realismo (no podemos notar la diferencia), la inmersión (nos olvidamos de la diferencia), la participación (estamos comprometidos con la clase, pero somos conscientes de la diferencia) a la suspensión de la incredulidad (estamos haciendo la mayor parte del trabajo). La diferencia crucial que distinguen es la presencia física, como la sensación de estar en algún lugar, y presencia social, que es la sensación de estar con los demás. Esta última es más importante en la mayoría de situaciones de aprendizaje, y otra vez, podemos fomentarlo utilizando la tecnología diaria de las/os estudiantes en el aula. Por ejemplo, [Deb2018] descubrió que la retroalimentación en tiempo real sobre los ejercicios en clase, usando los dispositivos móviles de las/os estudiantes, mejoró la retención de conceptos y la participación de las/os estudiantes, al tiempo que redujo las tasas de fracaso.

La enseñanza en línea y asincrónica aún está en pañales. Los MOOC centralizados pueden llegar a ser un callejón sin salida evolutivo, pero aún quedan muchos otros modelos prometedores por explorar. En particular, [Broo2016] describe cincuenta formas en que los grupos pueden discutir cosas de manera productiva, y solo unos pocos son ampliamente conocidos o implementados en línea. Si vamos a donde están tecnológicamente nuestros/as estudiantes, en lugar de pedirles que vengan a nosotros, podemos terminar aprendiendo tanto como ellas/os.

11.5. Ejercicios

Video bidireccional (pares/10')

Grabate en un video de 2 a 3 haciendo algo, luego intercambia el video con una/un compañera/o para que cada uno pueda ver el video de la otra persona a una velocidad 4x. ¿Qué tan fácil es seguir lo que está pasando? ¿Te perdiste de algo?

Puntos de vista (individual/10')

De acuerdo a [Irib2009], diferentes disciplinas se centran en diferentes factores que afectan el éxito o no de las comunidades en línea:

Negocios: fidelización de clientes, gestión de marca, motivación extrínseca.

Psicología: sentido de comunidad, motivación intrínseca.

Sociología: identidad de grupo, comunidad física, capital social, acción colectiva.

Ciencias de la computación: implementación tecnológica.

¿Cuál de estas perspectivas se corresponde más con la tuya? ¿Con cuál estás menos alineada/o?

Ayudar o dañar (grupos pequeños/30')

Este artículo de Susan Dynarski's en el *New York Times*[15] explica cómo y por qué las escuelas están colocando a las/os estudiantes que no aprueban los cursos presenciales en cursos en línea y cómo esto los prepara para un fracaso aún mayor. Lea el artículo y luego:

1. En pequeños grupos, piensen en dos o tres cosas que las escuelas podrían hacer para compensar estos efectos negativos y crear estimaciones aproximadas de sus costos por estudiante.

[15] https://www.nytimes.com/2018/01/19/business/online-courses-are-harming-the-students-who-need-the-most-help.html

2. Compara tus sugerencias y costos con los de otros grupos. ¿Cuántos puestos de enseñanza a tiempo completo cree que deberían eliminarse con el fin de liberar recursos para implementar las ideas más populares para un centenar de estudiantes?

3. Como una clase, ¿Crees que sería un beneficio real para las/os estudiantes o no?

Los ejercicios de presupuestación como este son una buena manera de saber quién se toma en serio el cambio educativo. Todas las personas pueden pensar en cosas que les gustaría hacer; pero muchas menos están dispuestas a hablar sobre las compensaciones necesarias para que suceda el cambio.

12

Tipos de Ejercicios

Todo buen carpintero tiene un juego de destornilladores y todo buen docente tiene diferentes tipos de ejercicios para comprobar qué están aprendiendo realmente los alumnos y las alumnas, para ayudarlos a practicar sus nuevas habilidades, y para mantener su compromiso.

Este capítulo comienza describiendo varios tipos de ejercicios que se pueden usar para corroborar si tu forma de enseñar ha resultado efectiva.

Luego examina el estado del arte en cuanto a calificación automatizada, y culmina con la exploración de la discusión, proyectos y otros tipos importantes de trabajo que requieren una atención más humana para evaluar.

Nuestra discusión se basa parcialmente en el banco de preguntas Canterbury Question Bank[1] [Sand2013], que trae entradas para varios lenguajes de programación y temas introductorios a las ciencias de la computación.

12.1. Los clásicos

Como se discute en Section 2.1, las *preguntas de opción múltiple* (PMC) son más efectivas cuando las respuestas incorrectas sondean conceptos erróneos. Están designadas para evaluar los niveles más bajos en la taxonomía de Bloom (Section 6.2) ,pero también puede requerir que los/las estudiantes utilicen su propio juicio/criterio.

Una pregunta de opción múltiple

En qué orden ocurren las operaciones cuando la computadora evalúa la expresión `precio = agregarImpuestos(costo - descuento)`*?*

1. *resta, llamado a función, asignación*
2. *llamado a función, resta, asignación*
3. *llamado a función, luego asignación y resta simultáneamente*
4. *ninguna de las anteriores*

El segundo tipo de ejercicio de programación clásico es *códificar y ejecutar* (C&R, por sus siglas en inglés), donde el estudiante escribe un código que produce una salida especificada. Los ejercicios C&R pueden ser tan simples o complejos

[1] http://web-cat.org/questionbank/

como el docente o la docente quieran, pero cuando se usen en clase deben ser breves y deben tener solamente una o dos respuestas correctas posibles. Generalmente es suficiente con pedir a quienes se inician que calculen e impriman un solo valor o que llamen a una función específica:

Docentes experimentados frecuentemente se olvidan de cuán difícil puede ser darse cuenta de cuál parámetro van en qué lugar. Para los estudiantes y las estudiantes más avanzados, darse cuenta de cuál es la función que hay que llamar, es una actividad más atractiva y una mejor evaluación de su comprensión.

Programa & ejecuta

La variable `picture`*contiene una imagen a todo color de un archivo. Usando una función, crea una versión blanco y negro de la imagen y asígnala a una variable nueva llamada* `monochrome`*.*

Los ejercicios de escribir y ejecutar pueden ser combinados con MCQ. Por ejemplo, este MCQ sólo puede ser contestado ejecutando el comando Unix `ls`:

Combinar MCQ con Programar & ejecutar

Estás en el directorio `/home`*. ¿Cuál de los siguientes archivos está*no está *en dicho directorio?*

1. `autumn.csv`
2. `fall.csv`
3. `spring.csv`
4. `winter.csv`

Los ejercicios de C&Rs ayudan a practicar las habilidades que más quieren aprender, pero pueden ser difíciles de evaluar: puede haber muchos caminos posibles para obtener la respuesta correcta, y se genera desánimo si el sistema de evaluación rechaza el código elaborado porque no se corresponde con el del docente. Una forma de reducir cuán frecuentemente ocurre esto es evaluar solo la salida, pero esto no da feedback sobre cómo están programando. Otra manera es darles un pequeño conjunto de pruebas con el que puedan comparar su código antes de enviarlo (ya que se corre/ejecuta contra un conjunto de pruebas más completo). Hacer esto les ayuda a descubrir si han malinterpretado completamente la intención del ejercicio antes de hacer cualquier cosa que pueda afectarles la nota.

En vez de escribir código que satisface una cierta especificación, se les puede pedir a los/las estudiantes y a las estudiantes que escriban pruebas que determinen si un fragmento de código satisface una cierta especificación. Esta es una habilidad útil por sí misma, y al hacerlo puede generar en los/las estudiantes un poco más de simpatía por lo duro que trabajan sus profesores.

Invirtiendo programar & ejecutar

La función `monotonic_sum` *calcula la suma de una lista de números de una sección, en la que los valores aumentan estrictamente.*

Por ejemplo, dada la entrada `[1, 3, 3, 4, 5, 1]`*, la salida es* `[4, 12, 1]`*.*

Escribe y corre pruebas unitarias para determinar cuál de los siguientes errores está contenido en la función :

- *Considera cada número negativo como el inicio de una nueva sub-secuencia.*
- *No incluye el primer valor de cada sub-secuencia en la sub-suma.*
- *No incluye el último valor de cada sub-secuencia en la sub-suma.*
- *Solo reinicia la suma cuando los valores decrecen en vez de aumentar.*

Llenar los espacios en blanco es un refinamiento de C&R donde se le da al estudiante y a la estudiante el comienzo de un código y debe completarlo

(En la práctica, la mayoría de los ejercicios C&R son en realidad del tipo completar los espacios en blanco porque el docente o la docente proveen comentarios para recordarles a los estudiantes los pasos que deben seguir. Las preguntas de este tipo son la bases para ejemplos descoloridos; como se discute en Chapter 4, las personas principiantes encuentran este tipo de ejercicios menos intimidante que escribir todo el código desde cero, y como el docente o la docente ha provisto la mayor parte de la estructura de la respuesta, las respuestas enviadas son más predecibles y fáciles de revisar.

Completar los espacios en blanco

Completa los espacios en blanco, para que el código de abajo imprima el texto `'hat'`.

```
text = 'todo lo que es'
slice = text[____:____]
print(slice)
```

Los problemas de Parsons también evitan el problema de la "pantalla blanca del terror" y permiten que los alumnos y las alumnas se concentren en el control del flujo separadamente del vocabulario [Pars2006; Eric2015; Morr2016; Eric2017].

Existen herramientas en línea para construir y hacer Problemas de Parsons [Ihan2011], pero pueden ser emuladas (aunque torpemente) pidiendo a los/las estudiantes que reorganicen líneas de código en un editor de texto.

Problemas de Parsons

Reordena e indenta estas líneas para sumar los valores positivos en una lista. (Agrega también punto y coma en lugares apropiados.)

```
total = 0
if v > 0
total += v
for v in values
```

Dar más líneas de las que necesitan los/las estudiantes, o pedirles que reordenen algunas líneas y añadan otras más, hace que los problemas de Parsons sean significativamente más difíciles [Harm2016].

12.2. Seguir

Seguir la ejecución es lo contrario de un Problema de Parsons: dadas unas pocas líneas de código, los/as estudiantes tienen que trazar el orden en el que se ejecutan esas líneas. Esta es una habilidad esencial de depuración y una buena manera de consolidar la comprensión de los bucles, los condicionales y el orden de evaluación de las llamadas a funciones y a métodos.

La forma más fácil de implementarlo es hacer que los alumnos y las alumnas escriban una secuencia de pasos etiquetados. Hacer que elijan la secuencia correcta de un conjunto (es decir, presentarla como un MCQ) añade carga cognitiva pero sin añadir valor, ya que tienen que hacer todo el trabajo descifrar cuál es la secuencia correcta, y luego buscarla en la lista de opciones.

Seguir el orden de ejecución

En qué orden se ejecutan las líneas etiquetadas de este bloque de código?

```
A)      vals = [-1, 0, 1]
B)      inverse_sum = 0
        try:
            for v in vals:
C)              inverse_sum += 1/v
        except:
D)          pass
```

Seguir los valores es similar a trazar la ejecución, pero en lugar de listar el orden en que se ejecuta el código, el estudiante enumera los valores que una o más variables asumen a medida que el programa se ejecuta.

Una forma de implementar esto es dar al estudiante y a la estudiante una tabla cuyas columnas están etiquetadas con nombres de variables y cuyas filas estén etiquetadas con números de línea, y pedirle que complete los valores que toman las variables en esas líneas.

Seguir los valores

¿Qué valores toman `left` *y* `right` *a medida que este programa se ejecuta?*

```
A) left = 23
B) right = 6
C) while right:
D)     left, right = right, left % right
```

Line	`left`	`right`

También puedes pedir a los alumnos y a las alumnas que rastreen el código hacia atrás para averiguar cuál debe haber sido la entrada para que el código produzca un resultado determinado [Armo2008].

Estos problemas de *ejecución inversa* requieren razonamientos de búsqueda y deducción, y cuando la salida es un mensaje de error, ayudan a los/las alumnos/as a desarrollar habilidades valiosas de depuración.

Ejecución Reversa

Completa el valor numérico faltante en `values` *que causó que esta función se interrumpa.*

```
values = [ [1.0, -0.5], [3.0, 1.5], [2.5, ___] ]
runningTotal = 0.0
for (reading, scaling) in values:
    runningTotal += reading / scaling
```

Minimal fix exercises also help learners develop debugging skills. Los ejercicios de *ajuste mínimo* también ayudan a desarrollar habilidades de depuración.

Dadas unas pocas líneas de código que contienen un error, el estudiante o la estudiante deben encontrarlo y hacer un pequeño cambio para arreglarlo.

El cambio puede hacerse usando C&R, mientras que la identificación puede hacerse como una pregunta de opción múltiple.

Ajuste mínimo

Se asume que esta función comprueba si un número se encuentra dentro de un rango. Haz un pequeño cambio para que la función realmente lo haga.

```
def inside(point, lower, higher):
    if (point <= lower):
        return false
    elif (point <= higher):
        return false
```

```
    else:
        return true
```

Los ejercicios de *Tema y variación*son similares, pero se pide al alumno o a la alumna que hagan una pequeña alteración que cambie el resultado de alguna manera específica en lugar de hacer un cambio para arreglar un error. Los cambios permitidos pueden incluir cambiar el valor inicial de una variable, reemplazar una llamada de función por otra, intercambiar bucles internos y externos, o cambiar el orden de las pruebas en un condicional complejo

De nuevo, este tipo de ejercicio da a los estudiantes y a las estudiantes la oportunidad de practicar una habilidad útil para el mundo real: la forma más rápida de producir el código que necesitan es ajustando un código que ya hace algo parecido.

Tema y variaciones

Cambiar el bucle interior de la siguiente función para que llene el triángulo superior izquierdo de una imagen con un color especificado.

```
function fillTriangle(picture, color) is
    for x := 1 to picture.width do
        for y := 1 to picture.height do
            picture[x, y] = color
        end
    end
end
```

Los *Ejercicios de Modificación* son el complemento de los ejercicios de tema y variación: dado un trozo de código que funciona, el estudiante y la estudiante tienen que modificarlo de alguna manera sin cambiar la salida.

Por ejemplo, el alumno o la alumna pueden reemplazar los bucles con expresiones vectorizadas o simplificar la condición en un bucle while

Esto también es una habilidad útil para la vida real, pero como hay tantas formas de modificar el código, se necesita inspección humana para poder poner una nota.

Refactoring

Escribir una sola compresión de lista que tenga el mismo efecto que este bucle.

```
result = []
for v in values:
    if len(v) > threshold:
        result.append(v)
```

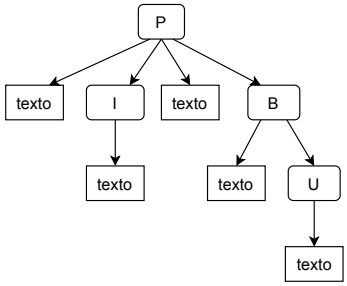

Figura 12.1: Etiquetando un diagrama

12.3. Diagramas

Hacer que los alumnos dibujen mapas conceptuales y otros diagramas, brinda una idea de cómo están pensando (Section 3.1), pero los diagramas de forma libre requieren tiempo y juicio humano para evaluarlos.

Etiquetar los diagramas, por otra parte, es pedagógicamente igual de poderoso pero mucho más fácil de escalar.

En lugar de hacer que los alumnos y las alumnas creen diagramas desde cero, hay que proporcionarles un diagrama y un conjunto de etiquetas y hacer que las coloquen en los lugares correctos. El diagrama puede ser una estructura de datos ("después de ejecutar este código, ¿qué variables apuntan a qué partes de esta estructura?"), un gráfico ("aparea cada uno de estos fragmentos de código con la parte del gráfico generado"), o el propio código ("haz coincidir cada término con un ejemplo del mismo en el programa").

Etiquetar un diagrama

Figure 12.1 muestra cómo un fragmento pequeño de HTML se representa en memoria. Pon las etiquetas 1–9 sobre los elementos del árbol para mostrar el orden de llegada cuando se hace recorrido en profundidad.

Otra forma de usar los diagramas es darles a los alumnos y a las alumnas partes del diagrama y pedirles que las ordenen correctamente.

Este es un equivalente visual de un Problema de Parsons, y se puede proporcionar tanto más o menos del esquema para ayudar con la colocación de las etiquetas como se considere apropiado.

Tengo buenos recuerdos tratando de colocar resistencias y capacitores en un circuito para obtener el voltaje correcto en un punto determinado, y he visto profesores que le dan a sus alumnas un conjunto definido de bloques de Scratch y les piden que creen un diseño particular usando sólo esos bloques.

Los *Problemas de Correspondencia* pueden pensarse como un caso especial de

etiquetado en el que el "diagrama" es una columna de texto y las etiquetas se toman de la otra columna.

En la *Correspondencia Uno-a-uno* se le da al estudiante o a la estudiante dos listas de igual longitud y se le pide que asocie los elementos correspondientes, e.g. "asocia cada fragmento de código con la salida que produce".

Correspondencia

Asocia cada operador de expresión regular en Figure 12.2 con lo que hace.

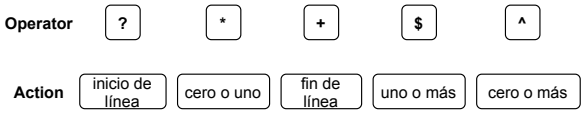

Figura 12.2: Matching items

Con la *correspondencia de muchos a muchos* las listas no tienen la misma longitud, por lo que algunos elementos pueden asociarse con varios y otros pueden no tener correspondencias.

Las correspondencias muchos-a-muchos son más difíciles porque los/las estudiantes no pueden hacer los apareamientos fáciles primero para reducir su espacio de búsqueda.

Los problemas de correspondencia pueden ser implementados haciendo que los/las alumnos/as envíen listas de ítems apareados (como "A3, B1, C2"), pero eso es burdo y propenso a errores.

Hacer que reconozcan un conjunto de pares correctos en un MCQ es aún peor, ya que es muy fácil de malinterpretar, lamentablemente .

Dibujar o arrastrar funciona mucho mejor, pero puede requerir más trabajo para implementarlo.

Ranking es un caso especial de emparejamiento que es (ligeramente) más fácil de responder mediante listas, ya que nuestras mentes son bastante buenas para detectar errores o anomalías en las secuencias. Los criterios de ranking determinan el nivel de razonamiento requerido.

Si haces que los alumnos y las alumnas ordenen los algoritmos de ranking del más rápido al más lento, probablemente estés ejercitando su memoria (es decir pidiéndoles que reconozcan los nombres de los algoritmos y sus propiedades), mientras que al pedirles que clasifiquen las soluciones desde las más robustas a las más frágiles, ejercitas su razonamiento y el juicio.

Resumir también requiere que los/as alumnos/as utilicen un pensamiento de orden superior y les da la oportunidad de practicar una habilidad que es muy útil para informar sobre errores. Por ejemplo, se puede les preguntar a los alumnos y a las alumnas, "¿Qué frase describe mejor cómo cambia la salida de f a medida que x varía de 0 a 10?" y luego se les dan varias opciones como una pregunta de opción múltiple.

También puedes pedir respuestas muy cortas de formato libre a preguntas en do-

minios restringidos, como por ejemplo, "¿Cuál es la característica clave de un algoritmo de ordenamiento que sea estable?" No podemos automatizar completamente la comprobación de las respuestas sin generar un número frustrante de falsos positivos (aceptando respuestas erróneas) y falsos negativos (rechazando las correctas), pero las preguntas de este tipo se prestan bien a la clasificación por pares (Section 5.3).

12.4. Calificación automática

Las herramientas de corrección automática han existido desde antes que yo naciera: la primera mención publicada data de 1960, y las encuestas publicadas por [Douc2005; Ihan2010] nombran muchas herramientas específicas.

Construir tales herramientas es mucho más complejo de lo que podría parecer a primera vista.

¿Cómo se representan las tareas?

¿Cómo se rastrean y reportan los envíos de tareas?

¿Pueden los/as alumnos/as trabajar cooperativamente?

¿Cómo se pueden enviar las tareas de manera segura?

[Edwa2014a] es un artículo entero dedicado a un esquema adaptativo para detectar y gestionar bucles infinitos de los envíos de código, y esa es sólo una de las muchas cuestiones que surgen.

Cuando se habla de autocalificadores, es importante diferenciar la satisfacción de los/las estudiantes de los resultados del aprendizaje. Por ejemplo, [Magu2018] sustituyó los laboratorios informales de programación de un curso de CS de segundo año por una prueba semanal evaluada utilizando un autocalificador. A los/as alumnos/as no les gustó el sistema automatizado, pero la tasa general de fracasos del curso se redujo a la mitad y se triplicó el número de alumnos/as que obtuvieron calificaciones distinguidas. Por el contrario , [Rubi2014] también comenzó a utilizar un autocalificador diseñado para competencias, pero no vio una disminución significativa en la tasa de abandono de sus alumnos y alumnas; una vez más, los alumnos y las alumnas hicieron comentarios negativos sobre la herramienta, que los autores atribuyeron a la calidad de los mensajes de retroalimentación más que a una aversión a la autocalificación.

[Srid2016] adoptaron un enfoque diferente. Utilizaron **fuzz testing** (es decir, fuzzing o fuzz testing, casos de prueba generados al azar) para comprobar si el código del alumno/a hace lo mismo que una implementación de referencia suministrada por el profesor. En el primer proyecto de un curso introductorio de 1400 alumnos y alumnas, el fuzz testing identificó errores que no fueron detectados por un conjunto de casos de prueba escritos a mano para más del 48

[Basu2015] dio a sus estudiantes una serie de soluciones a casos de prueba, pero tenían que desbloquear cada caso respondiendo a preguntas sobre su comportamiento esperado antes de poder aplicar la solución propuesta.

Por ejemplo, supongamos que los/as alumnos/as tuvieran que escribir una fun-

ción para encontrar el mayor par de números adyacentes en una lista.Antes de que se les permitiera usar las pruebas de la pregunta, tenían que elegir la respuesta correcta a, "¿Qué hace `largestPair(4, 3, -1, 5, 3, 3)`". En un curso universitario de 1300 personas, la gran mayoría de los estudiantes y las estudiantes eligió validar su comprensión sobre los casos de prueba de esta manera antes de intentar resolver los problemas, y luego hicieron menos preguntas y expresaron menos confusión sobre las tareas.

Contra las herramientas disponibles en el mercado

Es tentador usar herramientas de revisión de estilo fácilmente disponibles para calificar el código de los/las estudiantes. Sin embargo, [Nutb2016] inicialmente no encontró correlación entre las calificaciones proporcionadas por personas y las violaciones a las reglas del corrector de estilo. A veces esto se debía a que los alumnos y las alumnas violaban una regla muchas veces (perdiendo así más puntos de los que debían), pero otras veces se debía a que enviaban el código de inicio de tarea con pocas alteraciones y obtenían más puntos de los que debían.

Incluso herramientas construidas específicamente para la enseñanza pueden no estar a la altura de las necesidades de los profesores. [Keun2016a; Keun2016b] observaron los mensajes producidos por 69 herramientas de autocalificación. Encontraron que a menudo las herramientas de autocalificación no dan retroalimentación sobre cómo arreglar los problemas y dar el siguiente paso. También encontraron que la mayoría de los docentes y las docentes no pueden adaptar fácilmente la mayoría de las herramientas a sus necesidades: como muchas herramientas de flujo de trabajo, tienden a hacer cumplir supuestos no reconocidos por sus creadores, sobre el funcionamiento de sus instituciones. Su esquema de clasificación es una lista de compras útil cuando se examinan herramientas de este tipo.

[Buff2015] presenta una reflexión bien documentada sobre la idea de proporcionar una retroalimentación automatizada. Su punto de partida es que, "Los sistemas de calificación automatizados ayudan a los/las estudiantes a identificar los errores en su código, [pero] pueden inadvertidamente desalentarlos de pensar en forma crítica y a realizar pruebas exhaustivas, fomentando en cambio la dependencia de las pruebas del profesor". Una de las cuestiones clave que identificaron es que un estudiante o una estudiante pueden probar exhaustivamente su código, pero puede no ser implementado de acuerdo con las especificaciones del profesor. En este caso, el "fallo" no se debe a la falta de pruebas sino a un malentendido de los requisitos, y es poco probable que más pruebas expongan el problema. Si el sistema de autocalificación no proporciona una retroalimentación que brinde una mejor comprensión y aplicación de los conocimientos, esta experiencia sólo frustrará al alumno/a. Para proporcionar esa retroalimentación, el sistema de [Buff2015] identifica qué métodos son ejecutados por las pruebas fallidas del código del alumno para que el sistema pueda asociar dichas fallas con características particulares de lo que envío el alumno/a. El sistema decide si se han "ganado" ayudas específicas viendo si el alumno ha testeado lo

suficiente la característica asociada, para evitar que se apoye en las ayudas en vez de realizar las pruebas. En [Srid2016] se describen otros enfoques para compartir la retroalimentación con los/las estudiantes cuando estaban testeando automáticamente su código. Lo primero es proporcionar la salida esperada para las pruebas—pero luego los alumnos generan salidas hard-code para esas entradas (porque todo lo que se puede hacer se hace).

Lo segundo es informar los resultados de la aprobación o rechazo del código de los alumnos, pero suministrando las entradas y salidas reales de las pruebas sólo después de la fecha de envío. Sin embargo, es frustrante decirles a los alumnos que se equivocaron sin decirles por qué. Una tercera opción es utilizar una técnica llamada "hashing" para generar un valor que depende de la salida pero sin revelarla.

Si el usuario produce exactamente el resultado correcto su hash desbloqueará la solución, pero es imposible hacer el camino inverso a partir del hash para averiguar cuál es resultado supuesto.

El hash requiere más trabajo y explicación para configurarlo, pero da un buen balance entre mostrar la respuesta de manera prematura y no revelarla cuando podría ser de ayuda.

12.5. Higher-Level Thinking

Muchos tipos de ejercicios de programación son difíciles de evaluar por profesores en una clase con más de un puñado de alumnos y alumnas, e igualmente difícil para evaluar mediante plataformas automatizadas. Las clases van desarrollándose con vistas a proyectos de programación mayores (si todo va bien) , pero la única manera de dar retroalimentación es caso por caso. La *Revisión de código* es, en general, también difícil de calificar automáticamente, pero puede ser abordada si se les da a los/las estudiantes una lista de fallos y se les pide que cotejen determinados comentarios con determinadas líneas de código.

Por ejemplo, se le puede decir a la alumna que hay dos errores de indentación y un nombre de variable malo y pedirle que los señale.

Si son alumnos/as más avanzados, se les podría dar una media docena de tipos de comentarios que podrían hacerse sobre el código sin que se les diga cuánto de cada uno debería encontrar.

[Steg2016b] es un buen punto de partida para una rúbrica de estilo de código, mientras que [Luxt2009] muestra la revisión por pares en las clases de programación de manera más general.

Si se pide a los/las estudiantes que hagan revisiones, hay que usar la revisión por pares calibrada (Section 5.3) para que tengan modelos de cómo debería ser una buena retroalimentación.

Revisión de código

Marca los problemas en cada línea de código usando la rúbrica proporcionada.

```
01)  def addem(f):
02)      x1 = open(f).readlines()
03)      x2 = [x for x in x1 if x.strip()]
04)      changes = 0
05)      for v in x2:
06)          print('total', total)
07)          tot = tot + int(v)
08)      print('total')
```

1. nombre de la variable malo *2. uso de variable indefinida*
3. falta el valor de salida *4. variable no usada*

12.6. Ejercicios

Programar y ejecutar (en pares/10')

Crea un ejercicio corto de C&R, luego intercambia con un compañero y mira cuánto tiempo les toma a cada uno de ustedes entender y hacer el ejercicio del otro

¿Hubo alguna ambigüedad o malentendido en la descripción del ejercicio?

Invertir el código y la ejecución (grupos pequeños/15')

Forma grupos de 4 a 6 personas. Haz que cada miembro del grupo cree un ejercicio de C&R invertido que requiera que las personas averigüen qué entrada produce una salida en particular. Elige dos al azar y mira cuántas entradas diferentes que satisfagan los requisitos puede encontrar el grupo.

Siguiendo valores (en pares/10')

Escribe un programa corto (10-15 líneas), intercambia con una compañera y traza cómo las variables del programa cambian de valor con el tiempo. ¿Qué diferencias hay en la forma en que tú y tu compañero escribieron sus trazas?

Refactoring (grupos pequeños/15')

Forma grupos de 3-4 personas.

Haz que cada persona seleccione un trozo corto de código que haya escrito (de 10 a 30 líneas de largo) y que no sea tan ordenado como podría ser, luego elige una alumna al azar y haz que todos en el grupo lo ordenen independientemente.

¿En qué se diferencian las versiones limpias? ¿Qué tan bien o mal podrían acomodarse todas estas variaciones si se calificara automáticamente, o en una clase numerosa?

Rotulando un Diagrama (de a pares/10')

Dibuja un diagrama mostrando algo que se ha explicado recientemente: cómo los navegadores obtienen datos de los servidores, la relación entre los objetos y las clases, o cómo se indexan los dataframes en R.

Pon las etiquetas en el lateral y pídele a tu compañera que las coloque.

Acertijos de Lápiz y Papel (toda la clase/15')

[Butl2017] describe un conjunto de rompecabezas de lápiz y papel que pueden convertirse en asignaciones de programación introductorias, señalando que estas tareas son disfrutadas por los/as alumnos/as y que fomentan la meta-cognición. Piensa en un simple acertijo o juego de lápiz y papel al que jugabas de niño y describe cómo lo convertirías en un ejercicio de programación.

Contando Fallos (de a pares/15')

Cualquier estimación útil de cuánto tiempo se necesita para resolver un ejercicio debe considerar cuán frecuentes son los fallos y cuánto tiempo se pierde con ellos. Por ejemplo, editar archivos de texto parece una tarea sencilla, pero ¿qué hay de localizar esos archivos? La mayoría de los editores que usan interfase gráfica (GUI) guardan las cosas en el escritorio de la usuaria o en su directorio personal; si los archivos utilizados en un curso se almacenan en otro lugar, una proporción importante de los/las estudiantes no podrá navegar al directorio correcto sin ayuda. (Si esto te parece un problema menor, por favor vuelve a la discusión del punto ciego de las personas expertas en Chapter 3.)

Trabajando con un compañero, haz una lista de las cosas "simples" que has notado que salen mal en los ejercicios que has usado o tomado.

¿Con qué frecuencia aparecen?

¿Cuánto tiempo tardan los alumnos en arreglarse solos o con ayuda?

¿Cuánto tiempo de clase asignas actualmente para tratarlos?

Hablando de Tiempo (individual/10')

¿Qué tan precisas han sido las estimaciones de tiempo de los ejercicios de este libro hasta ahora?

13

Construyendo una comunidad de práctica

No tienes que arreglar todos los males de la sociedad para enseñar programación, pero *si debes* involucrarte en lo que sucede fuera de tu clase si quieres que las personas aprendan. Esto se aplica tanto a las personas que enseñan como a las que aprenden: muchos docentes free-range comienzan como voluntarios o como trabajadores a medio tiempo y deben balancear sus clases con muchos otros compromisos. Lo que sucede fuera del aula es tan importante para su éxito como lo es para el de sus estudiantes, así que la mejor manera de ayudar a ambos es fomentar una comunidad de enseñanza.

Finlandia y por qué no

Las escuelas de Finlandia se encuentran entre las más exitosas del mundo, pero así como lo señaló Anu Partanen,[1] *estas no han triunfado en aislamiento. Los intentos de otros países por adoptar métodos de enseñanza finlandeses están condenados al fracaso a menos que esos países también garanticen que la niñez (además de sus madres y padres) tengan seguridad, alimentación saludable, y reciban un trato adecuado en la administración de justicia t [Sahl2015; Wilk2011]. Esto no es ninguna sorpresa considerando lo que sabemos sobre la importancia de la motivación para el aprendizaje (Chapter 10): ninguna persona intentará hacerlo mejor si creen que el sistema es impredecible, injusto o indiferente.*

Un marco para pensar en enseñar a comunidades de práctica es el **aprendizaje situado**, el cual se centra en cómo la **participación periférica legítima** motiva a las personas a convertirse en miembros de una **comunidad de práctica** [Weng2015]. Desglosando estos términos, una comunidad de práctica se refiere a un grupo de personas unidas por su interés en alguna actividad, como tejer o la física de partículas. Una participación periférica legítima implica realizar aquellas tareas simples y de bajo riesgo que la comunidad reconoce como contribuciones válidas: tejer tu primera bufanda, llenar sobres durante una campaña electoral, o revisar documentación de software de código abierto.

El aprendizaje situado se centra en la transición de una persona recién llegada hasta su aceptación como colega por quienes ya son parte de la comunidad. Esto generalmente significa comenzar con tareas y herramientas simples, para después realizar tareas similares con herramientas más complejas y finalmente abordar el mismo

[1] https://www.theatlantic.com/national/archive/2011/12/what-americans-keep-ignoring-about-finlands-school-success/250564/

trabajo que practicantes avanzadas/os. Por ejemplo, las personas que aprenden música comienzan tocando canciones infantiles con flauta o ukelele, para después tocar otras canciones simples en trompeta o saxofón junto a una banda, y finalmente comienzan a explorar sus propios gustos musicales. Formas comunes de apoyar a esta progresión incluyen:

Resolución de problemas: "No puedo avanzar — ¿podemos trabajar en el diseño de esta lección conjuntamente?"

Solicita información: "¿Cuál es la contraseña del administrador de la lista de correo?"

Búsqueda de experiencia: "¿Alguien ha tenido una alumna o un alumno con discapacidad para leer?"

Compartir recursos: "El año pasado armé un sitio web para una clase que puedes usar como punto de partida".

Coordinación: "¿Podemos combinar nuestros pedidos de camisetas para obtener un descuento?"

Construir un argumento: "Será más fácil convencer a mi jefe para que haga cambios si sé cómo otros campamentos hacen esto".

Documentar proyectos: "Ya hemos tenido este problema cinco veces. Vamos a escribirlo de una vez por todas".

Mapeo de conocimientos: "¿Qué otros grupos están haciendo cosas como esta en vecindarios o ciudades cercanas?"

Visitas: "¿Podemos visitar su programa extracurricular? Necesitamos establecer uno en nuestra ciudad".

En términos generales[2], una comunidad de práctica puede ser:

Comunidad de acción: personas enfocadas en un objetivo compartido, cómo conseguir que un candidato sea elegido.

Comunidad de cuidado: en la que las personas integrantes de la comunidad se unen alrededor de un problema compartido, cómo tratar una enfermedad a largo plazo.

Comunidad de interés: enfocada en el amor compartido por algo como el backgammon o tejer.

Comunidad de lugar: personas que viven o trabajan juntas.

La mayoría de las comunidades son mezclas de estos diferentes tipos, como son las personas en Toronto a las que les gusta enseñar tecnología. El enfoque de una comunidad también puede cambiar con el tiempo: por ejemplo, un grupo de apoyo para personas que padecen depresión (comunidad de cuidado) puede decidir recaudar fondos para mantener una línea de ayuda (comunidad de acción). Mantener el funcionamiento de la línea de ayuda puede convertirse en el enfoque del grupo (comunidad de interés).

[2] https://www.feverbee.com/types-of-community-and-activity-within-the-community/

Sopa, luego himnos

Los manifiestos son divertidos de escribir, pero la mayoría de las personas se une a una comunidad para ayudar y recibir ayuda, no para discutir cómo redactar una declaración de visión[3]*. Por lo tanto, debes centrarte en qué personas pueden crear lo que otras/os integrantes de la comunidad usarán de inmediato. Una vez que tu organización demuestre que puede lograr cosas pequeñas, la gente estará más segura de que vale la pena ayudarte con proyectos más grandes. Ese es el momento de preocuparse por definir los valores que guiarán a tus miembros.*

13.1. Aprende, luego hazlo

El primer paso para construir una comunidad es decidir si deberías construirla, o si sería más efectivo unirte a una organización existente. Miles de grupos ya están enseñando habilidades tecnológicas, desde el 4-H Club[4] y programas de alfabetización[5] hasta organizaciones de programación sin fines de lucro como Black Girls Code[6] y Bridge[7]. Unirse a un grupo existente te dará ventaja en la enseñanza, un conjunto inmediato de colegas, y la oportunidad de aprender más sobre cómo manejar las cosas; Con suerte, ir aprendiendo esas habilidades mientras haces una contribución inmediata será más importante que poder decir que eres la persona que fundó algo nuevo.

Ya sea que te unas a un grupo existente o inicies uno propio, serás más efectivo si investigas un poco sobre organización de comunidades. [Alin1989; Lake2018] son probablemente los trabajos más conocidos sobre organización de grupos de base, mientras que [Brow2007; Midw2010; Lake2018] son manuales prácticos basados en décadas de experiencia. Si quieres leer más profundamente, [Adam1975] es la historia de la Highlander Folk School, cuyo enfoque ha sido emulado por muchos grupos exitosos, mientras que [Spal2014] es una guía para enseñar a adultos escrita por alguien con profundas raíces personales en la organización de grupos de base y NonprofitReady.org[8] ofrece capacitación profesional gratuita.

[3] Las personas que prefieren lo último a menudo están *solo* interesadas en discutir...
[4] http://www.4-h-canada.ca/
[5] https://www.frontiercollege.ca/
[6] http://www.blackgirlscode.com/
[7] http://bridgeschool.io/
[8] https://www.nonprofitready.org/

13.2. Cuatro pasos

Todas las personas que se involucren con tu organización (incluyéndote) pasan por cuatro fases: reclutamiento, incorporación, retención y retiro. No necesitas preocuparte por este ciclo cuando estés comenzando, pero vale la pena pensar en este tema tan pronto como más de un puñado de personas no fundadoras estén involucradas.

El primer paso es reclutar voluntarias y voluntarios. Tu estrategia de marketing debería ayudarte con esto haciendo que tu organización sea localizable y logrando que su misión y valor sean claros para las personas que quieran involucrarse

(Chapter 14). Comparte historias que ejemplifiquen el tipo de ayuda que buscas así como historias sobre las personas a las que estás ayudando, y deja en claro que hay muchas maneras de involucrarse. (Discutiremos esto con más detalle en la siguiente sección).

Tu mejor fuente de potenciales reclutas son tus propias clases: el método "verlo, practicarlo, enseñarlo" ha funcionado bien para organizaciones voluntarias desde que las organizaciones voluntarias *existen*. Asegúrate que cada clase u otro tipo de encuentro terminen informando a las personas cómo pueden ayudar y que su ayuda será bienvenida. De esta manera, las personas que se acerquen a ti sabrán lo que haces y habrán tenido la experiencia reciente de ser receptores de lo que ofreces, lo que ayuda a tu organización a evitar el punto ciego de la persona experta a nivel de la comunidad (Chapter 3).

Empieza pequeño

Ben Franklin[9] *observó que una persona que le ha hecho un favor a alguien es más propensa a hacer un nuevo favor a esa misma persona que la persona que ha recibido el favor. Por lo tanto, pedirle a la gente que haga algo pequeño por ti es un buen paso para lograr que hagan algo más grande. Una forma natural de hacer esto al enseñar es pedirle a la gente que corrijan la redacción o errores ortográficos en los materiales de tus lecciones, o que sugieran nuevos ejercicios o ejemplos. Escribir tus materiales de una manera mantenible(Section 6.3), les da la oportunidad de practicar algunas habilidades útiles y te da la oportunidad de comenzar una conversación que podría conducir a una nueva incorporación a tu organización.*

La mitad del ciclo de vida de la persona voluntaria es la incorporación y la retención, que cubriremos en las Secciones 13.3 y 13.4. El último paso es cuando alguien deja de ser parte de la organización: eventualmente, todas las personas siguen adelante, y las organizaciones saludables se preparan para ese momento. Algunas cosas simples pueden generar una sensación positiva ante el cambio, tanto para la persona que se va como para todas las que se quedan:

[9] https://en.wikipedia.org/wiki/Ben_Franklin_effect

Pide a las personas ser explícitas sobre su partida. para que todos sepan que realmente se han ido.

Asegúrate de que no se sientan con vergüenza de irse o sobre cualquier otra cosa.

Dales la oportunidad de transmitir sus conocimientos. Por ejemplo, puedes pedirles que asesoren a alguien durante algunas semanas como su última contribución, o que sean entrevistados por alguien que se queda en la organización para recopilar cualquier historia que valga la pena volver a contar.

Asegúrate que entreguen las llaves. Es incómodo descubrir, seis meses después de que alguien se fué, que esa era la única persona que sabía cómo reservar un lugar para el picnic anual.

Haz un seguimiento 2 a 3 meses después de que se vayan para ver si tienen más ideas sobre lo que funcionó y lo que no funcionó mientras estuvieron contigo, o algún consejo para ofrecer que tampoco pensaron dar o que se sentían incómodos de dar mientras se iban.

Agradéceles, tanto cuando se van como la próxima vez que tu grupo se reúna.

El manual que falta

Se han escrito miles de libros sobre cómo iniciar una empresa. Solo unos pocos describen cómo cerrar una o como dejarla sin problemas, a pesar de que existe un final para cada comienzo. Si alguna vez escribes uno, por favor házmelo saber.

13.3. Incorporación

Después de decidir formar parte de un grupo, la gente necesita ponerse al día, y [Shol2019] resume lo que sabemos al respecto. La primera regla es tener y hacer cumplir un código de conducta (Section 9.1), y encontrar una parte independiente que esté dispuesta a recibir y revisar informes de comportamiento inapropiado. Alguien fuera de la organización tendrá la objetividad que miembros de la organización pueden carecer, y puede proteger a quienes dudan reportar incidentes relacionados a las personas encargadas de proyectos por temor a represalias o daños a su reputación. El equipo que lidera el proyecto debe hacer públicas las decisiones donde se aplique el código de conducta para que la comunidad reconozca que el código es significativo.

La siguiente regla más importante es ser amigable. Como dijo Fogel [Foge2005], "Si un proyecto no hace una buena primera impresión, los recién llegados van a esperar mucho tiempo antes de darle una segunda oportunidad." Otros autores han confirmado empíricamente la importancia de entornos sociales, amables y educados en proyectos abiertos [Sing2012; Stei2013; Stei2018]:

Publica un mensaje de bienvenida en las páginas de redes sociales, canales de Slack, foros o listas de correo electrónico del proyecto. Los proyectos podrían considerar mantener un canal o lista de "Bienvenida", donde alguna de las personas que lidera el proyecto o gestiona la comunidad escribe una breve publicación pidiendo a los recién llegados que se presenten.

Ayuda a las personas a hacer una contribución inicial, por ejemplo, puedes etiquetar lecciones particulares o talleres que necesitan trabajo como "adecuados para las personas recién llegadas" y pedir a miembros ya establecidos que no las arreglen. De esta forma se aseguran lugares adecuados para que las personas recién llegadas comiencen a trabajar.

Dirige a personas recién llegadas hacia miembros similares a ellas para demostrarles que pertenecen.

Comparte los recursos esenciales del proyecto con las personas recién llegadas como las pautas de contribución.

Designa una o dos personas del proyecto como contacto para cada persona nueva. Esto puede hacer que quienes recién llegan sean menos reticentes a formular preguntas.

Una tercera regla que ayuda a todas las personas (no solo a quienes recién llegan) es hacer que el conocimiento esté actualizado y a disposición . Las personas nuevas son como exploradores que deben orientarse dentro de un paisaje desconocido [Dage2010]. La información dispersa hace que las nuevas personas se sientan perdidas y desorientadas. Considerando las diferentes opciones de lugares para mantener información, (por ejemplo, wikis, archivos en control de versiones, documentos compartidos, tweets antiguos o mensajes de Slack y archivos de correo electrónico) es importante mantener la información sobre un tema específico consolidada en un solo lugar para que las personas nuevas no naveguen por múltiples fuentes de datos para encontrar lo que necesitan. Organizar la información hace que las personas recién llegadas tengan más confianza y mejor orientación [Stei2016].

Finalmente, reconoce las primeras contribuciones de quienes recién inician y piensa dónde y cómo podrían ayudar a largo plazo. Una vez realizada su primera contribución, es probable que tengan una mejor idea de lo que pueden ofrecer y cómo el proyecto puede ayudarles. Ayuda a las personas nuevas a encontrar el siguiente problema en el que tal vez quieran trabajar o guiales hacía el siguiente tema que podrían disfrutar leyendo. En particular, animarles a ayudar a la próxima ola de nuevas personas es una buena manera de reconocer lo que han aprendido y una forma efectiva de transmitirlo.

13.4. Retención

> Si tu gente no la está pasando bien, algo está muy mal.
> — Saul Alinsky

Quienes participan de la comunidad no deberían esperar disfrutar cada momento de su trabajo con tu organización, pero si no disfrutan ninguno, no se quedarán. El disfrute no necesariamente significa tener una fiesta anual: la gente puede disfrutar cocinar, entrenar a otras personas, o simplemente trabajar en silencio junto a otros y otras. Hay varias cosas que toda organización debe hacer para garantizar que las personas tengan algo que valoran de su trabajo:

Pregunta a las personas qué quieren en vez de adivinar. Así como no eres tus estudiantes(Section 6.1), probablemente seas diferente de otras personas de tu organización. Pregúnta a las personas qué quieren hacer, que se sienten cómodas haciendo (que puede no ser lo mismo), y qué limitaciones de tiempo tienen. Pueden decir: "Cualquier cosa". pero incluso una breve conversación probablemente ayude a descubrir el hecho de que les gusta interactuar con las personas, pero preferirían no administrar las finanzas del grupo o viceversa.

Proporciona muchas formas de contribuir. Cuantas más formas haya para que las personas ayuden, más personas podrán hacerlo. Alguien a quien no le gusta estar frente a una audiencia puede mantener el sitio web de su organización, manejar sus cuentas o corregir las lecciones.

Reconoce las contribuciones. A todos y todas nos gusta que nos aprecien, así que las comunidades deben reconocer las contribuciones de sus miembros, tanto en público como en privado, mencionandoles en presentaciones, poniéndolas en el sitio web, etc. Cada hora que alguien le haya dado a tu proyecto puede ser una hora restada de su vida personal o de su empleo oficial; reconoce ese hecho y deja en claro que, si bien más horas serían bienvenidas, no esperas que hagan sacrificios insostenibles.

Haz espacio. Crees que estás siendo útil, pero intervenir en cada decisión priva a las personas de su autonomía, lo que reduce su motivación (Section 10). En particular, si siempre eres quien responde primero a correos electrónicos o mensajes de chat, las personas tienen menos oportunidades de crecer como miembros y crear colaboraciones horizontales. Como resultado, la comunidad continuará centrada en una o dos personas en lugar de convertirse en una red altamente conectada en la que otros y otras se sientan cómodos participando.

Otra forma de recompensar la participación es ofrecer capacitación. Las organizaciones necesitan presupuestos, propuestas de subvenciones y resolución de disputas. A la mayoría de las personas no se les enseña cómo hacer estas tareas más de lo que se les enseña a enseñar, así que la oportunidad de adquirir habilidades transferibles es una razón poderosa para que las personas se involucren y se mantengan

involucradas. Si vas a hacer esto, no intentes proporcionar la capacitación tú mismo a menos que sea en lo que te especializas. Muchos grupos cívicos y comunitarios tienen programas de este tipo y probablemente puedas llegar a un acuerdo con alguno de ellos.

Finalmente, si bien las personas voluntarias pueden hacer mucho, tareas como la administración del sistema y la contabilidad eventualmente necesitan personal remunerado. Cuando llegues a este punto, o no pagas nada o pagas un salario adecuado. Si no les paga nada, su verdadera recompensa es la satisfacción de hacer el bien; por otro lado, si les pagas una cantidad simbólica, le quitas esa satisfacción sin darles la posibilidad de ganarse la vida.

13.5. Gobernanza

Cada organización tiene una estructura de poder: la única pregunta es si es formal y se hace responsable o informal y, por lo tanto, no se hace responsable [Free1972]. Esta última forma, en realidad, funciona bastante bien para grupos de hasta media docena de personas en los que todos y todas se conocen. Más allá de esa cantidad, necesitas reglas para explicar quién tiene la autoridad para tomar qué decisiones y cómo lograr consenso (Section E.1).

El modelo de gobernanza que prefiero se denomina **Los comunes** (*commons*, en inglés). Los comunes es una manera de gestión conjunta realizada por una comunidad de acuerdo a las reglas que ella misma ha desarrollado y adoptado [Ostr2015]. Como subraya [Boll2014], las tres partes de esta definición son esenciales: Los comunes no es solo una propiedad compartida o un conjunto de bibliotecas de software, sino que también incluye a la comunidad que lo comparte y a las reglas que usan para hacerlo.

Los modelos más populares son los de corporaciones con fines de lucro y los de organizaciones sin fines de lucro; la mecánica varía de una jurisdicción a otra, por lo que debes buscar asesoramiento antes de elegir[10]. Ambos tipos de organización otorgan la máxima autoridad a su directorio. En términos generales, se trata de un **directorio de servicio** cuyos miembros también asumen otras funciones en la organización o un **directorio de gobernanza** cuya responsabilidad principal es contratar, supervisar y, si es necesario, despedir al director. Los miembros del directorio pueden ser elegidos por la comunidad o nombrados; en cualquier caso, es importante priorizar la capacidad sobre la pasión (la última es más importante para las bases) y tratar de reclutar habilidades particulares como contabilidad, marketing, etc.

Elige la democracia

Cuando llegue el momento, haz de tu organización una democracia: tarde o temprano (generalmente más temprano que tarde), cada directorio o

[10] Este es uno de los momentos en que vale la pena tener vínculos con el gobierno local u otras organizaciones afines.

junta designada se convierte en una sociedad de mutuo acuerdo. Darle poder a tus miembros es complicado, pero es la única forma inventada hasta ahora para garantizar que las organizaciones continúen satisfaciendo las necesidades reales de las personas.

13.6. Cuidarte y cuidar a tu comunidad

El Síndrome de desgaste ocupacional (emphburnout en inglés) es un riesgo crónico en cualquier actividad comunitaria [Pign2016], así que aprende a decir no más seguido de lo que dices sí. Si no te cuidas, no podrás cuidar a tu comunidad.

Quedándose sin "No"

Investigaciones en la década de 1990 parecían mostrar que nuestra capacidad de ejercer fuerza de voluntad es finita: si tenemos que resistirnos a comer la última dona en la caja cuando tenemos hambre, somos menos propensos a doblar la ropa y viceversa. Este fenómeno se llama ***agotamiento del ego****, y si bien estudios recientes no han podido replicar esos primeros resultados [Hagg2016], decir "sí" cuando estamos demasiado cansados para decir "no" es una trampa en la que caen muchos organizadores.*

Una forma de asegurarte de cumplir con tu "no" es escribir una lista de no-tareas, en la que anotes cosas que vale la pena hacer pero que *no* vas a hacer. Al momento de escribir este libro, mi lista incluye cuatro libros, dos proyectos de software, el rediseño de mi sitio web personal, y aprender a tocar la flauta irlandesa.

Finalmente, recuerda de vez en cuando que eventualmente toda organización necesita ideas frescas y nuevo liderazgo. Cuando llegue ese momento, entrena a tus sucesores y sigue adelante con la mayor gracia posible. Indudablemente harán cosas que tú no harías, pero pocas cosas en la vida son tan satisfactorias como ver algo que ayudaste a construir, cobrar vida propia. Celebra eso — no tendrás ningún problema para encontrar otra cosa que te mantenga ocupado/a.

13.7. Ejercicios

Varios de estos ejercicios se toman de [Brow2007].

¿Qué tipo de comunidad? (individual/15')

Vuelve a leer la descripción de los cuatro tipos de comunidades y decide con cuál o cuáles se identifican o a cuales aspiran..

Personas que puedes conocer (grupos pequeños/30')

Como organizador/a, a veces, parte de tu trabajo es ayudar a las personas a encontrar una manera de contribuir a pesar de sí mismas. En pequeños grupos, elige tres de las personas descriptas a continuación y discute cómo les ayudarías a convertirse en mejores colaboradores para tu organización.

Ana sabe más sobre cada tema que todas las demás personas juntas — al menos, ella cree que lo hace. No importa lo que digas, ella te corregirá; no importa lo que sepas, ella lo sabe mejor.

Catalina tiene tan poca confianza en su propia habilidad que no tomará ninguna decisión, sin importar que tan pequeña sea, hasta haber consultado con alguien más.

Fernando disfruta de saber cosas que otras personas no saben. Puede hacer milagros, pero cuando se le pregunta cómo lo hizo, sonreirá y dirá: "Oh, estoy seguro de que puedes resolverlo".

Andrea es tranquila. Nunca habla en las reuniones, incluso cuando sabe que otras personas están equivocadas. Podría contribuir a la lista de correo, pero es muy sensible a las críticas y siempre retrocede en lugar de defender su punto.

René aprovecha el hecho de que la mayoría de las personas preferirían hacer su parte del trabajo antes que quejarse de él. Lo frustrante es que él es muy convincente cuando alguien finalmente lo confronta. 'Todas las partes han cometido errores," dice él, o, "Bueno, creo que estás siendo un poco quisquilloso.".

Melisa tiene buenas intenciones pero, por alguna razón, siempre surge algo y sus tareas nunca se terminan hasta el último momento posible. Por supuesto, eso significa que todos los que dependen de ella no pueden hacer su trabajo hasta *después* del último momento posible ...

Roberto es grosero. "Esa es la forma en que hablo", dice. "Si no puedes solucionarlo, ve a buscar otro equipo". Su frase favorita es: "Eso es estúpido". y dice una obscenidad cada segunda oración.

Valores (grupos pequeños/45')

Responde estas preguntas por tu cuenta y luego compara tus respuestas con las de los demás.

1. ¿Cuáles son los valores que expresa tu organización?
2. ¿Son estos los valores que deseas que la organización exprese?
3. Si tu respuesta es no, ¿qué valores te gustaría expresar?
4. ¿Cuáles son los comportamientos específicos que demuestran esos valores?
5. ¿Qué comportamientos demostrarían lo contrario de esos valores?

Procedimientos para reuniones (grupos pequeños/30')

Responde estas preguntas por tu cuenta y luego compara tus respuestas con las de los demás.

1. ¿Cómo se manejan las reuniones?
2. ¿Es así cómo quieres que se realicen tus reuniones?
3. ¿Las reglas para conducir reuniones son explícitas o simplemente se asumen?
4. ¿Estas son las reglas que quieres tener?
5. ¿Quién tiene derecho a votar o tomar decisiones?
6. ¿Son estas las personas a quienes quisieras otorgar autoridad para tomar decisiones?
7. ¿Utilizan la regla de la mayoría, toman decisiones por consenso u otra cosa?
8. ¿Es así como quieres tomar decisiones?
9. ¿Cómo se enteran las personas en una reunión que se ha tomado una decisión?
10. ¿Cómo saben las personas que no estuvieron en una reunión qué decisiones se tomaron?
11. ¿Funciona esto para tu grupo?

Tamaño (grupos pequeños/20')

Responde estas preguntas por tu cuenta y luego compara tus respuestas con las de los demás.

1. ¿Qué tan grande es tu grupo?
2. ¿Es este el tamaño que deseas para su organización?
3. Si tu respuesta es no, ¿de qué tamaño te gustaría que fuera?
4. ¿Tienes algún límite en cuanto a la cantidad de miembros?
5. ¿Te beneficiarías de establecer ese límite?

Convertirse en miembro (grupos pequeños/45)

Responde estas preguntas por tu cuenta y luego compara tus respuestas con las de los demás.

1. ¿Cómo se une alguien a tu grupo?

2. ¿Qué tan bien funciona este proceso?
3. ¿Hay cuotas de membresía?
4. ¿Se requiere que las personas estén de acuerdo con alguna regla de comportamiento al unirse?
5. ¿Son estas las reglas de comportamiento que quieres?
6. ¿Cómo descubren las personas recién llegadas lo que hay que hacer?
7. ¿Qué tan bien funciona este proceso?

Contrataciones (grupos pequeños/30')

Responde estas preguntas por tu cuenta y luego compara tus respuestas con las de los demás.

1. ¿Tienes personal asalariado en tu organización o o son todos ad honorem?
2. ¿Deberías tener personal asalariado?
3. ¿Quieres / necesitas más o menos personal?
4. ¿Qué hace cada una de las personas del personal ?¿A que se dedican?
5. ¿Son estos los roles y funciones principales que necesitas que el personal desempeñe?
6. ¿Quién supervisa a tu personal?
7. ¿Es este el proceso de supervisión que quieres para tu grupo?
8. ¿Cuánto le pagan a tu personal?
9. ¿Es este el salario adecuado para realizar el trabajo necesario?

Dinero (grupos pequeños/30')

Responde estas preguntas por tu cuenta y luego compara tus respuestas con las de los demás.

1. ¿Quién paga qué?
2. ¿Esa es la persona que quisieras que pague?
3. ¿De dónde consiguen el dinero?
4. ¿Es así como quieres conseguirlo?
5. Si no, ¿tienes algún plan para conseguirlo de otra manera?

6. Si es así, ¿cuáles son esos planes?
7. ¿Quién da seguimiento a estos planes para asegurarse de que sucedan?
8. ¿Cuánto dinero tienen?
9. ¿Cuánto dinero necesitan?
10. ¿En qué gastan la mayor parte del dinero?
11. ¿Es así como quieres gastar el dinero?

Tomando ideas prestadas (toda la clase/15')

Muchas de mis ideas sobre cómo construir una comunidad han sido moldeadas por mi experiencia en el desarrollo de software de código abierto. [Foge2005] (que está disponible en línea[11]) es una buena guía de lo que ha funcionado y lo que no ha funcionado para esas comunidades, y el sitio de Guías de Código Abierto[12] también tiene una gran cantidad de información útil. Elige una sección de este último recurso, puede ser "Encontrando Usuarios para Tu Proyecto (*Finding Users for Your Project* en inglés)" o "Liderazgo y Gobernanza (*Leadership and Governance* en inglés)" y presenta al grupo, en dos minutos, una idea que encontraste útil o con la que estuviste muy en desacuerdo.

¿Quién eres tú? (grupos pequeños/20')

La Administración Nacional Oceánica y Atmosférica estadounidense (*NOAA* por sus siglas en inglés) tiene una guía breve, útil y divertida para lidiar con comportamientos disruptivos[13]. La guía clasifica esos comportamientos bajo etiquetas como "hablador/habladora", "indeciso/indecisa" y "tímido/tímida". y describe estrategias para manejar cada uno. En grupos de 3 a 6 personas, lean la guía y decidan cuál de las descripciones les queda mejor. ¿Crees que las estrategias descritas para manejar personas como tú son efectivas? ¿Son otras estrategias igualmente o más efectivas?

Creando lecciones entre todos y todas (grupos pequeños/30')

Una de las claves del éxito de Carpentries[14] es su énfasis en construir y mantener lecciones en forma colaborativa [Wils2016; Deve2018]. Trabajando en grupos de 3–4:

1. Elige una lección breve que todos y todas hayan usado.
2. Haz una revisión cuidadosa para crear una única lista con sugerencias de mejoras.
3. Ofrece esas sugerencias al autor de la lección.

[11] http://producingoss.com/
[12] https://opensource.guide/es
[13] https://coast.noaa.gov/ddb/
[14] http://carpentries.org

¿Estás crujiente? (individual/10')

Johnathan Nightingale escribió[15]:

> Cuando trabajaba en Mozilla, utilizábamos el término "crujiente" (crispy en inglés) para referirnos al estado justo antes de llegar al síndrome de desgaste ocupacional. Las personas que son crujientes no son divertidas. Son bruscas. Están ansiosas por una pelea que pueden ganar. Lloran sin mucha advertencia. ... reconocíamos lo "crujiente" en nuestros colegas y nos cuidábamos mutuamente [pero] es una cosa tan fea, que vimos tanto, que teníamos un proceso cultural completo a su alrededor.

Responde "sí" o "no" a cada una de las siguientes preguntas. ¿Qué tan cerca estás de tener síndrome de desgaste ocupacional?

- ¿Te has vuelto cínica/o o crítica/o en el trabajo?
- ¿Tienes que arrastrarte al trabajo o tienes problemas para comenzar a trabajar?
- ¿Te has vuelto irritable o impaciente con tus compañeros de trabajo?
- ¿Te resulta difícil concentrarte?
- ¿No logras satisfacción con tus logros?
- ¿Estás usando comida, drogas o alcohol para sentirte mejor o simplemente no sentir?

[15] https://mailchi.mp/d54702d0a790/take-my-horse-to-the-sand-hill-road

14

Difusión

Está de moda en los círculos tecnológicos menospreciar a las universidades y las instituciones gubernamentales como si fueran dinosaurios lentos, pero en mi experiencia no son peores que empresas de tamaño similar. El consejo de la escuela local, la biblioteca o la alcaldía o municipio pueden ofrecer espacio, financiación, publicidad, conexiones con otros grupos que puede que aún no hayas conocido y una serie de otras cosas útiles. Conocer estas posibilidades puede ayudar a resolver o evitar problemas a corto plazo y generar beneficios en el futuro.

14.1. Marketing

Las personas con antecedentes académicos y técnicos a menudo piensan que el **marketing** se trata sobre confundir y engañar. En realidad, trata sobre ver cosas desde la perspectiva de otras personas, comprendiendo sus deseos y necesidades, y explicando cómo puedes ayudarlas—en pocas palabras, cómo enseñarles. Este capítulo analizará cómo usar ideas de los capítulos anteriores para hacer que las personas entiendan y apoyen lo que estás haciendo.

El primer paso es averiguar qué le ofreces a cada persona, es decir, lo que realmente atrae a los/las voluntarios/as, a los fondos y a otro tipo de apoyo que puedas necesitar para continuar. La respuesta es contra-intuitiva. Por ejemplo, la mayoría de los/las científicos/as creen que los artículos científicos (conocidos como *papers*) que publican en revistas académicas son sus productos, pero en realidad los productos son los proyectos que presentan a convocatorias de subsidios: son estos proyectos los que atraen el dinero [Kuch2011]. Los artículos científicos son la publicidad que convence a otras personas a financiar esas propuestas, así como ahora los álbumes son los que convencen a la gente de comprar entradas de conciertos de músicos y camisetas.

Imagina que tu grupo ofrece talleres de programación de fin de semana a personas que están reinsertándose a la actividad laboral después de haber estado inactivas por varios años. Si los participantes del taller pueden pagar lo suficiente para cubrir tus costos, entonces son tus clientes y el taller es tu producto. Si por otro lado, los talleres son gratuitos o los/las estudiantes sólo están pagando un monto simbólico para reducir la tasa de ausencias, entonces tu producto real puede ser una mezcla de:

- tus propuestas de proyectos para solicitar subsidios;

- los/las personas que terminan exitosamente tus talleres a los que las empresas que te patrocinaron le gustaría contratar;
- el resumen de media página de tus talleres que la alcaldesa incluye en el balance o resumen anual presentado al concejo deliberante, que muestra cómo ella apoya el sector tecnológico local; o
- la satisfacción personal que obtienen los/las voluntarios/as cuando enseñan

Tal como con el diseño de lecciones (Chapter 6), los primeros pasos en marketing son crear la persona tipo, similar al estudiante tipo de la gente que podría estar interesada en lo que estás haciendo, y averiguar cuáles de sus necesidades puedes cumplir. Una manera de resumir lo último es escribir **discursos de presentación** dirigidos a diferentes personas tipo. Una plantilla ampliamente utilizada para este objetivo es:

Para	*objetivo de audiencia*
quién	*insatisfacción con lo que está disponible actualmente*
nuestros/as	*categoría*
proveen	*beneficio clave*.
A diferencia de	*alternativas*
nuestro programa	*característica distintiva clave.*

En el ejemplo del taller de fin de semana, podríamos usar este tono para los participantes
:

> Para *personas que regresan a la actividad laboral después de estar inactivas por varios años*
>
> quiénes *tienen todavía responsabilidades familiares* , nuestros/nuestras *talleres introductorios de programación* proveen *clases de fines de semana con guardería incluida*. A diferencia de las emphclases en línea, nuestro programa *le da a la gente la oportunidad de conocer a otras personas en la misma etapa de la vida*.

y este otro discurso de presentación para quienes toman decisiones en las empresas que podrían patrocinar los talleres:

> Para *empresas que quieren reclutar desarrolladores de software de nivel básico* quiénes *tienen dificultades para encontrar candidatos con suficiente madurez de diversas formaciones* nuestros *talleres introductorios de programación* proveen *potenciales candidatos/as*. A diferencia de *una feria de reclutamiento universitario*, nuestro programa *conecta empresas con una gran variedad de candidatos/as*.

Si no sabes por qué diferentes sponsors potenciales podrían estar interesados/as en lo que haces, pregúntales. Si lo sabes, pregúntales igual: las respuestas pueden cambian con el tiempo, y puedes descubrir cosas que no habías notado antes.

Estos discursos de presentación, una vez elaborados, deberían guiar lo que publicas en tu sitio web y en el material de difusión, para ayudar a la gente a descubrir lo más rápido posible si tú y ellos tienen algo de qué hablar. (*No deberías* copiar los discursos textualmente, aunque muchas personas en tecnología han visto esta plantilla tan a menudo que perderán total interés si vuelven a leerlo.

Mientras escribes estos discursos, recuerda que hay varias razones para aprender cómo programar

(Section 1.4). Una sensación de logro, control sobre sus propias vidas, y ser parte de una comunidad puede motivar a las personas más que el dinero (Chapter 10).

Podrían ofrecerse voluntarios/as a enseñar contigo porque sus amigos lo están haciendo. Igualmente, una empresa puede decir que está patrocinando clases para estudiantes de secundaria desfavorecidos económicamente porque quieren tener un grupo más grande de empleados potenciales en el futuro, aunque quizás el/la director/a ejecutivo/a simplemente podría estar haciéndolo porque es lo correcto.

14.2. Marcas y posicionamiento

Una **marca** es la primera reacción de una persona a la mención de un producto; Si la reacción es "¿Qué es eso?", tú (todavía) no tienes una marca. La marca es importante porque la gente no va a ayudar a algo que no conoce o no les importa.

La mayor parte de la discusión actual sobre las marcas se enfoca en cómo crear reconocimiento en línea. Las listas de correo, los blogs y Twitter te dan maneras de llegar a la gente, pero a medida que aumenta el volumen de desinformación, la gente presta menos atención a cada interrupción individual.

Esto hace que el **posicionamiento** sea cada vez más importante. A veces llamado "diferenciación", es lo que distingue tu oferta de las otras, la sección "a diferencia de" de tu discurso de presentación. Cuando te comunicas con personas que están familiarizadas con tu campo, debes enfatizar el posicionamiento, ya que es eso lo que llamará su atención.

Hay otras cosas que puedes hacer para ayudar a construir tu marca. Una de ellas es usar accesorios como un robot que una de tus estudiantes hizo a partir de restos que encontró en su casa [Schw2013] o el sitio web que otro estudiante hizo para el geriátrico sus padres.

Otra opción es hacer un video corto—no más de unos pocos minutos de duración— que resalte los antecedentes y logros de tus estudiantes. El objetivo de ambos ítems anteriores es contar una historia: mientras que la gente siempre pide datos, lo que creen y recuerdan son las historias.

Mitos fundacionales

Una de las historias más convincentes que una persona o grupo puede contar es por qué y cómo comenzaron. ¿Estás enseñando algo que desearías que alguien te hubiera enseñado pero no lo hizo? ¿Había una persona en particular a la que querías ayudar y eso abrió las compuertas?

Si no hay una sección en tu sitio web que comience con "Había una vez", piensa en agregar una.

Un paso crucial es lograr que tu organización sea encontrada en las búsquedas en línea. [DiSa2014b] descubrió que los términos de búsqueda que los padres y madres usaban para las clases de computación fuera de la escuela no encontraron esas clases, y muchos otros grupos se enfrentan a desafíos similares. Hay mucho mito sobre técnicas para hacer que las cosas puedan ser encontradas en internet (lo que a veces se refiere como**motor de optimización de posicionamiento en buscadores** o SEO por su sigla en inglés, *"search engine optimization"*). Dado el poder casi-monopólico y la falta de transparencia de Google, la mayor parte de estas estrategias se reducen a estar un paso por delante de los algoritmos diseñados para prevenir a las personas sobre rankings manipulados, sesgados o poco realistas.

A menos que tengas muy buena financiación, lo mejor que puedes hacer es hacer búsquedas regulares de tu organización y de ti en internet, y y ver qué encuentras. en-

tonces puedes leer estas guías[1] y hacer lo que puedas para mejorar tu sitio. Tener en menteesta viñeta de XKCD (en inglés)[2]: la gente no quiere saber sobre tu organigrama o obtener un recorrido virtual de su sitio– ellos quieren tu dirección, información sobre dónde hay estacionamiento cerca y alguna idea de lo que enseñas, cuándo lo enseñas, y cómo va a cambiar sus vidas.

No todo el mundo vive en línea

Estos ejemplos asumen que la gente tiene acceso a internet y que los grupos tienen dinero, materiales, tiempo libre y/o habilidades técnicas. La mayoría no tiene estos recursos—de hecho, aquellos que trabajan con grupos económicamente desfavorecidos muy probablemente no los tengan. (Como Rosario Robinson dice, "Gratis funciona para aquellos para que pueden permitirse lo gratuito.") Las historias son más importantes que el programa del curso en esas situaciones, porque son más fáciles de volver a contar. Igualmente, si las personas a las que esperas llegar no están en línea tan a menudo como tú, entonces publicar avisos en las carteleras de las escuelas, en bibliotecas locales, en centros de acogida, y las tiendas de comestibles pueden ser la forma más efectiva de llegar a ellos/ellas.

14.3. El arte de la llamada en frío

Crear un sitio web y esperar que las personas lo encuentren es fácil; llamar por teléfono o golpear en las puertas de sus casas sin ningún tipo de introducción previa es mucho más difícil. Al igual que con ponerse de pie y enseñar, sin embargo, es un oficio que puede aprenderse. Aquí hay diez reglas simples para convencer a las personas:

1: No lo hagas Si tienes que convencer a alguien de algo, lo más probable es que realmente no quieren hacerlo. Respeta eso: casi siempre es mejor a largo plazo dejar algo en particular sin hacer que usar la culpa o cualquier truco psicológico encubierto que sólo generará resentimiento.

2: Sé amable. No sé si realmente existe un libro llamado *Trucos secretos de los maestros de ventas ninja*, pero si existe, probablemente le dice a los lectores que hacer algo por un cliente potencial crea un sentido de obligación, lo que a su vez aumenta las probabilidades de una venta. Esto puede funcionar, pero solo funciona una vez y es algo dudoso de hacer. Por otro lado, si eres genuinamente amable y ayudas a otras personas porque eso es lo que las buenas personas hacen, sólo podrías inspirarlos a ser buenas personas también.

3: Apela al bien mayor. Si comienzas hablándole a las personas sobre lo que hay

[1] https://moz.com/learn/seo/on-page-factors
[2] https://xkcd.com/773/

disponible para ellas, les estás señalando que deberían pensar en su interacción contigo como si se tratara de un intercambio comercial de valor para ser negociado. En cambio, comienza explicando cómo su aporte y ayuda puede puede hacer del mundo un lugar mejor aquello para lo que quieres la ayuda de esas personas, y *dilo en serio*. Si lo que estás proponiendo no va a hacer del mundo un lugar mejor, propone algo mejor.

4: Comienza desde algo pequeño. La mayoría de las personas son comprensiblemente reacias a sumergirse de lleno en las cosas, así que debes darles la oportunidad de conocer el terreno y conocerte a ti y a las demás personas involucradas en lo que sea que necesites ayuda. No te sorprendas o decepciones si ahí es donde terminan las cosas: todo el mundo está ocupado o cansado o tiene proyectos propios, o tal vez simplemente tiene un modelo mental diferente de cómo se supone que funcionan las colaboraciones. Recuerda la regla 90-9-1– el 90

5: No construyas un proyecto: construye una comunidad. Solía pertenecer a un equipo de béisbol que nunca jugaba realmente al béisbol: nuestros "juegos" eran sólo una excusa para que estuviéramos juntos y disfrutemos de la compañía del otro. Probablemente no quieras llegar tan lejos, pero compartir una taza de té con alguien o celebrar el cumpleaños de su primer/a nieto/a pueden darte cosas que no podrías obtener con ninguna cantidad razonable de dinero.

6: Establece un punto de conexión. "Estaba hablando con X" o "Nos conocimos en Y" les da contexto, lo que a su vez hace sentir más cómodas a las personas. Esto debe ser específico: quienes envían correo basura y quienes hacen llamadas en frío nos han entrenado para ignorar cualquier cosa que comience con la frase "Hace poco tiempo encontré tu sitio web . . . "

7: Sé específico/a sobre lo que estás pidiendo. Las personas necesitan saber esto para que puedan determinar si el tiempo y las habilidades que tienen coinciden con lo que necesitas. Ser realista desde el principio también es una señal de respeto: si le dices a la gente que necesitas una mano para mover algunas cajas cuando en realidad estás mudando una casa entera, probablemente no te ayudarán por segunda vez.

8: Establece tu credibilidad. Menciona a quienes te patrocinan, tu tamaño, cuánto tiempo hace que existe tu grupo o algo que hayas logrado en el pasado para que las personas crean que vale la pena tomarte en serio.

9: Crea una ligera sensación de urgencia. "Esperamos lanzar esto en primavera" es mucho más probable que genere una respuesta positiva que "Eventualmente queremos lanzar esto." Sin embargo la palabra "ligera" es importante: si tu pedido es urgente, la mayoría de las personas asume que eres una persona desorganizada o que algo ha salido mal y luego puede pecar de prudencia.

10: Entiende la indirecta. Si la primera persona a la que le pides ayuda dice no, pregúntale a otra. Si la quinta o décima persona dice no, preguntarte si lo que estás tratando de hacer tiene sentido y vale la pena hacerlo.

La siguiente plantilla de correo electrónico sigue todas estas reglas. Ha funcionado bastante bien: descubrimos que alrededor de la mitad de los correos fueron respondidos, en aproximadamente la mitad de estas respuestas querían hablar más, y la mitad de estos últimos condujeron a talleres, lo que significa que 10–15 % de las cuentas de correo a las que nos dirigimos resultaron en talleres. Esto puede ser bastante desmoralizadorsi no estás acostumbrado/a, pero es mucho mejor que la tasa de respuesta de 2–3 % que la mayoría de las organizaciones esperan con llamadas en frío.

> Hola NOMBRE
>
> Espero que no te resulte inoportuno recibir este correo, pero quería continuar con nuestra conversación en LUGAR DE REUNIÓN para ver si tendrías interés en que realicemos un taller de entrenamiento para docentes—estamos programando la próxima tanda para las próximas dos semanas.
>
> Este taller de un día les enseñará a tus voluntarios/as una serie de prácticas de enseñanza útiles y basadas en evidencias. El taller se ha impartido más de cien veces de diversas maneras en seis continentes para organizaciones sin fines de lucro, bibliotecas y empresas, y todo el material está disponible gratuitamente en línea en http://teachtogether.tech.
>
> El temario incluye:
>
> - estudiantes tipo
> - diferencias entre diferentes tipos de estudiantes
> - uso de evaluaciones formativas para diagnosticar malentendidos
> - la enseñanza como un arte escénica
> - que motiva y desmotiva a estudiantes adultos/as
> - la importancia de la inclusividad y como ser un buen aliado/a
>
> Si esto te resulta interesante, por favor avísame—Agradecería la oportunidad de hablar de los modos y medios para hacerlo.
>
> Gracias,
>
> NOMBRE

Referencias

Construir alianzas con otros grupos que hacen cosas relacionadas a tu trabajo vale la pena de muchas maneras. Una de ellas son las referencias: si la persona que se aproxima en busca de tu ayuda podría ser mejor atendida por alguna otra organización, tómate un momento para presentarlos. Si ya has hecho esto varias veces, agrega información a tu sitio web para ayudar a la próxima persona a encontrar lo que necesita. En retribución, las organizaciones a las que estás ayudando pronto empezarán a ayudarte.

14.4. Cambio académico

Todo el mundo tiene miedo a lo desconocido y a pasar vergüenza.En consecuencia, la mayoría de la gente prefiere fracasar que cambiar. Por ejemplo, Lauren Herckis investigó por qué los/las profesores/as de las universidades no adoptan mejores metodos de enseñanza[3]. Lauren halló que el principal motivo es el miedo a parecer estúpido/a delante de los estudiantes; Las razones secundarias fueron preocupación de que los inevitables impactos al cambiar los métodos de enseñanza pudieran afectar las evaluaciones de los cursos (que a su vez podría afectar promociones y cargos) y el deseo de la gente de seguir emulando a los/las docentes que los inspiraron.

No tiene sentido discutir si estas cuestiones son "reales" o no: los/las profesores creen que son reales, por lo que cualquier plan para trabajar con el personal docente de las universidades necesita tenerlas en cuenta[4].

[Bark2015] realizaron un estudio de dos etapas sobre cómo quienes son docentes de ciencias de la computación adoptan nuevas prácticas de enseñanza, ya sea individualmente, como organización o en la sociedad en su conjunto. En este estudio se preguntaron y respondieron tres preguntas claves:

¿Cómo se enteran de nuevas prácticas de enseñanza los/las profesores/as? Buscan intencionalmente nuevas prácticas porque su motivación es resolver un problema (en particular, la participación de sus estudiantes), son conscientes de las nuevas prácticas a través de iniciativas deliberadas por parte de sus instituciones, las replican de colegas, o las obtienen por interacciones esperadas *e inesperadas* en conferencias (relacionadas a la enseñanza o de otro tipo).

¿Por qué las prueban? A veces, debido a los incentivos institucionales (por ejemplo, innovan para mejorar sus chances de promoción), pero a menudo hay tensión en las instituciones de investigación, donde la retórica sobre la importancia de la enseñanza tiene poca credibilidad. Otra razón importante es su propio análisis costo/beneficio: ¿les va a ahorrar tiempo esa innovación? Una tercera razón es que se inspiran en modelos de roles a seguir—otra vez, esto afecta en gran medida a las innovaciones destinadas a mejorar la motivación y participación más que los resultados del aprendizaje. Finalmente, cuarto factor son fuentes de confianza, por ejemplo, personas que han conocido en congresos o conferencias que se encuentran en la misma situación que ellos/as y han informado éxitos al adoptar las nuevas prácticas. Pero los/las profesores/as tienen preocupaciones que no siempre son abordadas por el grupo de personas que abogan por modificaciones. La primera era la ley de Glass: cualquier nueva herramienta o práctica inicialmente te relentiza, entonces mientras que las nuevas prácticas pueden hacer la enseñanza más efectiva a largo plazo, no pueden permitirse en el corto plazo. Otra preocu-

[3] https://www.insidehighered.com/news/2017/07/06/anthropologist-studies-why-professors-dont-adopt-innovative-teaching-methods

[4] Así como la prevalencia de la mentalidad fija entre los/las profesores cuando se trata de, la creencia de que algunas personas son "solo mejores profesores"

pación es que el diseño físico de las aulas dificulta muchas prácticas nuevas: por ejemplo, los grupos de discusión no funcionan bien en asientos de estilo teatro.

Pero el resultado más revelador fue el siguiente: "A pesar de ser los propios investigadores/as quienes enseñan, la mayor parte de los/las profesores/as de ciencias de computación con quienes hablamos no creía que los resultados de estudios sobre educación fueran razones creíbles para probar prácticas de enseñanza." Esto es consistente con otros hallazgos: incluso personas cuyas carreras están dedicadas a la investigación a menudo ignoran investigaciones sobre educación.

¿Por qué las siguen usando? Como dice [Bark2015], "Las devoluciones de los estudiantes son vitales," y son normalmente la razón más fuerte para continuar usando una práctica, a pesar de que sabemos que las auto-evaluaciones de los/las estudiantes no se correlacionan fuertemente con los resultados del aprendizaje [Star2014; Uttl2017] (aunque la asistencia a clases sí es un buen indicador del compromiso de los/las estudiantes). Otro motivo para retener una práctica de son los requisitos institucionales, aunque si esta es la única motivación, las personas abandonarán la práctica cuando el incentivo explícito o el monitoreo desaparezcan.

La buena noticia es que puedes abordar estos problemas sistemáticamente. [Baue2015] observó la adopción de nuevas técnicas médicas dentro de la Administración de Veteranos de los Estados Unidos. Hallaron que las prácticas médicas basadas en evidencia toman en promedio 17 años en ser incorporadas a prácticas generales de rutina, y que solo alrededor de la mitad de estas prácticas llegan a ser ampliamente adoptadas. Este hallazgo deprimente y otros, han estimulado el crecimiento de la **ciencia de la implementación**, que es el estudio de cómo lograr que la gente adopte mejores prácticas.

Como decía el Chapter 13, el punto de partida es descifrar qué es lo que creen que necesitan las personas que quieres ayudar. Por ejemplo, [Yada2016] resume las devoluciones de docentes de nivel primario y secundario sobre la preparación y el apoyo que desean. Aunque esto puede no ser aplicable a todos los entornos, tomar una taza de té con unas pocas personas y escucharlas antes de hablar hace un mundo de diferencia en su voluntad de intentar algo nuevo.

Una vez que sepas lo que la gente necesita, el siguiente paso es hacer cambios de manera incremental, dentro de los propios esquemas de las instituciones. [Nara2018] describe un programa intensivo de tres años basado en cohortes muy unidas y apoyo administrativo que triplicó las tasas de graduación, mientras que [Hu2017] describe el impacto de implementar un programa de certificaciones de seis meses para docentes de secundaria que quieren enseñar computación. El número de docentes de computación se ha mantenido estable entre 2007 y 2013, pero se cuadruplicó –sin disminuir la calidad– después de la introducción de un nuevo programa de certificación: los/las docentes que eran personas novatas en computación parecían ser tan eficaces en el curso introductorio como los/las docentes con más entrenamiento o formación informática en la enseñanza.

En un sentido más amplio, [Borr2014] categoriza maneras para lograr que ocurran cambios en educación superior. Las categorías están definidas de acuerdo a si el cambio es individual o sistémico y si se prescribe (de arriba hacia abajo) o es un

cambio emergente (de abajo hacia arriba) La persona que trata de hacer los cambios (y hacer que duren) tiene un rol distinto en cada una de estas situaciones, y consecuentemente debe seguir diferentes estrategias. El artículo continúa explicando en detalle cada uno de los métodos, mientras que [Hend2015a; Hend2015b] presentan las mismas ideas en una forma más procesable.

Al llegar a una institución, probablemente en principio caigas en la categoría cambio individual/emergente, dado que te aproximas a los/las docentes uno a uno y tratarás de lograr que los cambios ocurran de abajo hacia arriba. Si este es el caso, las estrategias que recomiendan Borrego y Henderson se enfocan en que los/las docentes reflexionen sobre sus prácticas de enseñanza de manera individual o en grupos. Haz programación en vivo para mostrarles lo que haces o los ejemplos que usas, luego haz que tus estudiantes programen en vivo en turnos para mostrar cómo usarían esas ideas y técnicas en su desempeño así les darás a todos/as la oportunidad de elegir cosas que les serán útiles en su contexto.

14.5. Docentes de rango libre

Las escuelas y las universidades no son los únicos lugares en donde la gente va a aprender programación; en los últimos años, un número creciente de personas ha acudido a talleres de rango libre y programas de entrenamiento intensivo. Estos últimos suelen tener entre uno y seis meses de duración, son llevados a cabo por grupos de voluntarios/as o por empresas con fines de lucro y su objetivo/apuntan a personas que se están re-entrenando para entrar en tecnología. Algunos programas son de muy alta calidad, pero otros existen primariamente para sacarle dinero del bolsillo a la gente [McMi2017].

[Thay2017] entrevistó a 26 graduados de estos entrenamientos intensivos que les dan una segunda oportunidad a aquellos que han perdido oportunidades previas de educación en computación (aunque expresarlo de este modo implica realizar grandes suposiciones en lo que respecta a personas de grupos subrepresentados). Las personas que participan de los entrenamientos intensivos enfrentan grandes costos y riesgos personales: deben gastar una cantidad significativa de tiempo, dinero y esfuerzo antes, durante y después de los entrenamientos intensivos, y cambiar de carrera puede tomar un año o más. Varias de las personas entrevistadas sienten que sus certificados fueron subestimados por sus empleadores; como dijeron algunas de ellas: obtener un trabajo significa aprobar una entrevista, pero dado que los/as entrevistadores/as muchas veces no comparten sus motivos de rechazo, es difícil saber qué arreglar o qué más aprender. Muchas personas han tenido que recurrir a pasantías (pagas o de otro tipo) y pasan mucho tiempo construyendo sus portfolios y haciendo networking. Las tres barreras informales que más fácilmente identificaron las personas entrevistadas fueron la jerga, el síndrome del impostor o impostora, y una sensación de no encajar.

[Burk2018] profundizó en este asunto al comparar las habilidades y credenciales que los/las reclutadores/as de la industria tecnológica buscan con aquellas habilida-

des provistas por carreras de cuatro años y programas de entrenamiento intensivo.. A partir de entrevistas con 15 gerentes de contratación de empresas de varios tamaños y algunos grupos focales, encontraron que los/las reclutadores/as enfatizaban uniformemente en habilidades blandas (especialmente trabajo en equipo, comunicación y la habilidad para continuar aprendiendo). Muchas compañías requerían un título de cuatro años (aunque no necesariamente en ciencias de la computación), pero muchas también elogiaron a egresados/as de programas de entrenamiento intensivo por ser mayores en edad o tener más madurez y por poseer unos conocimientos más actualizados.

Si te estás aproximando a un programa de entrenamiento intensivo que ya existe, tu mejor estrategia podría ser enfatizar lo que sabes sobre enseñanza, en lugar de lo que sabes sobre tecnología, ya que gran parte del personal y fundadores tiene experiencia en programación pero poca o ninguna capacitación en educación. Los primeros capítulos de este libro han servido con este público en el pasado, y [Lang2016] describe prácticas de enseñanza basadas en evidencia que pueden implementarse con un esfuerzo mínimo y a bajo costo. Estas prácticas tal vez no tengan el mayor impacto, pero lograr algunas victorias tempranas ayuda a generar apoyo para esfuerzos más grandes.

14.6. Reflexiones finales

Es imposible cambiar grandes instituciones por tu cuenta: necesitas aliados/as y para conseguir aliados/as, necesitas tácticas. La guía más útil que he encontrado es [Mann2015], que cataloga más de cuatro docenas de estas tácticas y las organiza de acuerdo a si se implementan mejor temprano, más adelante, a lo largo del ciclo de cambio, o cuando encuentras resistencia. Algunos de sus patrones incluyen:

En tu espacio: Mantén la nueva idea visible colocando recordatorios en toda la organización.

Símbolo: Para una nueva idea se mantenga viva en la memoria de una persona, entrega un objeto simbólico que pueda identificarse con el tema que se está presentando.

El rol del escepticismo: Identifica a los/las líderes de opinión fuertes que son escépticos de su nueva idea para desempeñar el papel de "persona escéptica oficial". Usa sus comentarios para mejorar tus esfuerzos, incluso si no logras hacer que cambien de opinión.

Compromiso futuro: Si puedes anticipar algunas de sus necesidades, puedes pedir un compromiso futuro a las personas más ocupadas. Si se les da un tiempo de entrega, pueden estar más dispuestos a ayudar.

La estrategia más importante es el deseo de cambiar tus metas basado en lo que

aprendes de las personas a las que estás tratando de ayudar. Por ejemplo, enseñarles tutoriales que muestran cómo usar una hoja de cálculo podría ser una ayuda más rápida y confiable que una introducción a JavaScript. A menudo he cometido el error de confundir cosas que me apasionan con cosas que las otras personas deberían saber; si realmente quieres ser quien las acompañe, recuerda siempre que el aprendizaje y el cambio tienen que ir en ambos sentidos.

La parte más difícil de construir relaciones es comenzarlas. Reserva una o dos horas cada mes para encontrar aliados/as y mantener tus relaciones con ellos/as. Una forma de hacer esto es pedirles consejo: ¿cómo creen que deberías hacer para que lo que están haciendo sea más conocido? ¿Dónde han encontrado espacio para dar clases? ¿Qué necesidades creen que no son cubiertas y tú serías capaz de lograr? Cualquier grupo que haya existido durante algunos años tendrá consejos útiles; también se sentirán halagados/as de que se les haya consultado, y sabrán quién eres la próxima vez que llames.

Y como dice [Kuch2011], si no puedes ser el primero en una categoria, tratar de crear una nueva categoría en la que sí puedas ser el primero. Si no puedes hacer eso, únete a un grupo existente o piensa en hacer algo completamente diferente. Esto no es derrotista: si alguien más ya está haciendo lo que tienes en mente, deberías incorporarte a ese grupo o abordar alguna de las otras cosas igualmente útiles que podrías estar haciendo en su lugar.

14.7. Ejercicios

Discurso de presentación para un miembro del concejo de la ciudad (individual/10')

Este capítulo describe una organización que ofrece talleres de programación de fin de semana para las personas que re-ingresan a la actividad laboral. Escribe un discurso de presentación para esa organización, dirigido a un miembro del consejo de la ciudad cuyo apoyo la organización necesita.

Presenta tu organización (individual/30')

Identifica dos grupos de personas de los que tu organización necesite apoyo y escribe un discurso de presentación dirigido a cada uno de estos grupos.

Asuntos de correo electrónico (parejas/10')

Escriba las líneas de asunto (y solo las líneas de asunto) para tres mensajes de correo electrónico: uno anunciando un nuevo curso, uno anunciando un nuevo patrocinio, y uno anunciando un cambio en el liderazgo del proyecto. Compare sus

líneas de asunto con las de tu pareja. Analicen si se pueden combinar las mejores características de cada asunto a la par que los acortan.

Manejando la resistencia pasiva (grupos pequeños/30')

Las personas que no quieren cambios a veces lo dirán en voz alta, pero también pueden utilizar a menudo varias formas de resistencia pasiva, como simplemente no lidiar con ello una y otra vez, o plantear un posible problema después de otro para hacer que el cambio parezca más arriesgado y más costoso de lo que en realidad es probable que sea [Scot1987]. En grupos pequeños, enumeren tres o cuatro razones por las que las personas podrían no querer que su iniciativa de enseñanza siga adelante, y expliquen qué pueden hacer con el tiempo y los recursos que tienen para contrarrestar cada una de esas razones.

¿Por qué aprender a programar? (individual/15')

Revisa el ejercicio "¿Por qué aprender a programar?" en Section 1.4. ¿Dónde se alinean tus razones para enseñar y las razones de tus estudiantes para aprender? ¿Dónde no se alinean? ¿Cómo afecta eso a tu marketing?

Programadores/as conversacionales (pensar en parejas y compartir/15')

Un/una **programador/a conversacional** es alguien que necesita saber lo suficiente sobre computación para tener una conversación valiosa con un/una programador/a, pero no se van a programar por su cuenta. [Wang2018] descubrió que la mayoría de los recursos de aprendizaje no abordan las necesidades de este grupo. En parejas, escriban un discurso para un taller de medio día destinado a ayudar a las personas que se ajustan a esta descripción. Luego, comparte el discurso de tu pareja con el resto de la clase.

Colaboraciones (grupos pequeños/30')

Responde por tu cuenta las siguientes preguntas, luego compara tus respuestas con las dadas por otros miembros de su grupo.

1. ¿Tienes algún acuerdo o relación con otros grupos?
2. ¿Quieres generar lazos con algún otro grupo?
3. El hecho de tener (o no tener) colaboraciones, ¿podría ayudarte a alcanzar tus metas?
4. ¿Cuáles son tus colaboraciones clave?
5. ¿Son estos los colaboradores adecuados o indicados para alcanzar tus metas u objetivos?

6. ¿Con qué grupos o entidades te gustaría que tu organización tuviera acuerdos o lazos?

Educacionalización (toda la clase/10')

[Laba2008] explora por qué los Estados Unidos y otros países siguen trasladando la solución a los problemas sociales hacia las instituciones educativas y por qué eso sigue sin funcionar. Tal como él remarca, "[La educación] ha hecho muy poco para promover la igualdad de raza, clase y género; para mejorar la salud pública, la productividad económica y la buena ciudadanía; o reducir el sexo sin cuidados en la adolescencia, las muertes por accidentes de tránsito, la obesidad y la destrucción ambiental. De hecho, en muchos sentidos la educación ha tenido un efecto negativo sobre estos problemas, sacando dinero y energía de las reformas sociales que podrían haber tenido un impacto más substancial." El autor continúa escribiendo:

> Entonces, ¿cómo debemos entender el éxito de esta institución, a la luz de su fracaso en lo que le pedimos? Una forma de pensar en esto es que la educación puede no estar haciendo lo que pedimos, pero está haciendo lo que queremos. Queremos una institución que persiga nuestros objetivos sociales de una manera que esté en línea con el individualismo en el corazón del ideal liberal, con el objetivo de resolver problemas sociales buscando cambiar los corazones, mentes y capacidades de cada estudiante individual. Otra forma de decir esto es que queremos una institución a través de la cual podamos expresar nuestros objetivos sociales sin violar el principio de elección individual que se encuentra en el centro de la estructura social, incluso si esto tiene el costo de no lograr estos objetivos. Entonces la educación puede servir como un punto de orgullo cívico, un lugar para exponer nuestros ideales, y un medio para participar en disputas edificantes pero que en última instancia intrascendentes sobre visiones alternativas de la buena vida. Al mismo tiempo, la educación también puede servir como un conveniente chivo expiatorio al que podemos culpar por su fracaso en lograr nuestras más altas aspiraciones para nosotros mismos como sociedad.

¿Cómo los esfuerzos para enseñar pensamiento computacional y la ciudadanía digital en las escuelas se adaptan a este marco? ¿Los programas de entrenamiento intensivo evitan estas trampas o simplemente las entregan con una nueva apariencia?

Adopción institucional (clase completa/15')

Relean la lista de motivaciones para adoptar nuevas prácticas dada en Section 14.4.

¿Cuáles de estas motivaciones se aplican a tí y a tus colegas? ¿Cuáles son irrelevantes en tu contexto? ¿En cuáles de estas motivaciones haces hincapié en los casos en que interactúas con personas que trabajan en instituciones educativas formales?

Si al principio no tienes éxito (grupos pequeños/15')

W.C. Fields probablemente nunca dijo "Si al principio no tienes éxito, inténtalo, inténtalo de nuevo. Luego abandona—no sirve de nada ser un maldito tonto al respecto." Sin embargo, sigue siendo un buen consejo: si las personas con las que intentas comunicarte no están respondiendo, podría ser que nunca los convenzas. En grupos de 3–4 personas, hagan una breve lista de señales que indiquen que debes dejar de intentar hacer algo en lo que crees. ¿Cuántas de estas señales ya son verdaderas?

Haz que falle (individual/15')

[Farm2006] presenta algunas reglas irónicas para lograr que nuevas herramientas *no sean* adoptadas, todas de las cuales aplican a nuevas prácticas de enseñanza:

1. Hazlo opcional.
2. Economiza en entrenamiento.
3. No las uses en un proyecto real.
4. Nunca la integres.
5. Úsala esporádicamente.
6. Hazla parte de una iniciativa de calidad
7. Marginaliza al campeón o campeona.
8. Capitaliza los primeros errores.
9. Haz una inversión pequeña.
10. Explota el miedo, la incertidumbre, la duda, la pereza y la inercia.

¿Cuál de estas reglas has visto implementarse recientemente? ¿Cuáles has utilizado tú mismo/a? ¿De qué forma se usaron esas reglas?

Mentoreo (todo la clase/15')

El Institute for African-American Mentoring in Computer Science[5] (Instituto de mentoría afroamericana en ciancias de la computación) ha publicado guías para mentorear estudiantes de doctorado[6]. Lee estas guías individualmente, luego discutan en la clase y califiquen los esfuerzos para tu propio grupo como: +1 (definitivamente lo haría), 0 (no estoy seguro o no es aplicable), o -1 (definitivamente no lo haría).

[5] http://www.iaamcs.org/
[6] http://iaamcs.org/guidelines

15

¿Por qué enseño?

Cuando comencé a trabajar como voluntario en la Universidad de Toronto, mis estudiantes me preguntaron por qué enseñaba gratis. Esta fue mi respuesta:

> Cuando tenía tu edad, pensaba que las universidades existían para enseñarle a la gente a aprender. Más tarde, en la escuela de posgrado, pensaba que las universidades se dedicaban a investigar y a crear nuevos conocimientos. Sin embargo, ahora que estoy en mis cuarenta años de edad, pienso que lo que realmente te estamos enseñando es cómo hacerte cargo del mundo, porque vas a tener que hacerlo quieras o no.
>
> Mis padres tienen setenta años. Ya no manejan el mundo; son las personas de mi edad quienes aprueban leyes y toman decisiones de vida o muerte en los hospitales. Y sin importar que tan aterrador sea , *nosotras/os* somos las personas adultas.
>
> En veinte años, nosotras/os estaremos camino hacia la jubilación y *tú* estarás a cargo. Eso puede parecer mucho tiempo cuando tienes diecinueve años, pero se pasa en un suspiro. Por eso te damos problemas cuyas respuestas no se pueden encontrar en las notas del año pasado. Por eso te ponemos en situaciones en las que tienes que decidir qué hacer ahora, qué se puede dejar para más tarde y qué puedes simplemente ignorar. Porque si no aprendes cómo hacer estas cosas ahora, no estarás lista/o para hacerlo cuando sea necesario.

Todo esto era verdad, pero no es toda la historia. No quiero que la gente haga del mundo un lugar mejor para que yo me pueda retirar cómodamente. Quiero que lo hagan porque es la aventura más grande de nuestro tiempo. Hace ciento cincuenta años, la mayoría de las sociedades practicaban la esclavitud. Hace cien años, en Canadá, mi abuela no era legalmente considerada una persona[1]. El año en que nací, la mayoría de las personas del mundo sufrían bajo algún régimen totalitario, y los jueces todavía dictaminaban terapia de electroshock para "curar" a los homosexuales. Todavía hay muchas cosas que están mal en el mundo, pero mira cuántas opciones más que nuestros abuelos y abuelas tenemos. Mira cuántas cosas más podemos saber, ser y disfrutar porque finalmente nos estamos tomando en serio la Regla de Oro.

Hoy soy menos optimista que entonces. Cambio climático, extinción masiva, capitalismo de vigilancia, desigualdad a una escala que no hemos visto hace un siglo, el resurgimiento del nacionalismo racista: mi generación vio cómo sucedió todo y

[1] https://en.wikipedia.org/wiki/The_Famous_Five_(Canada)

se quedó de brazos cruzados. La factura de nuestra cobardía, letargo y avaricia no se pagará hasta que mi hija crezca, pero *llegará*, y para cuando lo haga, no habrá soluciones fáciles para estos problemas (y posiblemente no hayan soluciones en absoluto).

Así que por eso enseño: Estoy enojado. Estoy enojado porque tu sexo, tu color y la riqueza y conexiones de tu madre y tu padre no deberían contar más que cuán inteligente, honesto/a o trabajador/a seas . Estoy enojado porque convertimos a Internet en una cloaca. Estoy enojado porque los nazis están en marcha una vez más y los multimillonarios juegan con cohetes espaciales mientras el planeta se derrite. Estoy enojado, entonces enseño, porque el mundo solo mejora cuando enseñamos a las personas cómo mejorarlo.

En su ensayo de 1947 "¿Por qué escribo?", George Orwell[2] escribió:

> En una época pacífica, podría haber escrito libros superficiales, decorativos o simplemente descriptivos, y podría haber permanecido casi inconsciente de mis lealtades políticas. Pero tal como están las cosas, me he visto obligado a convertirme en una especie de panfletista ... Cada línea de trabajo serio que he escrito desde 1936 ha sido escrita, directa o indirectamente, en contra del totalitarismo... Me parece una tontería, en un período como el nuestro, pensar que uno/a puede evitar escribir sobre tales temas. Todos escriben al respecto de una manera u otra. La cuestión es simplemente elegir de qué lado lo hacemos.

Reemplaza "escribir" por "enseñar" y tendrás la razón por la que hago lo que hago.

Gracias por leer — Espero que podamos enseñar juntos algún día. Hasta entonces:

Comienza donde estás.
Usa lo que tienes.
Ayuda a quien puedas.

[2] http://www.resort.com/~prime8/Orwell/whywrite.html

Bibliografía

[Abba2012] Janet Abbate. *Recoding Gender: Women's Changing Participation in Computing*. Describes the careers and accomplishments of the women who shaped the early history of computing, but have all too often been written out of that history. MIT Press, 2012. ISBN: 9780262534536.

[Abel2009] Andrew Abela. *Chart Suggestions - A Thought Starter*. http://extremepresentation.typepad.com/files/choosing-a-good-chart-09.pdf. A graphical decision tree for choosing the right type of chart. 2009.

[Adam1975] Frank Adams y Myles Horton. *Unearthing Seeds of Fire: The Idea of Highlander*. A history of the Highlander Folk School and its founder, Myles Horton. Blair, 1975. ISBN: 0895870193.

[Aike1975] Edwin G. Aiken, Gary S. Thomas y William A. Shennum. «Memory for a Lecture: Effects of Notes, Lecture Rate, and Informational Density». En: *Journal of Educational Psychology* 67.3 (1975). An early landmark study showing that taking notes improved retention., págs. 439-444. DOI: 10.1037/h0076613.

[Aiva2016] Efthimia Aivaloglou y Felienne Hermans. «How Kids Code and How We Know». En: *2016 International Computing Education Research Conference (ICER'16)*. Presents an analysis of 250,000 Scratch projects. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2960310.2960325.

[Dahl2018] Sarah Dahlby Albright, Titus H. Klinge y Samuel A. Rebelsky. «A Functional Approach to Data Science in CS1». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Describes the design of a CS1 class built around data science. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159550.

[Alin1989] Saul D. Alinsky. *Rules for Radicals: A Practical Primer for Realistic Radicals*. A widely-read guide to community organization written by one of the 20th Century's great organizers. Vintage, 1989. ISBN: 0679721134.

[Alqa2017] Basma S. Alqadi y Jonathan I. Maletic. «An Empirical Study of Debugging Patterns Among Novice Programmers». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports patterns in the debugging activities and success rates of novice programmers. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017761.

[Alvi1999] Jennifer Alvidrez y Rhona S. Weinstein. «Early Teacher Perceptions and Later Student Academic Achievement». En: *Journal of Educational Psychology* 91.4 (1999). An influential study of the effects of teachers' perceptions of students on their later achievements., págs. 731-746. DOI: 10.1037/0022-0663.91.4.731.

[Ambr2010] Susan A. Ambrose, Michael W. Bridges, Michele DiPietro, Marsha C. Lovett y Marie K. Norman. *How Learning Works: Seven Research-Based Principles for Smart Teaching*. Summarizes what we know about education and why we believe it's true, from cognitive psychology to social factors. Jossey-Bass, 2010. ISBN: 0470484101.

[Ande2001] Lorin W. Anderson y David R. Krathwohl, eds. *A Taxonomy for Learning, Teaching, And Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. A widely-used revision to Bloom's Taxonomy. Longman, 2001. ISBN: 080131903X.

[Armo2008] Michal Armoni y David Ginat. «Reversing: A Fundamental Idea in Computer Science». En: *Computer Science Education* 18.3 (sep. de 2008). Argues that the notion of reversing things is an unrecognized fundamental concept in computing education., págs. 213-230. DOI: 10.1080/08993400802332670.

[Atki2000] Robert K. Atkinson, Sharon J. Derry, Alexander Renkl y Donald Wortham. «Learning from Examples: Instructional Principles from the Worked Examples Research». En: *Review of Educational Research* 70.2 (jun. de 2000). A comprehensive survey of worked examples research at the time., págs. 181-214. DOI: 10.3102/00346543070002181.

[Auro2019] Valerie Aurora y Mary Gardiner. *How to Respond to Code of Conduct Reports*. Version 1.1. A short, practical guide to enforcing a Code of Conduct. Frame Shift Consulting LLC, 2019. ISBN: 978-1386922575.

[Avel2013] Emma-Louise Aveling, Peter McCulloch y Mary Dixon-Woods. «A Qualitative Study Comparing Experiences of the Surgical Safety Checklist in Hospitals in High-Income and Low-Income Countries». En: *BMJ Open* 3.8 (ago. de 2013). Reports the effectiveness of surgical checklist implementations in the UK and Africa. DOI: 10.1136/bmjopen-2013-003039.

[Bacc2013] Alberto Bacchelli y Christian Bird. «Expectations, Outcomes, and Challenges of Modern Code Review». En: *2013 International Conference on Software Engineering (ICSE'13)*. A summary of work on code review. Mayo de 2013.

[Bari2017] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill y Chris Parnin. «Do Developers Read Compiler Error Messages?» En: *2017 International Conference on Software Engineering (ICSE'17)*. Reports that developers do read error messages and doing so is as hard

as reading source code: it takes 13-25 % of total task time. Institute of Electrical y Electronics Engineers (IEEE), mayo de 2017. DOI: 10.1109/icse.2017.59.

[Bark2015] Lecia Barker, Christopher Lynnly Hovey y Jane Gruning. «What Influences CS Faculty to Adopt Teaching Practices?» En: *2015 Technical Symposium on Computer Science Education (SIGCSE'15)*. Describes how computer science educators adopt new teaching practices. Association for Computing Machinery (ACM), 2015. DOI: 10.1145/2676723.2677282.

[Bark2014] Lecia Barker, Christopher Lynnly Hovey y Leisa D. Thompson. «Results of a Large-Scale, Multi-Institutional Study of Undergraduate Retention in Computing». En: *2014 Frontiers in Education Conference (FIE'14)*. Reports that meaningful assignments, faculty interaction with students, student collaboration on assignments, and (for male students) pace and workload relative to expectations drive retention in computing classes, while interactions with teaching assistants or with peers in extracurricular activities have little impact. Institute of Electrical y Electronics Engineers (IEEE), oct. de 2014. DOI: 10.1109/fie.2014.7044267.

[Basi1987] Victor R. Basili y Richard W. Selby. «Comparing the Effectiveness of Software Testing Strategies». En: *IEEE Transactions on Software Engineering* SE-13.12 (dic. de 1987). An early and influential summary of the effectiveness of code review., págs. 1278-1296. DOI: 10.1109/tse.1987.232881.

[Basu2015] Soumya Basu, Albert Wu, Brian Hou y John DeNero. «Problems Before Solutions: Automated Problem Clarification at Scale». En: *2015 Conference on Learning @ Scale (L@S'15)*. Describes a system in which students have to unlock test cases for their code by answering MCQs, and presents data showing that this is effective. Association for Computing Machinery (ACM), 2015. DOI: 10.1145/2724660.2724679.

[Batt2018] Lina Battestilli, Apeksha Awasthi y Yingjun Cao. «Two-Stage Programming Projects: Individual Work Followed by Peer Collaboration». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Reports that learning outcomes were improved by two-stage projects in which students work individually, then re-work the same problem in pairs. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159486.

[Baue2015] Mark S. Bauer, Laura Damschroder, Hildi Hagedorn, Jeffrey Smith y Amy M. Kilbourne. «An Introduction to Implementation Science for the Non-Specialist». En: *BMC Psychology* 3.1 (sep. de 2015). Explains what implementation science is, using examples from the US Veterans Administration to illustrate. DOI: 10.1186/s40359-015-0089-9.

[Beck2013] Leland Beck y Alexander Chizhik. «Cooperative Learning Instructional methods for CS1: Design, Implementation, and Evaluation». En: *ACM Transactions on Computing Education* 13.3 (ago. de 2013). Reports that

cooperative learning enhances learning outcomes and self-efficacy in CS1., 10:1-10:21. ISSN: 1946-6226. DOI: 10.1145/2492686.

[Beck2014] Victoria Beck. «Testing a Model to Predict Online Cheating—Much Ado About Nothing». En: *Active Learning in Higher Education* 15.1 (ene. de 2014). Reports that cheating is no more likely in online courses than in face-to-face courses., págs. 65-75. DOI: 10.1177/1469787413514646.

[Beck2016] Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin y Catherine Mooney. «Effective Compiler Error Message Enhancement for Novice Programming Students». En: *Computer Science Education* 26.2-3 (jul. de 2016). Reports that improved error messages helped novices learn faster., págs. 148-175. DOI: 10.1080/08993408.2016.1225464.

[Beni2017] Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach y Dror G. Feitelson. «Meaningful Identifier Names: The Case of Single-Letter Variables». En: *2017 International Conference on Program Comprehension (ICPC'17)*. Reports that use of single-letter variable names doesn't affect ability to modify code, and that some single-letter variable names have implicit types and meanings. Institute of Electrical y Electronics Engineers (IEEE), mayo de 2017. DOI: 10.1109/icpc.2017.18.

[Benn2007a] Jens Bennedsen y Michael E. Caspersen. «Failure Rates in Introductory Programming». En: *ACM SIGCSE Bulletin* 39.2 (jun. de 2007). Reports that 67 % of students pass CS1, with variation from 5 % to 100 %., pág. 32. DOI: 10.1145/1272848.1272879.

[Benn2007b] Jens Bennedsen y Carsten Schulte. «What Does "Objects-first" Mean?: An International Study of Teachers' Perceptions of Objects-first». En: *2007 Koli Calling Conference on Computing Education Research (Koli'07)*. Teases out three meanings of "objects first" in computing education. 2007, págs. 21-29.

[Benn2000] Patricia Benner. *From Novice to Expert: Excellence and Power in Clinical Nursing Practice*. A classic study of clinical judgment and the development of expertise. Pearson, 2000. ISBN: 0130325228.

[Berg2012] Joseph Bergin, Jane Chandler, Jutta Eckstein, Helen Sharp, Mary Lynn Manns, Klaus Marquardt, Marianna Sipos, Markus Völter y Eugene Wallingford. *Pedagogical Patterns: Advice for Educators*. A catalog of design patterns for teaching. CreateSpace, 2012. ISBN: 9781479171828.

[Biel1995] Katerine Bielaczyc, Peter L. Pirolli y Ann L. Brown. «Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem Solving». En: *Cognition and Instruction* 13.2 (jun. de 1995). Reports that training learners in self-explanation accelerates their learning., págs. 221-252. DOI: 10.1207/s1532690xci1302_3.

[Bigg2011] John Biggs y Catherine Tang. *Teaching for Quality Learning at University*. A step-by-step guide to lesson development, delivery, and evaluation for people working in higher education. Open University Press, 2011. ISBN: 0335242758.

[Bink2012] Dave Binkley, Marcia Davis, Dawn Lawrie, Jonathan I. Maletic, Christopher Morrell y Bonita Sharif. «The Impact of Identifier Style on Effort and Comprehension». En: *Empirical Software Engineering* 18.2 (mayo de 2012). Reports that reading and understanding code is fundamentally different from reading prose, and that experienced developers are relatively unaffected by identifier style, but beginners benefit from the use of camel case (versus pothole case)., págs. 219-276. DOI: 10.1007/s10664-012-9201-4.

[Blik2014] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper y Daphne Koller. «Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming». En: *Journal of the Learning Sciences* 23.4 (oct. de 2014). Reports an attempt to categorize novice programmer behavior using machine learning that found interesting patterns on individual assignments., págs. 561-599. DOI: 10.1080/10508406.2014.954750.

[Bloo1984] Benjamin S. Bloom. «The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring». En: *Educational Researcher* 13.6 (jun. de 1984). Reports that students tutored one-to-one using mastery learning techniques perform two standard deviations better than those who learned through conventional lecture., págs. 4-16. DOI: 10.3102/0013189x013006004.

[Boha2011] Mark Bohay, Daniel P. Blakely, Andrea K. Tamplin y Gabriel A. Radvansky. «Note Taking, Review, Memory, and Comprehension». En: *American Journal of Psychology* 124.1 (2011). Reports that note-taking improves retention most at deeper levels of understanding., pág. 63. DOI: 10.5406/amerjpsyc.124.1.0063.

[Boll2014] David Bollier. *Think Like a Commoner: A Short Introduction to the Life of the Commons*. A short introduction to a widely-used model of governance. New Society Publishers, 2014. ISBN: 0865717680.

[Borr2014] Maura Borrego y Charles Henderson. «Increasing the Use of Evidence-Based Teaching in STEM Higher Education: A Comparison of Eight Change Strategies». En: *Journal of Engineering Education* 103.2 (abr. de 2014). Categorizes different approaches to effecting change in higher education., págs. 220-252. DOI: 10.1002/jee.20040.

[DuBo1986] Benedict Du Boulay. «Some Difficulties of Learning to Program». En: *Journal of Educational Computing Research* 2.1 (feb. de 1986). Introduces the idea of a notional machine., págs. 57-73. DOI: 10.2190/3lfx-9rrf-67t8-uvk9.

[Bria2015] Samuel A. Brian, Richard N. Thomas, James M. Hogan y Colin Fidge. «Planting Bugs: A System for Testing Students' Unit Tests». En: *2015 Conference on Innovation and Technology in Computer Science Education (ITiCSE'15)*. Describes a tool for assessing students' programs and unit tests and finds that students often write weak tests and misunderstand the role of unit testing. Association for Computing Machinery (ACM), 2015. DOI: 10.1145/2729094.2742631.

[Broo2016] Stephen D. Brookfield y Stephen Preskill. *The Discussion Book: 50 Great Ways to Get People Talking*. Describes fifty different ways to get groups talking productively. Jossey-Bass, 2016. ISBN: 9781119049715.

[Brop1983] Jere E. Brophy. «Research on the Self-Fulfilling Prophecy and Teacher Expectations». En: *Journal of Educational Psychology* 75.5 (1983). A early, influential study of the effects of teachers' perceptions on students' achievements., págs. 631-661. DOI: 10.1037/0022-0663.75.5.631.

[Brow2007] Michael Jacoby Brown. *Building Powerful Community Organizations: A Personal Guide to Creating Groups that Can Solve Problems and Change the World*. A practical guide to creating effective organizations in and for communities. Long Haul Press, 2007. ISBN: 0977151808.

[Brow2017] Neil C. C. Brown y Amjad Altadmri. «Novice Java Programming Mistakes». En: *ACM Transactions on Computing Education* 17.2 (mayo de 2017). Summarizes the authors' analysis of novice programming mistakes. DOI: 10.1145/2994154.

[Brow2018] Neil C. C. Brown y Greg Wilson. «Ten Quick Tips for Teaching Programming». En: *PLoS Computational Biology* 14.4 (abr. de 2018). A short summary of what we actually know about teaching programming and why we believe it's true. DOI: 10.1371/journal.pcbi.1006023.

[DeBr2015] Pedro De Bruyckere, Paul A. Kirschner y Casper D. Hulshof. *Urban Myths about Learning and Education*. Describes and debunks some widely-held myths about how people learn. Academic Press, 2015. ISBN: 9780128015377.

[Buff2015] Kevin Buffardi y Stephen H. Edwards. «Reconsidering Automated Feedback: A Test-Driven Approach». En: *2015 Technical Symposium on Computer Science Education (SIGCSE'15)*. Describes a system that associates failed tests with particular features in a learner's code so that learners cannot game the system. Association for Computing Machinery (ACM), 2015. DOI: 10.1145/2676723.2677313.

[Burg2015] Sheryl E. Burgstahler. *Universal Design in Higher Education: From Principles to Practice*. Second. Describes how to make online teaching materials accessible to everyone. Harvard Education Press, 2015. ISBN: 9781612508160.

[Burk2018] Quinn Burke, Cinamon Bailey, Louise Ann Lyon y Emily Greeen. «Understanding the Software Development Industry's Perspective on Coding Boot Camps Versus Traditional 4-Year Colleges». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Compares the skills and credentials that tech industry recruiters are looking for to those provided by 4-year degrees and bootcamps. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159485.

[Butl2017] Zack Butler, Ivona Bezakova y Kimberly Fluet. «Pencil Puzzles for Introductory Computer Science». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Describes pencil-and-paper puzzles that can be turned into CS1/CS2 assignments, and reports that they are enjoyed by students and encourage meta-cognition. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017765.

[Byck2005] Pauli Byckling, Petri Gerdt y Jorma Sajaniemi. «Roles of Variables in Object-Oriented Programming». En: *2005 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05)*. Presents single-variable design patterns common in novice programs. Association for Computing Machinery (ACM), 2005. DOI: 10.1145/1094855.1094972.

[Camp2016] Jennifer Campbell, Diane Horton y Michelle Craig. «Factors for Success in Online CS1». En: *2016 Conference on Innovation and Technology in Computer Science Education (ITiCSE'16)*. Compares students who opted in to an online CS1 class online with those who took it in person in a flipped classroom. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2899415.2899457.

[Carr2014] John Carroll. «Creating Minimalist Instruction». En: *International Journal of Designs for Learning* 5.2 (nov. de 2014). A look back on the author's work on minimalist instruction. DOI: 10.14434/ijdl.v5i2.12887.

[Carr1987] John Carroll, Penny Smith-Kerker, James Ford y Sandra Mazur-Rimetz. «The Minimal Manual». En: *Human-Computer Interaction* 3.2 (jun. de 1987). The foundational paper on minimalist instruction., págs. 123-153. DOI: 10.1207/s15327051hci0302_2.

[Cart2017] Adam Scott Carter y Christopher David Hundhausen. «Using Programming Process Data to Detect Differences in Students' Patterns of Programming». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Shows that students of different levels approach programming tasks differently, and that these differences can be detected automatically. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017785.

[Casp2007] Michael E. Caspersen y Jens Bennedsen. «Instructional Design of a Programming Course». En: *2007 International Computing Education Research Conference (ICER'07)*. Goes from a model of human cognition to three lear-

ning theories, and from there to the design of an introductory object-oriented programming course. Association for Computing Machinery (ACM), 2007. DOI: 10.1145/1288580.1288595.

[Cele2018] Mehmet Celepkolu y Kristy Elizabeth Boyer. «Thematic Analysis of Students' Reflections on Pair Programming in CS1». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Reports that pair programming has the same learning gains side-by-side programming but higher student satisfaction. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159516.

[Ceti2016] Ibrahim Cetin y Christine Andrews-Larson. «Learning Sorting Algorithms Through Visualization Construction». En: *Computer Science Education* 26.1 (ene. de 2016). Reports that people learn more from constructing algorithm visualizations than they do from viewing visualizations constructed by others., págs. 27-43. DOI: 10.1080/08993408.2016.1160664.

[Chen2018] Chen Chen, Paulina Haduong, Karen Brennan, Gerhard Sonnert y Philip Sadler. «The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages». En: *Computer Science Education* 29.1 (nov. de 2018). Finds that students whose first language was graphical had higher grades than students whose first language was textual when the languages were introduced in or before early adolescent years., págs. 23-48. DOI: 10.1080/08993408.2018.1547564. URL: https://doi.org/10.1080/08993408.2018.1547564.

[Chen2009] Nicholas Chen y Maurice Rabb. «A Pattern Language for Screencasting». En: *2009 Conference on Pattern Languages of Programs (PLoP'09)*. A brief, well-organized collection of tips for making screencasts. Association for Computing Machinery (ACM), 2009. DOI: 10.1145/1943226.1943234.

[Chen2017] Nick Cheng y Brian Harrington. «The Code Mangler: Evaluating Coding Ability Without Writing Any Code». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports that student performance on exercises in which they undo code mangling correlates strongly with performance on traditional assessments. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017704.

[Cher2007] Mauro Cherubini, Gina Venolia, Rob DeLine y Amy J. Ko. «Let's Go to the Whiteboard: How and Why Software Developers Use Drawings». En: *2007 Conference on Human Factors in Computing Systems (CHI'07)*. Reports that developers draw diagrams to aid discussion rather than to document designs. Association for Computing Machinery (ACM), 2007. DOI: 10.1145/1240624.1240714.

[Cher2009] Sapna Cheryan, Victoria C. Plaut, Paul G. Davies y Claude M. Steele. «Ambient Belonging: How Stereotypical Cues Impact Gender Participation in Computer Science». En: *Journal of Personality and Social Psycho-*

logy 97.6 (2009). Reports that subtle environmental clues have a measurable impact on the interest that people of different genders have in computing., págs. 1045-1060. DOI: 10.1037/a0016239.

[Chet2014] Raj Chetty, John N. Friedman y Jonah E. Rockoff. «Measuring the Impacts of Teachers II: Teacher Value-Added and Student Outcomes in Adulthood». En: *American Economic Review* 104.9 (sep. de 2014). Reports that good teachers have a small but measurable impact on student outcomes., págs. 2633-2679. DOI: 10.1257/aer.104.9.2633.

[Chi1989] Michelene T. H. Chi, Miriam Bassok, Matthew W. Lewis, Peter Reimann y Robert Glaser. «Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems». En: *Cognitive Science* 13.2 (abr. de 1989). A seminal paper on the power of self-explanation., págs. 145-182. DOI: 10.1207/s15516709cog1302_1.

[Coll1991] Allan Collins, John Seely Brown y Ann Holum. «Cognitive Apprenticeship: Making Thinking Visible». En: *American Educator* 6 (1991). Describes an educational model based on the notion of apprenticeship and master guidance., págs. 38-46.

[Coco2018] Center for Community Organizations. *The "Problem" Woman of Colour in the Workplace*. https://coco-net.org/problem-woman-colour-nonprofit-organizations/. Outlines the experience of many women of color in the workplace. 2018.

[Coom2012] Norman Coombs. *Making Online Teaching Accessible*. An accessible guide to making online lessons accessible. Jossey-Bass, 2012. ISBN: 9781458725288.

[Covi2017] Martin V. Covington, Linda M. von Hoene y Dominic J. Voge. *Life Beyond Grades: Designing College Courses to Promote Intrinsic Motivation*. Explores ways of balancing intrinsic and extrinsic motivation in institutional education. Cambridge University Press, 2017. ISBN: 9780521805230.

[Craw2010] Matthew B. Crawford. *Shop Class as Soulcraft: An Inquiry into the Value of Work*. A deep analysis of what we learn about ourselves by doing certain kinds of work. Penguin, 2010. ISBN: 9780143117469.

[Crou2001] Catherine H. Crouch y Eric Mazur. «Peer Instruction: Ten Years of Experience and Results». En: *American Journal of Physics* 69.9 (sep. de 2001). Reports results from the first ten years of peer instruction in undergraduate physics classes, and describes ways in which its implementation changed during that time., págs. 970-977. DOI: 10.1119/1.1374249.

[Csik2008] Mihaly Csikszentmihaly. *Flow: The Psychology of Optimal Experience*. An influential discussion of what it means to be fully immersed in a task. Harper, 2008. ISBN: 978-0061339202.

[Cunn2017] Kathryn Cunningham, Sarah Blanchard, Barbara J. Ericson y Mark Guzdial. «Using Tracing and Sketching to Solve Programming Problems». En: *2017 Conference on International Computing Education Research (ICER'17)*. Found that writing new values near variables' names as they change is the most effective tracing technique. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3105726.3106190.

[Cutt2017] Quintin Cutts, Charles Riedesel, Elizabeth Patitsas, Elizabeth Cole, Peter Donaldson, Bedour Alshaigy, Mirela Gutica, Arto Hellas, Edurne Larraza-Mendiluze y Robert McCartney. «Early Developmental Activities and Computing Proficiency». En: *2017 Conference on Innovation and Technology in Computer Science Education (ITiCSE'17)*. Surveyed adult computer users about childhood activities and found strong correlation between confidence and computer use based on reading on one's own and playing with construction toys with no moving parts (like Lego). Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3174781.3174789.

[Dage2010] Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard y Jacqueline P. de Vries. «Moving Into a New Software Project Landscape». En: *2010 International Conference on Software Engineering (ICSE'10)*. A look at how people move from one project or domain to another. ACM Press, 2010. DOI: 10.1145/1806799.1806842.

[Deb2018] Debzani Deb, Muztaba Fuad, James Etim y Clay Gloster. «MRS: Automated Assessment of Interactive Classroom Exercises». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Reports that doing in-class exercises with realtime feedback using mobile devices improved concept retention and student engagement while reducing failure rates. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159607.

[Denn2019] Paul Denny, Brett A. Becker, Michelle Craig, Greg Wilson y Piotr Banaszkiewicz. «Research This! Questions that Computing Educators Most Want Computing Education Researchers to Answer». En: *2019 Conference on International Computing Education Research (ICER'19)*. Found little overlap between the questions that computing education researchers are most interested in and the questions practitioners want answered. Association for Computing Machinery (ACM), 2019.

[Derb2006] Esther Derby y Diana Larsen. *Agile Retrospectives: Making Good Teams Great*. Describes how to run a good project retrospective. Pragmatic Bookshelf, 2006. ISBN: 0977616649.

[Deve2018] Gabriel A. Devenyi, Rémi Emonet, Rayna M. Harris, Kate L. Hertweck, Damien Irving, Ian Milligan y Greg Wilson. «Ten Simple Rules for Collaborative Lesson Development». En: *PLoS Computational Biology* 14.3 (mar. de 2018). Describes how to develop lessons together. DOI: 10.1371/journal.pcbi.1005963.

[Dida2016] David Didau y Nick Rose. *What Every Teacher Needs to Know About Psychology*. An informative, opinionated explanation of what modern psychology has to say about teaching. John Catt Educational, 2016. ISBN: 1909717851.

[DiSa2014a] Betsy DiSalvo, Mark Guzdial, Amy Bruckman y Tom McKlin. «Saving Face While Geeking Out: Video Game Testing as a Justification for Learning Computer Science». En: *Journal of the Learning Sciences* 23.3 (jul. de 2014). Found that 65 % of male African-American participants in a game testing program went on to study computing., págs. 272-315. DOI: 10.1080/10508406.2014.893434.

[DiSa2014b] Betsy DiSalvo, Cecili Reid y Parisa Khanipour Roshan. «They Can't Find Us». En: *2014 Technical Symposium on Computer Science Education (SIGCSE'14)*. Reports that the search terms parents were likely to use for out-of-school CS classes didn't actually find those classes. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2538862.2538933.

[Douc2005] Christopher Douce, David Livingstone y James Orwell. «Automatic Test-Based Assessment of Programming». En: *Journal on Educational Resources in Computing* 5.3 (sep. de 2005). Reviews the state of auto-graders at the time. DOI: 10.1145/1163405.1163409.

[Edwa2014b] Stephen H. Edwards y Zalia Shams. «Do Student Programmers All Tend to Write the Same Software Tests?» En: *2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE'14)*. Reports that students wrote tests for the happy path rather than to detect hidden bugs. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2591708.2591757.

[Edwa2014a] Stephen H. Edwards, Zalia Shams y Craig Estep. «Adaptively Identifying Non-Terminating Code when Testing Student Programs». En: *2014 Technical Symposium on Computer Science Education (SIGCSE'14)*. Describes an adaptive scheme for detecting non-terminating student coding submissions. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2538862.2538926.

[Endr2014] Stefan Endrikat, Stefan Hanenberg, Romain Robbes y Andreas Stefik. «How Do API Documentation and Static Typing Affect API Usability?» En: *2014 International Conference on Software Engineering (ICSE'14)*. Shows that types do add complexity to programs, but it pays off fairly quickly by acting as documentation hints for a method's use. ACM Press, 2014. DOI: 10.1145/2568225.2568299.

[Ensm2003] Nathan L. Ensmenger. «Letting the "Computer Boys" Take Over: Technology and the Politics of Organizational Transformation». En: *International Review of Social History* 48.S11 (dic. de 2003). Describes how programming was turned from a female into a male profession in the 1960s., págs. 153-180. DOI: 10.1017/s0020859003001305.

[Ensm2012] Nathan L. Ensmenger. *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*. Traces the emergence and rise of computer experts in the 20th Century, and particularly the way that computing became male-gendered. MIT Press, 2012. ISBN: 9780262517966.

[Eppl2006] Martin J. Eppler. «A Comparison Between Concept Maps, Mind Maps, Conceptual Diagrams, and Visual Metaphors as Complementary Tools for Knowledge Construction and Sharing». En: *Information Visualization* 5.3 (jun. de 2006). Compares concept maps, mind maps, conceptual diagrams, and visual metaphors as learning tools., págs. 202-210. DOI: 10.1057/palgrave.ivs.9500131.

[Epst2002] Lewis Carroll Epstein. *Thinking Physics: Understandable Practical Reality*. An entertaining problem-based introduction to thinking like a physicist. Insight Press, 2002. ISBN: 0935218084.

[Eric2017] Barbara J. Ericson, Lauren E. Margulieux y Jochen Rick. «Solving Parsons Problems versus Fixing and Writing Code». En: *2017 Koli Calling Conference on Computing Education Research (Koli'17)*. Reports that solving 2D Parsons problems with distractors takes less time than writing or fixing code but has equivalent learning outcomes. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3141880.3141895.

[Eric2015] Barbara J. Ericson, Steven Moore, Briana B. Morrison y Mark Guzdial. «Usability and Usage of Interactive Features in an Online Ebook for CS Teachers». En: *2015 Workshop in Primary and Secondary Computing Education (WiPSCE'15)*. Reports that learners are more likely to attempt Parsons Problems than nearby multiple choice questions in an ebook. Association for Computing Machinery (ACM), 2015, págs. 111-120. ISBN: 978-1-4503-3753-3. DOI: 10.1145/2818314.2818335.

[Eric2016] K. Anders Ericsson. «Summing Up Hours of Any Type of Practice Versus Identifying Optimal Practice Activities». En: *Perspectives on Psychological Science* 11.3 (mayo de 2016). A critique of a meta-study of deliberate practice based on the latter's overly-broad inclusion of activities., págs. 351-354. DOI: 10.1177/1745691616635600.

[Farm2006] Eugene Farmer. «The Gatekeeper's Guide, or How to Kill a Tool». En: *IEEE Software* 23.6 (nov. de 2006). Ten tongue-in-cheek rules for making sure that a new software tool doesn't get adopted., págs. 12-13. DOI: 10.1109/ms.2006.174.

[Fehi2008] Chris Fehily. *SQL: Visual QuickStart Guide*. Third. An introduction to SQL that is both a good tutorial and a good reference guide. Peachpit Press, 2008. ISBN: 0321553578.

[Finc2012] Sally Fincher, Brad Richards, Janet Finlay, Helen Sharp e Isobel Falconer. «Stories of Change: How Educators Change Their Practice». En: *2012 Frontiers in Education Conference (FIE'12)*. A detailed look at how educators actually adopt new teaching practices. Institute of Electrical y Electronics Engineers (IEEE), oct. de 2012. DOI: 10.1109/fie.2012.6462317.

[Finc2019] Sally Fincher y Anthony Robins, eds. *The Cambridge Handbook of Computing Education Research*. A 900-page summary of what we know about computing education. Cambridge University Press, 2019. ISBN: 978-1108721899.

[Finc2007] Sally Fincher y Josh Tenenberg. «Warren's Question». En: *2007 International Computing Education Research Conference (ICER'07)*. A detailed look at a particular instance of transferring a teaching practice. Association for Computing Machinery (ACM), 2007. DOI: 10.1145/1288580.1288588.

[Fink2013] L. Dee Fink. *Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses*. A step-by-step guide to a systematic lesson design process. Jossey-Bass, 2013. ISBN: 1118124251.

[Fisc2015] Lars Fischer y Stefan Hanenberg. «An empirical investigation of the effects of type systems and code completion on API usability using TypeScript and JavaScript in MS Visual Studio». En: *11th Symposium on Dynamic Languages (DLS'15)*. Found that static typing improved programmer efficiency independently of code completion. ACM Press, 2015. DOI: 10.1145/2816707.2816720.

[Fisl2014] Kathi Fisler. «The Recurring Rainfall Problem». En: *2014 International Computing Education Research Conference (ICER'14)*. Reports that students made fewer low-level errors when solving the Rainfall Problem in a functional language. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2632320.2632346.

[Fitz2008] Sue Fitzgerald, Gary Lewandowski, Renée McCauley, Laurie Murphy, Beth Simon, Lynda Thomas y Carol Zander. «Debugging: Finding, Fixing and Flailing, a Multi-Institutional Study of Novice Debuggers». En: *Computer Science Education* 18.2 (jun. de 2008). Reports that good undergraduate debuggers are good programmers but not necessarily vice versa, and that novices use tracing and testing rather than causal reasoning., págs. 93-116. DOI: 10.1080/08993400802114508.

[Foge2005] Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. The definite guide to managing open source software development projects. O'Reilly Media, 2005. ISBN: 0596007590.

[Ford2016] Denae Ford, Justin Smith, Philip J. Guo y Chris Parnin. «Paradise Unplugged: Identifying Barriers for Female Participation on Stack Overflow». En: *2016 International Symposium on Foundations of Software Engineering (FSE'16)*. Reports that lack of awareness of site features, feeling unqualified

to answer questions, intimidating community size, discomfort interacting with or relying on strangers, and perception that they shouldn't be slacking were seen as significantly more problematic by female Stack Overflow contributors rather than male ones. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2950290.2950331.

[Fran2018] Pablo Frank-Bolton y Rahul Simha. «Docendo Discimus: Students Learn by Teaching Peers Through Video». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Reports that students who make short videos to teach concepts to their peers have a significant increase in their own learning compared to those who only study the material or view videos. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159466.

[Free1972] Jo Freeman. «The Tyranny of Structurelessness». En: *The Second Wave* 2.1 (1972). Points out that every organization has a power structure: the only question is whether it's accountable or not.

[Free2014] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt y M. P. Wenderoth. «Active learning increases student performance in science, engineering, and mathematics». En: *Proc. National Academy of Sciences* 111.23 (mayo de 2014). Presents a meta-analysis of the benefits of active learning., págs. 8410-8415. DOI: 10.1073/pnas.1319030111.

[Frie2016] Marilyn Friend y Lynne Cook. *Interactions: Collaboration Skills for School Professionals*. Eighth. A textbook on how teachers can work with other teachers. Pearson, 2016. ISBN: 0134168542.

[Galp2002] Vashti Galpin. «Women in Computing Around the World». En: *ACM SIGCSE Bulletin* 34.2 (jun. de 2002). Looks at female participation in computing in 35 countries. DOI: 10.1145/543812.543839.

[Gauc2011] Danielle Gaucher, Justin Friesen y Aaron C. Kay. «Evidence that Gendered Wording in Job Advertisements Exists and Sustains Gender Inequality». En: *Journal of Personality and Social Psychology* 101.1 (2011). Reports that gendered wording in job recruitment materials can maintain gender inequality in traditionally male-dominated occupations., págs. 109-128. DOI: 10.1037/a0022530.

[Gawa2007] Atul Gawande. «The Checklist». En: *The New Yorker* (dic. de 2007). Describes the life-saving effects of simple checklists.

[Gawa2011] Atul Gawande. «Personal Best». En: *The New Yorker* (oct. de 2011). Describes how having a coach can improve practice in a variety of fields.

[Gick1987] Mary L. Gick y Keith J. Holyoak. «The Cognitive Basis of Knowledge Transfer». En: *Transfer of Learning: Contemporary Research and Applications*. Ed. por S. J. Cormier y J. D. Hagman. Finds that transference only comes with mastery. Elsevier, 1987, págs. 9-46. DOI: 10.1016/b978-0-12-188950-0.50008-4.

[Gorm2014] Cara Gormally, Mara Evans y Peggy Brickman. «Feedback About Teaching in Higher Ed: Neglected Opportunities to Promote Change». En: *Cell Biology Education* 13.2 (jun. de 2014). Summarizes best practices for providing instructional feedback, and recommends some specific strategies., págs. 187-199. DOI: 10.1187/cbe.13-12-0235.

[Gree2014] Elizabeth Green. *Building a Better Teacher: How Teaching Works (and How to Teach It to Everyone)*. Explains why educational reforms in the past fifty years has mostly missed the mark, and what we should do instead. W. W. Norton & Company, 2014. ISBN: 0393351084.

[Grif2016] Jean M. Griffin. «Learning by Taking Apart». En: *2016 Conference on Information Technology Education (SIGITE'16)*. Reports that people learn to program more quickly by deconstructing code than by writing it. ACM Press, 2016. DOI: 10.1145/2978192.2978231.

[Grov2017] Shuchi Grover y Satabdi Basu. «Measuring Student Learning in Introductory Block-Based Programming». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports that middle-school children using blocks-based programming find loops, variables, and Boolean operators difficult to understand. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017723.

[Gull2004] Ned Gulley. «In Praise of Tweaking». En: *interactions* 11.3 (mayo de 2004). Describes an innovative collaborative coding contest., pág. 18. DOI: 10.1145/986253.986264.

[Guo2013] Philip J. Guo. «Online Python Tutor». En: *2013 Technical Symposium on Computer Science Education (SIGCSE'13)*. Describes the design and use of a web-based execution visualization tool. Association for Computing Machinery (ACM), 2013. DOI: 10.1145/2445196.2445368.

[Guo2014] Philip J. Guo, Juho Kim y Rob Rubin. «How Video Production Affects Student Engagement». En: *2014 Conference on Learning @ Scale (L@S'14)*. Measured learner engagement with MOOC videos and reports that short videos are more engaging than long ones and that talking heads are more engaging than tablet drawings. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2556325.2566239.

[Guzd2013] Mark Guzdial. «Exploring Hypotheses about Media Computation». En: *2013 International Computing Education Research Conference (ICER'13)*. A look back on ten years of media computation research. Association for Computing Machinery (ACM), 2013. DOI: 10.1145/2493394.2493397.

[Guzd2015a] Mark Guzdial. *Learner-Centered Design of Computing Education: Research on Computing for Everyone*. Argues that we must design computing education for everyone, not just people who think they are going to become professional programmers. Morgan & Claypool Publishers, 2015. ISBN: 9781627053518.

[Guzd2015b] Mark Guzdial. *Top 10 Myths About Teaching Computer Science*. https://cacm.acm.org/blogs/blog-cacm/189498-top-10-myths-about-teaching-computer-science/fulltext. Ten things many people believe about teaching computing that simply aren't true. 2015.

[Guzd2016] Mark Guzdial. *Five Principles for Programming Languages for Learners*. https://cacm.acm.org/blogs/blog-cacm/203554-five-principles-for-programming-languages-for-learners/fulltext. Explains how to choose a programming language for people who are new to programming. 2016.

[Haar2017] Lassi Haaranen. «Programming as a Performance - Live-streaming and Its Implications for Computer Science Education». En: *2017 Conference on Innovation and Technology in Computer Science Education (ITiCSE'17)*. An early look at live streaming of coding as a teaching technique. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3059009.3059035.

[Hagg2016] M. S. Hagger, N. L. D. Chatzisarantis, H. Alberts, C. O. Anggono, C. Batailler, A. R. Birt, R. Brand, M. J. Brandt, G. Brewer, S. Bruyneel, D. P. Calvillo, W. K. Campbell, P. R. Cannon, M. Carlucci, N. P. Carruth, T. Cheung, A. Crowell, D. T. D. De Ridder, S. Dewitte, M. Elson, J. R. Evans, B. A. Fay, B. M. Fennis, A. Finley, Z. Francis, E. Heise, H. Hoemann, M. Inzlicht, S. L. Koole, L. Koppel, F. Kroese, F. Lange, K. Lau, B. P. Lynch, C. Martijn, H. Merckelbach, N. V. Mills, A. Michirev, A. Miyake, A. E. Mosser, M. Muise, D. Muller, M. Muzi, D. Nalis, R. Nurwanti, H. Otgaar, M. C. Philipp, P. Primoceri, K. Rentzsch, L. Ringos, C. Schlinkert, B. J. Schmeichel, S. F. Schoch, M. Schrama, A. Schütz, A. Stamos, G. Tinghög, J. Ullrich, M. vanDellen, S. Wimbarti, W. Wolff, C. Yusainy, O. Zerhouni y M. Zwienenberg. «A Multilab Preregistered Replication of the Ego-Depletion Effect». En: *Perspectives on Psychological Science* 11.4 (2016). A meta-analysis that found insufficient evidence to substantiate the ego depletion effect., págs. 546-573. DOI: 10.1177/1745691616652873.

[Hake1998] Richard R. Hake. «Interactive Engagement versus Traditional Methods: A Six-Thousand-Student Survey of Mechanics Test Data for Introductory Physics Courses». En: *American Journal of Physics* 66.1 (ene. de 1998). Reports the use of a concept inventory to measure the benefits of interactive engagement as a teaching technique., págs. 64-74. DOI: 10.1119/1.18809.

[Hamo2017] Sally Hamouda, Stephen H. Edwards, Hicham G. Elmongui, Jeremy V. Ernst y Clifford A. Shaffer. «A Basic Recursion Concept Inventory». En: *Computer Science Education* 27.2 (abr. de 2017). Reports early work on de-

veloping a concept inventory for recursion., págs. 121-148. DOI: 10.1080/08993408.2017.1414728.

[Hank2011] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy y Carol Zander. «Pair Programming in Education: a Literature Review». En: *Computer Science Education* 21.2 (jun. de 2011). Reports increased success rates and retention with pair programming, with some evidence that it is particularly beneficial for women, but finds that scheduling and partner compatibility can be problematic., págs. 135-173. DOI: 10.1080/08993408.2011.579808.

[Hann2010] Jo Erskine Hannay, Erik Arisholm, Harald Engvik y Dag I. K. Sjøberg. «Effects of Personality on Pair Programming». En: *IEEE Transactions on Software Engineering* 36.1 (ene. de 2010). Reports weak correlation between the "Big Five" personality traits and performance in pair programming., págs. 61-80. DOI: 10.1109/tse.2009.41.

[Hann2009] Jo Erskine Hannay, Tore Dybå, Erik Arisholm y Dag I. K. Sjøberg. «The Effectiveness of Pair Programming: A Meta-analysis». En: *Information and Software Technology* 51.7 (jul. de 2009). A comprehensive meta-analysis of research on pair programming., págs. 1110-1122. DOI: 10.1016/j.infsof.2009.02.001.

[Hans2015] John D. Hansen y Justin Reich. «Democratizing education? Examining access and usage patterns in massive open online courses». En: *Science* 350.6265 (dic. de 2015). Reports that MOOCs are mostly used by the affluent., págs. 1245-1248. DOI: 10.1126/science.aab3782.

[Harm2016] Kyle James Harms, Jason Chen y Caitlin L. Kelleher. «Distractors in Parsons Problems Decrease Learning Efficiency for Young Novice Programmers». En: *2016 International Computing Education Research Conference (ICER'16)*. Shows that adding distractors to Parsons Problems does not improve learning outcomes but increases solution times. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2960310.2960314.

[Harr2018] Brian Harrington y Nick Cheng. «Tracing vs. Writing Code: Beyond the Learning Hierarchy». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Finds that the gap between being able to trace code and being able to write it has largely closed by CS2, and that students who still have a gap (in either direction) are likely to do poorly in the course. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159530.

[Hazz2014] Orit Hazzan, Tami Lapidot y Noa Ragonis. *Guide to Teaching Computer Science: An Activity-Based Approach*. Second. A textbook for teaching computer science at the K-12 level with dozens of activities. Springer, 2014. ISBN: 9781447166290.

[Hend2015a] Charles Henderson, Renée Cole, Jeff Froyd, Debra Friedrichsen, Raina Khatri y Courtney Stanford. *Designing Educational Innovations for Sustained Adoption*. A detailed analysis of strategies for getting institutions in higher education to make changes. Increase the Impact, 2015. ISBN: 0996835210.

[Hend2015b] Charles Henderson, Renée Cole, Jeff Froyd, Debra Friedrichsen, Raina Khatri y Courtney Stanford. *Designing Educational Innovations for Sustained Adoption (Executive Summary)*. http://www.increasetheimpact.com/resources.html. A short summary of key points from the authors' work on effecting change in higher education. 2015.

[Hend2017] Carl Hendrick y Robin Macpherson. *What Does This Look Like In The Classroom?: Bridging The Gap Between Research And Practice*. A collection of responses by educational experts to questions asked by classroom teachers, with prefaces by the authors. John Catt Educational, 2017. ISBN: 9781911382379.

[Henr2010] Joseph Henrich, Steven J. Heine y Ara Norenzayan. «The Weirdest People in the World?» En: *Behavioral and Brain Sciences* 33.2-3 (jun. de 2010). Points out that the subjects of most published psychological studies are Western, educated, industrialized, rich, and democratic., págs. 61-83. DOI: 10.1017/s0140525x0999152x.

[Hest1992] David Hestenes, Malcolm Wells y Gregg Swackhamer. «Force Concept Inventory». En: *The Physics Teacher* 30.3 (mar. de 1992). Describes the Force Concept Inventory's motivation, design, and impact., págs. 141-158. DOI: 10.1119/1.2343497.

[Hick2018] Marie Hicks. *Programmed Inequality: How Britain Discarded Women Technologists and Lost Its Edge in Computing*. Describes how Britain lost its early dominance in computing by systematically discriminating against its most qualified workers: women. MIT Press, 2018. ISBN: 9780262535182.

[Hofm2017] Johannes Hofmeister, Janet Siegmund y Daniel V. Holt. «Shorter Identifier Names Take Longer to Comprehend». En: *2017 Conference on Software Analysis, Evolution and Reengineering (SANER'17)*. Reports that using words for variable names makes comprehension faster than using abbreviations or single-letter names for variables. Institute of Electrical y Electronics Engineers (IEEE), feb. de 2017. DOI: 10.1109/saner.2017.7884623.

[Hu2017] Helen H. Hu, Cecily Heiner, Thomas Gagne y Carl Lyman. «Building a Statewide Computer Science Teacher Pipeline». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports that a six-month program for high school teachers converting to teach CS quadruples the number of teachers without noticeable reduction of student outcomes and increases teachers' belief that anyone can program. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017788.

[Hust2012] Therese Huston. *Teaching What You Don't Know*. A pointed, funny, and very useful exploration of exactly what the title says. Harvard University Press, 2012. ISBN: 0674066170.

[Ihan2010] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta y Otto Seppälä. «Review of Recent Systems for Automatic Assessment of Programming Assignments». En: *2010 Koli Calling Conference on Computing Education Research (Koli'10)*. Reviews auto-grading tools of the time. Association for Computing Machinery (ACM), 2010. DOI: 10.1145/1930464.1930480.

[Ihan2011] Petri Ihantola y Ville Karavirta. «Two-dimensional Parson's Puzzles: The Concept, Tools, and First Observations». En: *Journal of Information Technology Education: Innovations in Practice* 10 (2011). Describes a 2D Parsons Problem tool and early experiences with it that confirm that experts solve outside-in rather than line-by-line., págs. 119-132. DOI: 10.28945/1394.

[Ihan2016] Petri Ihantola, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, Daniel Toll, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen y Andrew Petersen. «Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies». En: *2016 Conference on Innovation and Technology in Computer Science Education (ITiCSE'16)*. A survey of methods used in mining and analyzing programming data. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2858796.2858798.

[Ijss2000] Wijnand A. IJsselsteijn, Huib de Ridder, Jonathan Freeman y Steve E. Avons. «Presence: Concept, Determinants, and Measurement». En: *2000 Conference on Human Vision and Electronic Imaging*. Ed. por Bernice E. Rogowitz y Thrasyvoulos N. Pappas. Summarizes thinking of the time about real and virtual presence. SPIE, jun. de 2000. DOI: 10.1117/12.387188.

[Irib2009] Alicia Iriberri y Gondy Leroy. «A Life-Cycle Perspective on Online Community Success». En: *ACM Computing Surveys* 41.2 (feb. de 2009). Reviews research on online communities organized according to a five-stage lifecycle model., págs. 1-29. DOI: 10.1145/1459352.1459356.

[Juss2005] Lee Jussim y Kent D. Harber. «Teacher Expectations and Self-Fulfilling Prophecies: Knowns and Unknowns, Resolved and Unresolved Controversies». En: *Personality and Social Psychology Review* 9.2 (mayo de 2005). A survey of the effects of teacher expectations on student outcomes., págs. 131-155. DOI: 10.1207/s15327957pspr0902_3.

[Kaly2003] Slava Kalyuga, Paul Ayres, Paul Chandler y John Sweller. «The Expertise Reversal Effect». En: *Educational Psychologist* 38.1 (mar. de 2003). Reports that instructional techniques that work well with inexperienced learners lose their effectiveness or have negative consequences when used with more experienced learners., págs. 23-31. DOI: 10.1207/s15326985ep3801_4.

[Kaly2015] Slava Kalyuga y Anne-Marie Singh. «Rethinking the Boundaries of Cognitive Load Theory in Complex Learning». En: *Educational Psychology Review* 28.4 (dic. de 2015). Argues that cognitive load theory is basically micro-management within a broader pedagogical context., págs. 831-852. DOI: 10.1007/s10648-015-9352-0.

[Kang2016] Sean H. K. Kang. «Spaced Repetition Promotes Efficient and Effective Learning». En: *Policy Insights from the Behavioral and Brain Sciences* 3.1 (ene. de 2016). Summarizes research on spaced repetition and what it means for classroom teaching., págs. 12-19. DOI: 10.1177/2372732215624708.

[Kapu2016] Manu Kapur. «Examining Productive Failure, Productive Success, Unproductive Failure, and Unproductive Success in Learning». En: *Educational Psychologist* 51.2 (abr. de 2016). Looks at productive failure as an alternative to inquiry-based learning and approaches based on cognitive load theory., págs. 289-299. DOI: 10.1080/00461520.2016.1155457.

[Karp2008] Jeffrey D. Karpicke y Henry L. Roediger. «The Critical Importance of Retrieval for Learning». En: *Science* 319.5865 (feb. de 2008). Reports that repeated testing improves recall of word lists from 35 % to 80 %, even when learners can still access the material but are not tested on it., págs. 966-968. DOI: 10.1126/science.1152408.

[Kauf2000] Deborah B. Kaufman y Richard M. Felder. «Accounting for Individual Effort in Cooperative Learning Teams». En: *Journal of Engineering Education* 89.2 (2000). Reports that self-rating and peer ratings in undergraduate courses agree, that collusion isn't significant, that students don't inflate their self-ratings, and that ratings are not biased by gender or race.

[Keme2009] Chris F. Kemerer y Mark C. Paulk. «The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data». En: *IEEE Transactions on Software Engineering* 35.4 (jul. de 2009). Uses individual data to explore the effectiveness of code review., págs. 534-550. DOI: 10.1109/tse.2009.27.

[Kepp2008] Jeroen Keppens y David Hay. «Concept Map Assessment for Teaching Computer Programming». En: *Computer Science Education* 18.1 (mar. de 2008). A short review of ways concept mapping can be used in CS education., págs. 31-42. DOI: 10.1080/08993400701864880.

[Kern1983] Brian W. Kernighan y Rob Pike. *The Unix Programming Environment*. An influential early description of Unix. Prentice-Hall, 1983. ISBN: 013937681X.

[Kern1999] Brian W. Kernighan y Rob Pike. *The Practice of Programming*. A programming style manual written by two of the creators of modern computing. Addison-Wesley, 1999. ISBN: 9788177582482.

[Kern1978] Brian W. Kernighan y P. J. Plauger. *The Elements of Programming Style*. Second. An early and influential description of the Unix programming philosophy. McGraw-Hill, 1978. ISBN: 0070342075.

[Kern1988] Brian W. Kernighan y Dennis M. Ritchie. *The C Programming Language*. Second. The book that made C a popular programming language. Prentice-Hall, 1988. ISBN: 0131103628.

[Keun2016a] Hieke Keuning, Johan Jeuring y Bastiaan Heeren. «Towards a Systematic Review of Automated Feedback Generation for Programming Exercises». En: *2016 Conference on Innovation and Technology in Computer Science Education (ITiCSE'16)*. Reports that auto-grading tools often do not give feedback on what to do next, and that teachers cannot easily adapt most of the tools to their needs. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2899415.2899422.

[Keun2016b] Hieke Keuning, Johan Jeuring y Bastiaan Heeren. *Towards a Systematic Review of Automated Feedback Generation for Programming Exercises - Extended Version*. Technical Report UU-CS-2016-001, Utrecht University. An extended look at feedback messages from auto-grading tools. 2016.

[Kim2017] Ada S. Kim y Amy J. Ko. «A Pedagogical Analysis of Online Coding Tutorials». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports that online coding tutorials largely teach similar content, organize content bottom-up, and provide goal-directed practices with immediate feedback, but are not tailored to learners' prior coding knowledge and usually don't tell learners how to transfer and apply knowledge. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017728.

[King1993] Alison King. «From Sage on the Stage to Guide on the Side». En: *College Teaching* 41.1 (ene. de 1993). An early proposal to flip the classroom., págs. 30-35. DOI: 10.1080/87567555.1993.9926781.

[Kirk1994] Donald L. Kirkpatrick. *Evaluating Training Programs: The Four Levels*. Defines a widely-used four-level model for evaluating training. Berrett-Koehle, 1994. ISBN: 1881052494.

[Kirs2006] Paul A. Kirschner, John Sweller y Richard E. Clark. «Why Minimal Guidance During Instruction does not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching». En: *Educational Psychologist* 41.2 (jun. de 2006). Argues that inquiry-based learning is less effective for novices than guided instruction., págs. 75-86. DOI: 10.1207/s15326985ep4102_1.

[Kirs2018] Paul A. Kirschner, John Sweller, Femke Kirschner y Jimmy Zambrano R. «From Cognitive Load Theory to Collaborative Cognitive Load Theory». En: *International Journal of Computer-Supported Collaborative Learning* (abr. de 2018). Extends cognitive load theory to include collaborative aspects of learning. DOI: 10.1007/s11412-018-9277-y.

[Kirs2013] Paul A. Kirschner y Jeroen J. G. van Merriënboer. «Do Learners Really Know Best? Urban Legends in Education». En: *Educational Psychologist* 48.3 (jul. de 2013). Argues that three learning myths—digital natives, learning styles, and self-educators—all reflect the mistaken belief that learners know what is best for them, and cautions that we may be in a downward spiral in which every attempt by education researchers to rebut these myths confirms their opponents' belief that learning science is pseudo-science., págs. 169-183. DOI: 10.1080/00461520.2013.804395.

[Koed2015] Kenneth R. Koedinger, Jihee Kim, Julianna Zhuxin Jia, Elizabeth A. McLaughlin y Norman L. Bier. «Learning is Not a Spectator Sport: Doing is Better than Watching for Learning from a MOOC». En: *2015 Conference on Learning @ Scale (L@S'15)*. Measures the benefits of doing rather than watching. Association for Computing Machinery (ACM), 2015. DOI: 10.1145/2724660.2724681.

[Koeh2013] Matthew J. Koehler, Punya Mishra y William Cain. «What is Technological Pedagogical Content Knowledge (TPACK)?» En: *Journal of Education* 193.3 (2013). Refines the discussion of PCK by adding technology, and sketches strategies for building understanding of how to use it., págs. 13-19. DOI: 10.1177/002205741319300303.

[Kohn2017] Tobias Kohn. «Variable Evaluation: An Exploration of Novice Programmers' Understanding and Common Misconceptions». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports that students often believe in delayed evaluation or that entire equations are stored in variables. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017724.

[Koll2015] Michael Kölling. «Lessons From the Design of Three Educational Programming Environments». En: *International Journal of People-Oriented Programming* 4.1 (ene. de 2015). Compares three generations of programming environments intended for novice use., págs. 5-32. DOI: 10.4018/ijpop.2015010102.

[Krau2016] Robert E. Kraut y Paul Resnick. *Building Successful Online Communities: Evidence-Based Social Design*. Sums up what we actually know about making thriving online communities and why we believe it's true. MIT Press, 2016. ISBN: 0262528916.

[Krug1999] Justin Kruger y David Dunning. «Unskilled and Unaware of it: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments». En: *Journal of Personality and Social Psychology* 77.6 (1999). The original report on the Dunning-Kruger effect: the less people know, the less accurate their estimate of their knowledge., págs. 1121-1134. DOI: 10.1037/0022-3514.77.6.1121.

[Kuch2011] Marc J. Kuchner. *Marketing for Scientists: How to Shine in Tough Times*. A short, readable guide to making people aware of, and care about, your work. Island Press, 2011. ISBN: 1597269948.

[Kuit2004] Marja Kuittinen y Jorma Sajaniemi. «Teaching Roles of Variables in Elementary Programming Courses». En: *ACM SIGCSE Bulletin* 36.3 (sep. de 2004). Presents a few patterns used in novice programming and the pedagogical value of teaching them., pág. 57. DOI: 10.1145/1026487.1008014.

[Kulk2013] Chinmay Kulkarni, Koh Pang Wei, Huy Le, Daniel Chia, Kathryn Papadopoulos, Justin Cheng, Daphne Koller y Scott R. Klemmer. «Peer and Self Assessment in Massive Online Classes». En: *ACM Transactions on Computer-Human Interaction* 20.6 (dic. de 2013). Shows that peer grading can be as effective at scale as expert grading., págs. 1-31. DOI: 10.1145/2505057.

[Laba2008] David F. Labaree. «The Winning Ways of a Losing Strategy: Educationalizing Social Problems in the United States». En: *Educational Theory* 58.4 (nov. de 2008). Explores why the United States keeps pushing the solution of social problems onto educational institutions, and why that continues not to work., págs. 447-460. DOI: 10.1111/j.1741-5446.2008.00299.x.

[Lach2018] Michael Lachney. «Computational Communities: African-American Cultural Capital in Computer Science Education». En: *Computer Science Education* (feb. de 2018). Explores use of community representation and computational integration to bridge computing and African-American cultural capital in CS education., págs. 1-22. DOI: 10.1080/08993408.2018.1429062.

[Lake2018] George Lakey. *How We Win: A Guide to Nonviolent Direct Action Campaigning*. A short experience-based guide to effective campaigning. Melville House, 2018. ISBN: 978-1612197531.

[Lang2013] James M. Lang. *Cheating Lessons: Learning from Academic Dishonesty*. Explores why students cheat, and how courses often give them incentives to do so. Harvard University Press, 2013. ISBN: 0674724631.

[Lang2016] James M. Lang. *Small Teaching: Everyday Lessons from the Science of Learning*. Presents a selection of accessible evidence-based practices that teachers can adopt when they have little time and few resources. Jossey-Bass, 2016. ISBN: 9781118944493.

[Lazo1993] Ard W. Lazonder y Hans van der Meij. «The Minimal Manual: Is Less Really More?» En: *International Journal of Man-Machine Studies* 39.5 (nov. de 1993). Reports that the minimal manual approach to instruction outperforms traditional approaches regardless of prior experience with computers., págs. 729-752. DOI: 10.1006/imms.1993.1081.

[Leak2017] Mackenzie Leake y Colleen M. Lewis. «Recommendations for Designing CS Resource Sharing Sites for All Teachers». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Explores why CS teachers don't use resource sharing sites and recommends ways to make them more appealing. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017780.

[Lee2013] Cynthia Bailey Lee. «Experience Report: CS1 in MATLAB for Non-Majors, with Media Computation and Peer Instruction». En: *2013 Technical Symposium on Computer Science Education (SIGCSE'13)*. Describes an adaptation of media computation to a first-year MATLAB course. Association for Computing Machinery (ACM), 2013. DOI: 10.1145/2445196.2445214.

[Lee2017] Cynthia Bailey Lee. *What Can I Do Today to Create a More Inclusive Community in CS?* http://bit.ly/2oynmSH. A practical checklist of things instructors can do to make their computing classes more inclusive. 2017.

[Lemo2014] Doug Lemov. *Teach Like a Champion 2.0: 62 Techniques that Put Students on the Path to College*. Presents 62 classroom techniques drawn from intensive study of thousands of hours of video of good teachers in action. Jossey-Bass, 2014. ISBN: 1118901851.

[Lewi2015] Colleen M. Lewis y Niral Shah. «How Equity and Inequity Can Emerge in Pair Programming». En: *2015 International Computing Education Research Conference (ICER'15)*. Reports a study of pair programming in a middle-grade classroom in which less equitable pairs were ones that sought to complete the task quickly. Association for Computing Machinery (ACM), 2015. DOI: 10.1145/2787622.2787716.

[List2009] Raymond Lister, Colin Fidge y Donna Teague. «Further Evidence of a Relationship Between Explaining, Tracing and Writing Skills in Introductory Programming». En: *ACM SIGCSE Bulletin* 41.3 (ago. de 2009). Replicates earlier studies showing that students who cannot trace code usually cannot explain code and that students who tend to perform reasonably well at code writing tasks have also usually acquired the ability to both trace code and explain code., pág. 161. DOI: 10.1145/1595496.1562930.

[List2004] Raymond Lister, Otto Seppälä, Beth Simon, Lynda Thomas, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström y Kate Sanders. «A Multi-National Study of Reading and Tracing Skills in Novice Programmers». En: *2004 Conference on Innovation and Technology in Computer Science Education (ITiCSE'04)*. Reports that students are weak at both predicting the outcome of executing a short piece of code and at selecting the correct completion for short pieces of code. Association for Computing Machinery (ACM), 2004. DOI: 10.1145/1044550.1041673.

[Litt2004] Dennis Littky. *The Big Picture: Education Is Everyone's Business*. Essays on the purpose of education and how to make schools better. Association for Supervision & Curriculum Development (ASCD), 2004. ISBN: 0871209713.

[Luxt2009] Andrew Luxton-Reilly. «A Systematic Review of Tools That Support Peer Assessment». En: *Computer Science Education* 19.4 (dic. de 2009). Surveys peer assessment tools that may be of use in computing education., págs. 209-232. DOI: 10.1080/08993400903384844.

[Luxt2017] Andrew Luxton-Reilly, Jacqueline Whalley, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders y Simon. «Developing Assessments to Determine Mastery of Programming Fundamentals». En: *2017 Conference on Innovation and Technology in Computer Science Education (ITiCSE'17)*. Synthesizes work from many previous works to determine what CS instructors are actually teaching, how those things depend on each other, and how they might be assessed. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3174781.3174784.

[Macn2014] Brooke N. Macnamara, David Z. Hambrick y Frederick L. Oswald. «Deliberate Practice and Performance in Music, Games, Sports, Education, and Professions: A Meta-Analysis». En: *Psychological Science* 25.8 (jul. de 2014). A meta-study of the effectiveness of deliberate practice., págs. 1608-1618. DOI: 10.1177/0956797614535810.

[Magu2018] Phil Maguire, Rebecca Maguire y Robert Kelly. «Using Automatic Machine Assessment to Teach Computer Programming». En: *Computer Science Education* (feb. de 2018). Reports that weekly machine-evaluated tests are a better predictor of exam scores than labs (but that students didn't like the system)., págs. 1-18. DOI: 10.1080/08993408.2018.1435113.

[Majo2015] Claire Howell Major, Michael S. Harris y Tod Zakrajsek. *Teaching for Learning: 101 Intentionally Designed Educational Activities to Put Students on the Path to Success*. Catalogs a hundred different kinds of exercises to do with students. Routledge, 2015. ISBN: 0415699363.

[Malo2010] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman y Evelyn Eastmond. «The Scratch Programming Language and Environment». En: *ACM Transactions on Computing Education* 10.4 (nov. de 2010). Summarizes the design of the first generation of Scratch., págs. 1-15. DOI: 10.1145/1868358.1868363.

[Mann2015] Mary Lynn Manns y Linda Rising. *Fearless Change: Patterns for Introducing New Ideas*. A catalog of patterns for making change happen in large organizations. Addison-Wesley, 2015. ISBN: 9780201741575.

[Marc2011] Guillaume Marceau, Kathi Fisler y Shriram Krishnamurthi. «Measuring the Effectiveness of Error Messages Designed for Novice Programmers». En: *2011 Technical Symposium on Computer Science Education (SIGCSE'11)*. Looks at edit-level responses to error messages, and introduces a useful rubric for classifying user responses to errors. Association for Computing Machinery (ACM), 2011. DOI: 10.1145/1953163.1953308.

[Marg2015] Anoush Margaryan, Manuela Bianco y Allison Littlejohn. «Instructional Quality of Massive Open Online Courses (MOOCs)». En: *Computers & Education* 80 (ene. de 2015). Reports that instructional design quality in MOOCs poor, but that the organization and presentation of material is good., págs. 77-83. DOI: 10.1016/j.compedu.2014.08.005.

[Marg2010] Jane Margolis, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme y Kim Nao. *Stuck in the Shallow End: Education, Race, and Computing*. Dissects the school structures and belief systems that lead to under-representation of African American and Latinx students in computing. MIT Press, 2010. ISBN: 0262514044.

[Marg2003] Jane Margolis y Allan Fisher. *Unlocking the Clubhouse: Women in Computing*. A groundbreaking report on the gender imbalance in computing, and the steps Carnegie Mellon took to address the problem. MIT Press, 2003. ISBN: 0262632691.

[Marg2016] Lauren E. Margulieux, Richard Catrambone y Mark Guzdial. «Employing Subgoals in Computer Programming Education». En: *Computer Science Education* 26.1 (ene. de 2016). Reports that labelled subgoals improve learning outcomes in introductory computing courses., págs. 44-67. DOI: 10.1080/08993408.2016.1144429.

[Marg2012] Lauren E. Margulieux, Mark Guzdial y Richard Catrambone. «Subgoal-labeled Instructional Material Improves Performance and Transfer in Learning to Develop Mobile Applications». En: *2012 International Computing Education Research Conference (ICER'12)*. Reports that labelled subgoals improve outcomes and transference when learning about mobile app development. ACM Press, 2012, págs. 71-78. DOI: 10.1145/2361276.2361291.

[Mark2018] Rebecca A. Markovits y Yana Weinstein. «Can Cognitive Processes Help Explain the Success of Instructional Techniques Recommended by Behavior Analysts?» En: *NPJ Science of Learning* 3.1 (ene. de 2018). Points out that behavioralists and cognitive psychologists differ in approach, but wind up making very similar recommendations about how to teach, and gives two specific examples. DOI: 10.1038/s41539-017-0018-1.

[Mars2002] Herbert W. Marsh y John Hattie. «The Relation Between Research Productivity and Teaching Effectiveness: Complementary, Antagonistic, or Independent Constructs?» En: *Journal of Higher Education* 73.5 (2002). One study

of many showing there is zero correlation between research ability and teaching effectiveness., págs. 603-641. DOI: 10.1353/jhe.2002.0047.

[Masa2018] Susana Masapanta-Carrión y J. Ángel Velázquez-Iturbide. «A Systematic Review of the Use of Bloom's Taxonomy in Computer Science Education». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Reports that even experienced educators have trouble agreeing on the correct classification for a question or idea using Bloom's Taxonomy. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159491.

[Maso2016] Raina Mason, Carolyn Seton y Graham Cooper. «Applying Cognitive Load Theory to the Redesign of a Conventional Database Systems Course». En: *Computer Science Education* 26.1 (ene. de 2016). Reports how redesigning a database course using cognitive load theory reduced exam failure rate while increasing student satisfaction., págs. 68-87. DOI: 10.1080/08993408.2016.1160597.

[Matt2019] Eric Matthes. *Python Flash Cards: Syntax, Concepts, and Examples*. Handy flashcards summarizing the core of Python 3. 2019.

[Maye2004] Richard E. Mayer. «Teaching of Subject Matter». En: *Annual Review of Psychology* 55.1 (feb. de 2004). An overview of how and why teaching and learning are subject-specific., págs. 715-744. DOI: 10.1146/annurev.psych.55.082602.133124.

[Maye2009] Richard E. Mayer. *Multimedia Learning*. Second. Presents a cognitive theory of multimedia learning. Cambridge University Press, 2009. ISBN: 9780521735353.

[Maye2003] Richard E. Mayer y Roxana Moreno. «Nine Ways to Reduce Cognitive Load in Multimedia Learning». En: *Educational Psychologist* 38.1 (mar. de 2003). Shows how research into how we absorb and process information can be applied to the design of instructional materials., págs. 43-52. DOI: 10.1207/s15326985ep3801_6.

[Mazu1996] Eric Mazur. *Peer Instruction: A User's Manual*. A guide to implementing peer instruction. Prentice-Hall, 1996.

[McCa2008] Renée McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas y Carol Zander. «Debugging: A Review of the Literature from an Educational Perspective». En: *Computer Science Education* 18.2 (jun. de 2008). Summarizes research about why bugs occur, why types there are, how people debug, and whether we can teach debugging skills., págs. 67-92. DOI: 10.1080/08993400802114581.

[McCr2001] Michael McCracken, Tadeusz Wilusz, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas y Ian Utting. «A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students». En: *2001 Conference on Innovation and Technology in Computer Science Education (ITiCSE'01)*. Reports that most students still struggle to solve even basic programming problems at the end of their introductory course. Association for Computing Machinery (ACM), 2001. DOI: 10.1145/572133.572137.

[McDo2006] Charlie McDowell, Linda Werner, Heather E. Bullock y Julian Fernald. «Pair Programming Improves Student Retention, Confidence, and Program Quality». En: *Communications of the ACM* 49.8 (ago. de 2006). A summary of research showing that pair programming improves retention and confidence., págs. 90-95. DOI: 10.1145/1145287.1145293.

[McGu2015] Saundra Yancey McGuire. *Teach Students How to Learn: Strategies You Can Incorporate Into Any Course to Improve Student Metacognition, Study Skills, and Motivation*. Explains how metacognitive strategies can improve learning. Stylus Publishing, 2015. ISBN: 162036316X.

[McMi2017] Tressie McMillan Cottom. *Lower Ed: The Troubling Rise of For-Profit Colleges in the New Economy*. Lays bare the dynamics of the growing educational industry to show how it leads to greater inequality rather than less. The New Press, 2017. ISBN: 1620970600.

[McTi2013] Jay McTighe y Grant Wiggins. *Understanding by Design Framework*. http : / / www . ascd . org / ASCD / pdf / siteASCD / publications / UbD _ WhitePaper0312.pdf. Summarizes the backward instructional design process. 2013.

[Metc2016] Janet Metcalfe. «Learning from Errors». En: *Annual Review of Psychology* 68.1 (ene. de 2016). Summarizes work on the hypercorrection effect in learning., págs. 465-489. DOI: 10.1146/annurev-psych-010416-044022.

[Meys2018] Mark Meysenburg, Tessa Durham Brooks, Raychelle Burks, Erin Doyle y Timothy Frey. «DIVAS: Outreach to the Natural Sciences Through Image Processing». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Describes early results from a programming course for science undergrads built around image processing. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159537.

[Midw2010] Midwest Academy. *Organizing for Social Change: Midwest Academy Manual for Activists*. Fourth. A training manual for people building progressive social movements. The Forum Press, 2010. ISBN: 0984275215.

[Mill2016b] Craig S. Miller y Amber Settle. «Some Trouble with Transparency: An Analysis of Student Errors with Object-Oriented Python». En: *2016 International Computing Education Research Conference (ICER'16)*. Reports that

students have difficulty with `self` in Python. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2960310.2960327.

[Mill2015] David I. Miller y Jonathan Wai. «The Bachelor's to Ph.D. STEM Pipeline No Longer Leaks More Women Than Men: a 30-year Analysis». En: *Frontiers in Psychology* 6 (feb. de 2015). Shows that the "leaky pipeline" metaphor stopped being accurate some time in the 1990s. DOI: 10.3389/fpsyg.2015.00037.

[Mill1956] George A. Miller. «The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information». En: *Psychological Review* 63.2 (1956). The original paper on the limited size of short-term memory., págs. 81-97. DOI: 10.1037/h0043158.

[Mill2013] Kelly Miller, Nathaniel Lasry, Kelvin Chu y Eric Mazur. «Role of Physics Lecture Demonstrations in Conceptual Learning». En: *Physical Review Special Topics - Physics Education Research* 9.2 (sep. de 2013). Reports a detailed study of what students learn during demonstrations and why. DOI: 10.1103/physrevstper.9.020113.

[Mill2016a] Michelle D. Miller. *Minds Online: Teaching Effectively with Technology*. Describes ways that insights from neuroscience can be used to improve online teaching. Harvard University Press, 2016. ISBN: 0674660021.

[Milt2018] Kate M. Miltner. «Girls Who Coded: Gender in Twentieth Century U.K. and U.S. Computing». En: *Science, Technology, & Human Values* (mayo de 2018). A review of three books about how women were systematically pushed out of computing. DOI: 10.1177/0162243918770287.

[Mina1986] Anne Minahan. «Martha's Rules». En: *Affilia* 1.2 (jun. de 1986). Describes a lightweight set of rules for consensus-based decision making., págs. 53-56. DOI: 10.1177/088610998600100206.

[Miya2018] Toshiya Miyatsu, Khuyen Nguyen y Mark A. McDaniel. «Five Popular Study Strategies: Their Pitfalls and Optimal Implementations». En: *Perspectives on Psychological Science* 13.3 (mayo de 2018). Explains how learners misuse common study strategies and what they should do instead., págs. 390-407. DOI: 10.1177/1745691617710510.

[Mlad2017] Monika Mladenović, Ivica Boljat y Žana Žanko. «Comparing Loops Misconceptions in Block-Based and Text-Based Programming Languages at the K-12 Level». En: *Education and Information Technologies* (nov. de 2017). Reports that K-12 students have fewer misconceptions about loops using Scratch than using Logo or Python, and fewer misconceptions about nested loops with Logo than with Python. DOI: 10.1007/s10639-017-9673-3.

[More2019] Kayla Morehead, John Dunlosky y Katherine A. Rawson. «How Much Mightier Is the Pen than the Keyboard for Note-Taking? A Replication and Extension of Mueller and Oppenheimer (2014)». En: *Educational Psychology Review* (feb. de 2019). Reports a failure to replicate an earlier study comparing note-taking by hand and with computers. DOI: 10.1007/s10648-019-09468-2.

[Morr2016] Briana B. Morrison, Lauren E. Margulieux, Barbara J. Ericson y Mark Guzdial. «Subgoals Help Students Solve Parsons Problems». En: *2016 Technical Symposium on Computer Science Education (SIGCSE'16)*. Reports that students using labelled subgoals solve Parsons Problems better than students without labelled subgoals. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2839509.2844617.

[Muel2014] Pam A. Mueller y Daniel M. Oppenheimer. «The Pen is Mightier than the Keyboard». En: *Psychological Science* 25.6 (abr. de 2014). Presents evidence that taking notes by hand is more effective than taking notes on a laptop., págs. 1159-1168. DOI: 10.1177/0956797614524581.

[Mull2007a] Derek A. Muller, James Bewes, Manjula D. Sharma y Peter Reimann. «Saying the Wrong Thing: Improving Learning with Multimedia by Including Misconceptions». En: *Journal of Computer Assisted Learning* 24.2 (jul. de 2007). Reports that including explicit discussion of misconceptions significantly improves learning outcomes: students with low prior knowledge benefit most and students with more prior knowledge are not disadvantaged., págs. 144-155. DOI: 10.1111/j.1365-2729.2007.00248.x.

[Mull2007b] Orna Muller, David Ginat y Bruria Haberman. «Pattern-Oriented Instruction and Its Influence on Problem Decomposition and Solution Construction». En: *2007 Technical Symposium on Computer Science Education (SIGCSE'07)*. Reports that explicitly teaching solution patterns improves learning outcomes. Association for Computing Machinery (ACM), 2007. DOI: 10.1145/1268784.1268830.

[Murp2008] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas y Carol Zander. «Debugging: The Good, the Bad, and the Quirky - A Qualitative Analysis of Novices' Strategies». En: *ACM SIGCSE Bulletin* 40.1 (feb. de 2008). Reports that many CS1 students use good debugging strategies, but many others don't, and students often don't recognize when they are stuck., pág. 163. DOI: 10.1145/1352322.1352191.

[Nara2018] Sathya Narayanan, Kathryn Cunningham, Sonia Arteaga, William J. Welch, Leslie Maxwell, Zechariah Chawinga y Bude Su. «Upward Mobility for Underrepresented Students». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Describes an intensive 3-year bachelor's program based on tight-knit cohorts and administrative support that tripled graduation rates. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159551.

[Nath2003] Mitchell J. Nathan y Anthony Petrosino. «Expert Blind Spot Among Preservice Teachers». En: *American Educational Research Journal* 40.4 (ene. de 2003). Early work on expert blind spot., págs. 905-928. DOI: 10.3102/00028312040004905.

[Hpl2018] National Academies of Sciences, Engineering, and Medicine. *How People Learn II: Learners, Contexts, and Cultures*. A comprehensive survey of what we know about learning. National Academies Press, 2018. ISBN: 978-0309459648.

[Nils2017] Linda B. Nilson y Ludwika A. Goodson. *Online Teaching at Its Best: Merging Instructional Design with Teaching and Learning Research*. A guide for college instructors that focuses on online teaching. Jossey-Bass, 2017. ISBN: 1119242290.

[Nord2017] Emily Nordmann, Colin Calder, Paul Bishop, Amy Irwin y Darren Comber. *Turn Up, Tune In, Don'T Drop Out: The Relationship Between Lecture Attendance, Use of Lecture Recordings, and Achievement at Different Levels of Study*. https://psyarxiv.com/fd3yj. Reports on the pros and cons of recording lectures. 2017. DOI: 10.17605/OSF.IO/FD3YJ.

[Nutb2016] Stephen Nutbrown y Colin Higgins. «Static Analysis of Programming Exercises: Fairness, Usefulness and a Method for Application». En: *Computer Science Education* 26.2-3 (mayo de 2016). Describes ways auto-grader rules were modified and grades weighted to improve correlation between automatic feedback and manual grades., págs. 104-128. DOI: 10.1080/08993408.2016.1179865.

[Nuth2007] Graham Nuthall. *The Hidden Lives of Learners*. Summarizes a lifetime of work looking at what students actually do in classrooms and how they actually learn. NZCER Press, 2007. ISBN: 1877398241.

[Ojos2015] Bobby Ojose. *Common Misconceptions in Mathematics: Strategies to Correct Them*. A catalog of K-12 misconceptions in mathematics and what to do about them. UPA, 2015. ISBN: 0761858857.

[Ornd2015] Harold N. Orndorff III. «Collaborative Note-Taking: The Impact of Cloud Computing on Classroom Performance». En: *International Journal of Teaching and Learning in Higher Education* 27.3 (2015). Reports that taking notes together online is more effective than solo note-taking., págs. 340-351.

[Ostr2015] Elinor Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. A masterful description and analysis of cooperative governance. Cambridge University Press, 2015. ISBN: 978-1107569782.

[Pape1993] Seymour A. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Second. The foundational text on how computers can underpin a new kind of education. Basic Books, 1993. ISBN: 0465046746.

[Pare2008] Dwayne E. Paré y Steve Joordens. «Peering Into Large Lectures: Examining Peer and Expert Mark Agreement Using peerScholar, an Online Peer Assessment Tool». En: *Journal of Computer Assisted Learning* 24.6 (oct. de 2008). Shows that peer grading by small groups can be as effective as expert grading once accountability features are introduced., págs. 526-540. DOI: 10.1111/j.1365-2729.2008.00290.x.

[Park2015] Thomas H. Park, Brian Dorn y Andrea Forte. «An Analysis of HTML and CSS Syntax Errors in a Web Development Course». En: *ACM Transactions on Computing Education* 15.1 (mar. de 2015). Describes the errors students make in an introductory course on HTML and CSS., págs. 1-21. DOI: 10.1145/2700514.

[Park2016] Miranda C. Parker, Mark Guzdial y Shelly Engleman. «Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment». En: *2016 International Computing Education Research Conference (ICER'16)*. Describes construction and replication of a second concept inventory for basic computing knowledge. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2960310.2960316.

[Parn1986] David Lorge Parnas y Paul C. Clements. «A Rational Design Process: How and Why to Fake It». En: *IEEE Transactions on Software Engineering* SE-12.2 (feb. de 1986). Argues that using a rational design process is less important than looking as though you had., págs. 251-257. DOI: 10.1109/tse.1986.6312940.

[Parn2017] Chris Parnin, Janet Siegmund y Norman Peitek. «On the Nature of Programmer Expertise». En: *Psychology of Programming Interest Group Workshop 2017*. An annotated exploration of what "expertise" means in programming. 2017.

[Pars2006] Dale Parsons y Patricia Haden. «Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses». En: *2006 Australasian Conference on Computing Education (ACE'06)*. The first description of Parson's Problems. Australian Computer Society, 2006, págs. 157-163.

[Part2011] Anu Partanen. *What Americans Keep Ignoring About Finland's School Success*. https://www.theatlantic.com/national/archive/2011/12/what-americans-keep-ignoring-about-finlands-school-success/250564/. Explains that other countries struggle to replicate the success of Finland's schools because they're unwilling to tackle larger social factors. 2011.

[Pati2016] Elizabeth Patitsas, Jesse Berlin, Michelle Craig y Steve Easterbrook. «Evidence that Computer Science Grades are not Bimodal». En: *2016 International Computing Education Research Conference (ICER'16)*. Presents a statistical analysis and an experiment which jointly show that grades in computing classes are not bimodal. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2960310.2960312.

[Pea1986] Roy D. Pea. «Language-Independent Conceptual "Bugs" in Novice Programming». En: *Journal of Educational Computing Research* 2.1 (feb. de 1986). First named the "superbugïn coding: most newcomers think the computer understands what they want, in the same way that a human being would., págs. 25-36. DOI: 10.2190/689t-1r2a-x4w4-29j2.

[Petr2016] Marian Petre y André van der Hoek. *Software Design Decoded: 66 Ways Experts Think*. A short illustrated overview of how expert software developers think. MIT Press, 2016. ISBN: 0262035189.

[Pign2016] Alessandra Pigni. *The Idealist's Survival Kit: 75 Simple Ways to Prevent Burnout*. A guide to staying sane and healthy while doing good. Parallax Press, 2016. ISBN: 1941529348.

[Port2016] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Bailey Lee, Robert McCartney, Daniel Zingaro y Beth Simon. «A Multi-Institutional Study of Peer Instruction in Introductory Computing». En: *2016 Technical Symposium on Computer Science Education (SIGCSE'16)*. Reports that students in introductory programming classes value peer instruction, and that it improves learning outcomes. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2839509.2844642.

[Port2013] Leo Porter, Mark Guzdial, Charlie McDowell y Beth Simon. «Success in Introductory Programming: What Works?» En: *Communications of the ACM* 56.8 (ago. de 2013). Summarizes the evidence that peer instruction, media computation, and pair programming can significantly improve outcomes in introductory programming courses., pág. 34. DOI: 10.1145/2492007.2492020.

[Qian2017] Yizhou Qian y James Lehman. «Students' Misconceptions and Other Difficulties in Introductory Programming». En: *ACM Transactions on Computing Education* 18.1 (oct. de 2017). Summarizes research on student misconceptions about computing., págs. 1-24. DOI: 10.1145/3077618.

[Rago2017] Noa Ragonis y Ronit Shmallo. «On the (Mis)understanding of the `this` Reference». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports that most students do not understood when to use `this`, and that teachers are also often not clear on the subject. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017715.

[Raj2018] Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson y Erica Rosenfeld Halverson. «Role of Live-Coding in Learning Introductory Programming». En: *2018 Koli Calling International Conference on Computing Education Research (Koli'18)*. A grounded theory analysis of live coding that includes references to previous works. 2018. DOI: 10.1145/3279720.3279725.

[Rams2019] G. Ramsay, A. B. Haynes, S. R. Lipsitz, I. Solsky, J. Leitch, A. A. Gawande y M. Kumar. «Reducing surgical mortality in Scotland by use of the WHO Surgical Safety Checklist». En: *BJS* (abr. de 2019). Found that the

introduction of surgical checklists in Scottish hospitals significantly reduced mortality rates. DOI: 10.1002/bjs.11151.

[Raws2014] Katherine A. Rawson, Ruthann C. Thomas y Larry L. Jacoby. «The Power of Examples: Illustrative Examples Enhance Conceptual Learning of Declarative Concepts». En: *Educational Psychology Review* 27.3 (jun. de 2014). Reports that presenting examples helps students understand definitions, so long as examples and definitions are interleaved., págs. 483-504. DOI: 10.1007/s10648-014-9273-3.

[Ray2014] Eric J. Ray y Deborah S. Ray. *Unix and Linux: Visual QuickStart Guide*. Fifth. An introduction to Unix that is both a good tutorial and a good reference guide. Peachpit Press, 2014. ISBN: 0321997549.

[Rice2018] Gail Taylor Rice. *Hitting Pause: 65 Lecture Breaks to Refresh and Reinforce Learning*. Justifies and catalogs ways to take a pause in class to help learning. Stylus Publishing, 2018. ISBN: 9781620366530.

[Rich2017] Kathryn M. Rich, Carla Strickland, T. Andrew Binkowski, Cheryl Moran y Diana Franklin. «K-8 learning Trajectories Derived from Research Literature». En: *2017 International Computing Education Research Conference (ICER'17)*. Presents learning trajectories for K-8 computing classes for Sequence, Repetition, and Conditions gleaned from the literature. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3105726.3106166.

[Ritz2018] Anna Ritz. «Programming the Central Dogma: An Integrated Unit on Computer Science and Molecular Biology Concepts». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Describes an introductory computing course for biologists whose problems are drawn from the DNA-to-protein processes in cells. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159590.

[Robe2017] Eric Roberts. *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments: Annotated Findings*. http://cs.stanford.edu/people/eroberts/ResourcesForTheCSCapacityCrisis/files/AnnotatedFindings.pptx. Summarizes findings from a National Academies study about computer science enrollments. 2017.

[Robi2005] Evan Robinson. *Why Crunch Mode Doesn't Work: 6 Lessons*. http://www.igda.org/articles/erobinson_crunch.php. Summarizes research on the effects of overwork and sleep deprivation. 2005.

[Roge2018] Steven G. Rogelberg. *The Surprising Science of Meetings*. A short summary of research on effective meetings. Oxford University Press, 2018. ISBN: 978-0190689216.

[Rohr2015] Doug Rohrer, Robert F. Dedrick y Sandra Stershic. «Interleaved Practice Improves Mathematics Learning». En: *Journal of Educational Psychology* 107.3 (2015). Reports that interleaved practice is more effective than monotonous practice when learning., págs. 900-908. DOI: 10.1037/edu0000001.

[Rubi2013] Marc J. Rubin. «The Effectiveness of Live-coding to Teach Introductory Programming». En: *2013 Technical Symposium on Computer Science Education (SIGCSE'13)*. Reports that live coding is as good as or better than using static code examples. Association for Computing Machinery (ACM), 2013, págs. 651-656. DOI: 10.1145/2445196.2445388.

[Rubi2014] Manuel Rubio-Sánchez, Päivi Kinnunen, Cristóbal Pareja-Flores y J. Ángel Velázquez-Iturbide. «Student Perception and Usage of an Automated Programming Assessment Tool». En: *Computers in Human Behavior* 31 (feb. de 2014). Describes use of an auto-grader for student assignments., págs. 453-460. DOI: 10.1016/j.chb.2013.04.001.

[Sahl2015] Pasi Sahlberg. *Finnish Lessons 2.0: What Can the World Learn from Educational Change in Finland?* A frank look at the success of Finland's educational system and why other countries struggle to replicate it. Teachers College Press, 2015. ISBN: 978-0807755853.

[Saja2006] Jorma Sajaniemi, Mordechai Ben-Ari, Pauli Byckling, Petri Gerdt y Yevgeniya Kulikova. «Roles of Variables in Three Programming Paradigms». En: *Computer Science Education* 16.4 (dic. de 2006). A detailed look at the authors' work on roles of variables., págs. 261-279. DOI: 10.1080/08993400600874584.

[Sala2017] Giovanni Sala y Fernand Gobet. «Does Far Transfer Exist? Negative Evidence From Chess, Music, and Working Memory Training». En: *Current Directions in Psychological Science* 26.6 (oct. de 2017). A meta-analysis showing that far transfer rarely occurs., págs. 515-520. DOI: 10.1177/0963721417712760.

[Sand2013] Kate Sanders, Jaime Spacco, Marzieh Ahmadzadeh, Tony Clear, Stephen H. Edwards, Mikey Goldweber, Chris Johnson, Raymond Lister, Robert McCartney y Elizabeth Patitsas. «The Canterbury QuestionBank: Building a Repository of Multiple-Choice CS1 and CS2 Questions». En: *2013 Conference on Innovation and Technology in Computer Science Education (ITiCSE'13)*. Describes development of a shared question bank for introductory CS, and patterns for multiple choice questions that emerged from entries. Association for Computing Machinery (ACM), 2013. DOI: 10.1145/2543882.2543885.

[Scan1989] David A. Scanlan. «Structured Flowcharts Outperform Pseudocode: An Experimental Comparison». En: *IEEE Software* 6.5 (sep. de 1989). Reports that students understand flowcharts better than pseudocode if both are equally well structured., págs. 28-36. DOI: 10.1109/52.35587.

[Scho1984] Donald A. Schön. *The Reflective Practitioner: How Professionals Think In Action*. A groundbreaking look at how professionals in different fields actually solve problems. Basic Books, 1984. ISBN: 0465068782.

[Schw2013] Viviane Schwarz. *Welcome to Your Awesome Robot*. A wonderful illustrated guide to building wearable cardboard robot suits. Not just for kids. Flying Eye Books, 2013. ISBN: 978-1909263000.

[Scot1987] James C. Scott. *Weapons of the Weak: Everyday Forms of Peasant Resistance*. Describes the techniques of evasion and resistance that the weak use to resist the strong. Yale University Press, 1987. ISBN: 978-0300036411.

[Scot1998] James C. Scott. *Seeing Like a State: How Certain Schemes to Improve the Human Condition Have Failed*. Argues that large organizations consistently prefer uniformity over productivity. Yale University Press, 1998. ISBN: 0300078153.

[Sent2018] Sue Sentance, Erik Barendsen y Carsten Schulte, eds. *Computer Science Education: Perspectives on Teaching and Learning in School*. A collection of academic survey articles on teaching computing. Bloomsbury Press, 2018. ISBN: 135005710X.

[Sent2019] Sue Sentance, Jane Waite y Maria Kallia. «Teachers' Experiences of using PRIMM to Teach Programming in School». En: *2019 Technical Symposium on Computer Science Education (SIGCSE'19)*. Describes PRIMM and its effectiveness. ACM Press, 2019. DOI: 10.1145/3287324.3287477.

[Sepp2015] Otto Seppälä, Petri Ihantola, Essi Isohanni, Juha Sorva y Arto Vihavainen. «Do We Know How Difficult the Rainfall Problem Is?» En: *2015 Koli Calling Conference on Computing Education Research (Koli'15)*. A meta-study of the Rainfall Problem. ACM Press, 2015. DOI: 10.1145/2828959.2828963.

[Shap2007] Jenessa R. Shapiro y Steven L. Neuberg. «From Stereotype Threat to Stereotype Threats: Implications of a Multi-Threat Framework for Causes, Moderators, Mediators, Consequences, and Interventions». En: *Personality and Social Psychology Review* 11.2 (mayo de 2007). Explores the ways the term "stereotype threat" has been used., págs. 107-130. DOI: 10.1177/1088868306294790.

[Shel2017] Duane F. Shell, Leen-Kiat Soh, Abraham E. Flanigan, Markeya S. Peteranetz y Elizabeth Ingraham. «Improving Students' Learning and Achievement in CS Classrooms Through Computational Creativity Exercises that Integrate Computational and Creative Thinking». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Reports that having students work in small groups on computational creativity exercises improves learning outcomes. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017718.

[Shol2019] Dan Sholler, Igor Steinmacher, Denae Ford, Mara Averick, Mike Hoye y Greg Wilson. *Ten Simple Rules for Helping Newcomers Become Contributors to Open Source Projects*. https://github.com/gvwilson/10-newcomers/. Evidence-based practices for helping newcomers become productive in open projects. 2019.

[Simo2013] Simon. «Soloway's Rainfall Problem has Become Harder». En: *2013 Conference on Learning and Teaching in Computing and Engineering*. Argues that the Rainfall Problem is harder for novices than it used to be because they're not used to handling keyboard input, so direct comparison with past results may be unfair. Institute of Electrical y Electronics Engineers (IEEE), mar. de 2013. DOI: 10.1109/latice.2013.44.

[Sing2012] Vandana Singh. «Newcomer integration and learning in technical support communities for open source software». En: *2012 ACM International Conference on Supporting Group Work - GROUP'12*. An early study of onboarding in open source. ACM Press, 2012. DOI: 10.1145/2389176.2389186.

[Sirk2012] Teemu Sirkiä y Juha Sorva. «Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises». En: *2012 Koli Calling Conference on Computing Education Research (Koli'12)*. Analyzes data from student use of an execution visualization tool and classifies common mistakes. Association for Computing Machinery (ACM), 2012. DOI: 10.1145/2401796.2401799.

[Sisk2018] Victoria F. Sisk, Alexander P. Burgoyne, Jingze Sun, Jennifer L. Butler y Brooke N. Macnamara. «To What Extent and Under Which Circumstances Are Growth Mind-Sets Important to Academic Achievement? Two Meta-Analyses». En: *Psychological Science* (mar. de 2018). Reports meta-analyses of the relationship between mind-set and academic achievement, and the effectiveness of mind-set interventions on academic achievement, and finds that overall effects are weak for both, but some results support specific tenets of the theory., pág. 095679761773970. DOI: 10.1177/0956797617739704.

[Skud2014] Ben Skudder y Andrew Luxton-Reilly. «Worked Examples in Computer Science». En: *2014 Australasian Computing Education Conference, (ACE'14)*. A summary of research on worked examples as applied to computing education. 2014.

[Smar2018] Benjamin L. Smarr y Aaron E. Schirmer. «3.4 Million Real-World Learning Management System Logins Reveal the Majority of Students Experience Social Jet Lag Correlated with Decreased Performance». En: *Scientific Reports* 8.1 (mar. de 2018). Reports that students who have to work outside their natural body clock cycle do less well. DOI: 10.1038/s41598-018-23044-8.

[Smit2009] Michelle K. Smith, William B. Wood, Wendy K. Adams, Carl E. Wieman, Jennifer K. Knight, N. Guild y T. T. Su. «Why Peer Discussion Improves Student Performance on In-class Concept Questions». En: *Science* 323.5910 (ene. de 2009). Reports that student understanding increases during discussion in peer instruction, even when none of the students in the group initially know the right answer., págs. 122-124. DOI: 10.1126/science.1165919.

[Solo1986] Elliot Soloway. «Learning to Program = Learning to Construct Mechanisms and Explanations». En: *Communications of the ACM* 29.9 (sep. de 1986). Analyzes programming in terms of choosing appropriate goals and constructing plans to achieve them, and introduces the Rainfall Problem., págs. 850-858. DOI: 10.1145/6592.6594.

[Solo1984] Elliot Soloway y Kate Ehrlich. «Empirical Studies of Programming Knowledge». En: *IEEE Transactions on Software Engineering* SE-10.5 (sep. de 1984). Proposes that experts have programming plans and rules of programming discourse., págs. 595-609. DOI: 10.1109/tse.1984.5010283.

[Sond2012] Harald Søndergaard y Raoul A. Mulder. «Collaborative Learning Through Formative Peer Review: Pedagogy, Programs and Potential». En: *Computer Science Education* 22.4 (dic. de 2012). Surveys literature on student peer assessment, distinguishing grading and reviewing as separate forms, and summarizes features a good peer review system needs to have., págs. 343-367. DOI: 10.1080/08993408.2012.728041.

[Sorv2013] Juha Sorva. «Notional Machines and Introductory Programming Education». En: *ACM Transactions on Computing Education* 13.2 (jun. de 2013). Reviews literature on programming misconceptions, and argues that instructors should address notional machines as an explicit learning objective., págs. 1-31. DOI: 10.1145/2483710.2483713.

[Sorv2018] Juha Sorva. «Misconceptions and the Beginner Programmer». En: *Computer Science Education: Perspectives on Teaching and Learning in School*. Ed. por Sue Sentance, Erik Barendsen y Carsten Schulte. Summarizes what we know about what novices misunderstand about computing. Bloomsbury Press, 2018. ISBN: 135005710X.

[Sorv2014] Juha Sorva y Otto Seppälä. «Research-based Design of the First Weeks of CS1». En: *2014 Koli Calling Conference on Computing Education Research (Koli'14)*. Proposes three cognitively plausible frameworks for the design of a first CS course. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2674683.2674690.

[Spal2014] Dan Spalding. *How to Teach Adults: Plan Your Class, Teach Your Students, Change the World*. A short guide to teaching adult free-range learners informed by the author's social activism. Jossey-Bass, 2014. ISBN: 1118841360.

[Spoh1985] James C. Spohrer, Elliot Soloway y Edgar Pope. «A Goal/Plan Analysis of Buggy Pascal Programs». En: *Human-Computer Interaction* 1.2 (jun. de 1985). One of the first cognitively plausible analyses of how people program, which proposes a goal/plan model., págs. 163-207. DOI: 10.1207/s15327051hci0102_4.

[Srid2016] Sumukh Sridhara, Brian Hou, Jeffrey Lu y John DeNero. «Fuzz Testing Projects in Massive Courses». En: *2016 Conference on Learning @ Scale (L@S'16)*. Reports that fuzz testing student code catches errors that are missed by handwritten test suite, and explains how to safely share tests and results. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2876034.2876050.

[Stam2014] Eliane Stampfer Wiese y Kenneth R. Koedinger. «Investigating Scaffolds for Sense Making in Fraction Addition and Comparison». En: *2014 Annual Conference of the Cognitive Science Society (CogSci'14)*. Looks at how to scaffold learning of fraction operations. 2014.

[Stam2013] Eliane Stampfer y Kenneth R. Koedinger. «When Seeing Isn't Believing: Influences of Prior Conceptions and Misconceptions». En: *2013 Annual Meeting of the Cognitive Science Society (CogSci'13)*. Explores why giving children more information when they are learning about fractions can lower their performance. 2013.

[Star2014] Philip Stark y Richard Freishtat. «An Evaluation of Course Evaluations». En: *ScienceOpen Research* (sep. de 2014). Yet another demonstration that teaching evaluations don't correlate with learning outcomes, and that they are frequently statistically suspect. DOI: 10.14293/s2199-1006.1.sor-edu.aofrqa.v1.

[Stas1998] John Stasko, John Domingue, Mark H. Brown y Blaine A. Price, eds. *Software Visualization: Programming as a Multimedia Experience*. A survey of program and algorithm visualization techniques and results. MIT Press, 1998. ISBN: 0262193957.

[Stee2011] Claude M. Steele. *Whistling Vivaldi: How Stereotypes Affect Us and What We Can Do*. Explains and explores stereotype threat and strategies for addressing it. W. W. Norton & Company, 2011. ISBN: 0393339726.

[Stef2017] Andreas Stefik, Patrick Daleiden, Diana Franklin, Stefan Hanenberg, Antti-Juhani Kaijanaho, Walter Tichy y Brett A. Becker. *Programming Languages and Learning*. https://quorumlanguage.com/evidence.html. Summarizes what we actually know about designing programming languages and why we believe it's true. 2017.

[Stef2013] Andreas Stefik y Susanna Siebert. «An Empirical Investigation into Programming Language Syntax». En: *ACM Transactions on Computing Education* 13.4 (nov. de 2013). Reports that curly-brace languages are as hard to learn

as a language with randomly-designed syntax, but others are easier., págs. 1-40. DOI: 10.1145/2534973.

[Steg2014] Martijn Stegeman, Erik Barendsen y Sjaak Smetsers. «Towards an Empirically Validated Model for Assessment of Code Quality». En: *2014 Koli Calling Conference on Computing Education Research (Koli'14)*. Presents a code quality rubric for novice programming courses. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2674683.2674702.

[Steg2016a] Martijn Stegeman, Erik Barendsen y Sjaak Smetsers. «Designing a Rubric for Feedback on Code Quality in Programming Courses». En: *2016 Koli Calling Conference on Computing Education Research (Koli'16)*. Describes several iterations of a code quality rubric for novice programming courses. Association for Computing Machinery (ACM), 2016. DOI: 10.1145/2999541.2999555.

[Steg2016b] Martijn Stegeman, Erik Barendsen y Sjaak Smetsers. *Rubric for Feedback on Code Quality in Programming Courses*. http://stgm.nl/quality. Presents a code quality rubric for novice programming. 2016.

[Stei2016] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude y Marco Aurélio Gerosa. «Overcoming open source project entry barriers with a portal for newcomers». En: *2016 International Conference on Software Engineering (ICSE'16)*. Reports the effectiveness of a portal specifically designed to help newcomers. ACM Press, 2016. DOI: 10.1145/2884781.2884806.

[Stei2018] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese y Marco Aurélio Gerosa. «Almost There: A Study on Quasi-Contributors in Open-Source Software Projects». En: *2018 International Conference on Software Engineering (ICSE'18)*. Look at why external developers fail to get their contributions accepted into open source projects. ACM Press, 2018. DOI: 10.1145/3180155.3180208.

[Stei2013] Igor Steinmacher, Igor Wiese, Ana Paula Chaves y Marco Aurelio Gérosa. «Why do newcomers abandon open source software projects?» En: *2013 International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'13)*. Explores why new members *don't* stay in open source projects. Institute of Electrical y Electronics Engineers (IEEE), mayo de 2013. DOI: 10.1109/chase.2013.6614728.

[Stoc2018] Jean Stockard, Timothy W. Wood, Cristy Coughlin y Caitlin Rasplica Khoury. «The Effectiveness of Direct Instruction Curricula: A Meta-analysis of a Half Century of Research». En: *Review of Educational Research* (ene. de 2018). A meta-analysis that finds significant positive benefit for Direct Instruction., pág. 003465431775191. DOI: 10.3102/0034654317751919.

[Sung2012] Eunmo Sung y Richard E. Mayer. «When Graphics Improve Liking but not Learning from Online Lessons». En: *Computers in Human Behavior* 28.5 (sep. de 2012). Reports that students who receive any kind of graphics give significantly higher satisfaction ratings than those who don't, but only students who get instructive graphics perform better than groups that get no graphics, seductive graphics, or decorative graphics., págs. 1618-1625. DOI: 10.1016/j.chb.2012.03.026.

[Sved2016] Maria Svedin y Olle Bälter. «Gender Neutrality Improved Completion Rate for All». En: *Computer Science Education* 26.2-3 (jul. de 2016). Reports that redesigning an online course to be gender neutral improves completion probability in general, but decreases it for students with a superficial approach to learning., págs. 192-207. DOI: 10.1080/08993408.2016.1231469.

[Tedr2008] Matti Tedre y Erkki Sutinen. «Three Traditions of Computing: What Educators Should Know». En: *Computer Science Education* 18.3 (sep. de 2008). Summarizes the history and views of three traditions in computing: mathematical, scientific, and engineering., págs. 153-170. DOI: 10.1080/08993400802332332.

[Tew2011] Allison Elliott Tew y Mark Guzdial. «The FCS1: A Language Independent Assessment of CS1 Knowledge». En: *2011 Technical Symposium on Computer Science Education (SIGCSE'11)*. Describes development and validation of a language-independent assessment instrument for CS1 knowledge. Association for Computing Machinery (ACM), 2011. DOI: 10.1145/1953163.1953200.

[Thay2017] Kyle Thayer y Amy J. Ko. «Barriers Faced by Coding Bootcamp Students». En: *2017 International Computing Education Research Conference (ICER'17)*. Reports that coding bootcamps are sometimes useful, but quality is varied, and formal and informal barriers to employment remain. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3105726.3106176.

[Ubel2017] Robert Ubell. *How the Pioneers of the MOOC got It Wrong*. http://spectrum.ieee.org/tech-talk/at-work/education/how-the-pioneers-of-the-mooc-got-it-wrong. A brief exploration of why MOOCs haven't lived up to initial hype. 2017.

[Urba2014] David R. Urbach, Anand Govindarajan, Refik Saskin, Andrew S. Wilton y Nancy N. Baxter. «Introduction of Surgical Safety Checklists in Ontario, Canada». En: *New England Journal of Medicine* 370.11 (mar. de 2014). Reports a study showing that the introduction of surgical checklists did not have a significant effect on operative outcomes., págs. 1029-1038. DOI: 10.1056/nejmsa1308261.

[Utti2013] Ian Utting, Juha Sorva, Tadeusz Wilusz, Allison Elliott Tew, Michael McCracken, Lynda Thomas, Dennis Bouvier, Roger Frye, James Paterson, Michael E. Caspersen y Yifat Ben-David Kolikant. «A Fresh Look at Novice Pro-

grammers' Performance and Their Teachers' Expectations». En: *2013 Conference on Innovation and Technology in Computer Science Education (ITiCSE'13)*. Replicates an earlier study showing how little students learn in their first programming course. ACM Press, 2013. DOI: 10.1145/2543882.2543884.

[Uttl2017] Bob Uttl, Carmela A. White y Daniela Wong Gonzalez. «Meta-analysis of Faculty's Teaching Effectiveness: Student Evaluation of Teaching Ratings and Student Learning are not Related». En: *Studies in Educational Evaluation* 54 (sep. de 2017). Summarizes studies showing that how students rate a course and how much they actually learn are not related., págs. 22-42. DOI: 10.1016/j.stueduc.2016.08.007.

[Varm2015] Roli Varma y Deepak Kapur. «Decoding Femininity in Computer Science in India». En: *Communications of the ACM* 58.5 (abr. de 2015). Reports female participation in computing in India., págs. 56-62. DOI: 10.1145/2663339.

[Vell2017] Mickey Vellukunnel, Philip Buffum, Kristy Elizabeth Boyer, Jeffrey Forbes, Sarah Heckman y Ketan Mayer-Patel. «Deconstructing the Discussion Forum: Student Questions and Computer Science Learning». En: *2017 Technical Symposium on Computer Science Education (SIGCSE'17)*. Found that students mostly ask constructivist and logistical questions in forums, and that the former correlate with grades. Association for Computing Machinery (ACM), 2017. DOI: 10.1145/3017680.3017745.

[Viha2014] Arto Vihavainen, Jonne Airaksinen y Christopher Watson. «A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success». En: *2014 International Computing Education Research Conference (ICER'14)*. Consolidates studies of CS1-level teaching changes and finds media computation the most effective, while introducing a game theme is the least effective. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2632320.2632349.

[Wall2009] Thorbjorn Walle y Jo Erskine Hannay. «Personality and the Nature of Collaboration in Pair Programming». En: *2009 International Symposium on Empirical Software Engineering and Measurement (ESER'09)*. Reports that pairs with different levels of a given personality trait communicated more intensively. Institute of Electrical y Electronics Engineers (IEEE), oct. de 2009. DOI: 10.1109/esem.2009.5315996.

[Wang2018] April Y. Wang, Ryan Mitts, Philip J. Guo y Parmit K. Chilana. «Mismatch of Expectations: How Modern Learning Resources Fail Conversational Programmers». En: *2018 Conference on Human Factors in Computing Systems (CHI'18)*. Reports that learning resources don't really help conversational programmers (those who learn coding to take part in technical discussions). Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3173574.3174085.

[Ward2015] James Ward. *Adventures in Stationery: A Journey Through Your Pencil Case*. A wonderful look at the everyday items that would be in your desk drawer if someone hadn't walked off with them. Profile Books, 2015. ISBN: 1846686164.

[Wats2014] Christopher Watson y Frederick W. B. Li. «Failure Rates in Introductory Programming Revisited». En: *2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE'14)*. A larger version of an earlier study that found an average of one third of students fail CS1. Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2591708.2591749.

[Watt2014] Audrey Watters. *The Monsters of Education Technology*. A collection of essays about the history of educational technology and the exaggerated claims repeatedly made for it. CreateSpace, 2014. ISBN: 1505225051.

[Wein2018a] Yana Weinstein, Christopher R. Madan y Megan A. Sumeracki. «Teaching the Science of Learning». En: *Cognitive Research: Principles and Implications* 3.1 (ene. de 2018). A tutorial review of six evidence-based learning practices. DOI: 10.1186/s41235-017-0087-y.

[Wein2018b] Yana Weinstein, Megan Sumeracki y Oliver Caviglioli. *Understanding How We Learn: A Visual Guide*. A short graphical summary of effective learning strategies. Routledge, 2018. ISBN: 978-1138561724.

[Wein2017] David Weintrop y Uri Wilensky. «Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms». En: *ACM Transactions on Computing Education* 18.1 (oct. de 2017). Reports that students learn faster and better with blocks than with text., págs. 1-25. DOI: 10.1145/3089799.

[Weng2015] Etienne Wenger-Trayner y Beverly Wenger-Trayner. *Communities of Practice: A Brief Introduction*. http://wenger-trayner.com/intro-to-cops/. A brief summary of what communities of practice are and aren't. 2015.

[Wibu2016] Karin Wiburg, Julia Parra, Gaspard Mucundanyi, Jennifer Green y Nate Shaver, eds. *The Little Book of Learning Theories*. Second. Presents brief summaries of various theories of learning. CreateSpace, 2016. ISBN: 1537091808.

[Wigg2005] Grant Wiggins y Jay McTighe. *Understanding by Design*. A lengthy presentation of reverse instructional design. Association for Supervision & Curriculum Development (ASCD), 2005. ISBN: 1416600353.

[Wilc2018] Chris Wilcox y Albert Lionelle. «Quantifying the Benefits of Prior Programming Experience in an Introductory Computer Science Course». En: *2018 Technical Symposium on Computer Science Education (SIGCSE'18)*. Reports that students with prior experience outscore students without in CS1, but there is no significant difference in performance by the end of CS2; also finds that female students with prior exposure outperform their male peers in all areas,

but are consistently less confident in their abilities. Association for Computing Machinery (ACM), 2018. DOI: 10.1145/3159450.3159480.

[Wile2002] David Wiley. *The Reusability Paradox*. http://opencontent.org/docs/paradox.html. Summarizes the tension between learning objects being effective and reusable. 2002.

[Wilk2011] Richard Wilkinson y Kate Pickett. *The Spirit Level: Why Greater Equality Makes Societies Stronger*. Presents evidence that inequality harms everyone, both economically and otherwise. Bloomsbury Press, 2011. ISBN: 1608193411.

[Will2010] Daniel T. Willingham. *Why Don't Students Like School?: A Cognitive Scientist Answers Questions about How the Mind Works and What It Means for the Classroom*. A cognitive scientist looks at how the mind works in the classroom. Jossey-Bass, 2010. ISBN: 047059196X.

[Wils2016] Greg Wilson. «Software Carpentry: Lessons Learned». En: *F1000Research* (ene. de 2016). A history and analysis of Software Carpentry. DOI: 10.12688/f1000research.3-62.v2.

[Wils2007] Karen Wilson y James H. Korn. «Attention During Lectures: Beyond Ten Minutes». En: *Teaching of Psychology* 34.2 (jun. de 2007). Reports little support for the claim that students only have a 10–15 minute attention span (though there is lots of individual variation)., págs. 85-89. DOI: 10.1080/00986280701291291.

[Xie2019] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li y Amy J. Ko. «A theory of instruction for introductory programming skills». En: *Computer Science Education* 29.2-3 (ene. de 2019). Lays out a four-part theory for teaching novices based on reading vs. writing and code vs. templates., págs. 205-253. DOI: 10.1080/08993408.2019.1565235.

[Yada2016] Aman Yadav, Sarah Gretter, Susanne Hambrusch y Phil Sands. «Expanding Computer Science Education in Schools: Understanding Teacher Experiences and Challenges». En: *Computer Science Education* 26.4 (dic. de 2016). Summarizes feedback from K-12 teachers on what they need by way of preparation and support., págs. 235-254. DOI: 10.1080/08993408.2016.1257418.

[Yang2015] Yu-Fen Yang y Yuan-Yu Lin. «Online Collaborative Note-Taking Strategies to Foster EFL Beginners' Literacy Development». En: *System* 52 (ago. de 2015). Reports that students using collaborative note taking when learning English as a foreign language do better than those who don't., págs. 127-138. DOI: 10.1016/j.system.2015.05.006.

A

Licencia

Este es un resumen de lectura sencilla para personas (y no un sustituto) de la licencia. Por favor mira https://creativecommons.org/licenses/by-nc/4.0/legalcode para el texto legal completo.

Este trabajo se licencia bajo Creative Commons Atribución – No Comercial 4.0[1] (CC-BY-NC-4.0).

Eres libre de:

- **Compartir**—copiar y redistribuir el material en cualquier medio o formato
- **Adaptar**—reacomodar, transformar y construir sobre el material.

El/la licenciante no puede revocar estas libertades mientras sigas los términos de la licencia.

Bajo los siguientes términos:

- **Atribución**—Debes dar el crédito apropiado, proporcionar un enlace a la licencia e indicar si se hicieron cambios. Puedes hacerlo de cualquier manera razonable, pero siempre que no sugiriera que el/la licenciante te respalda a ti o al uso que le das al material.

- **No Comercial**—No puedes utilizar el material con fines comerciales.

Sin restricciones adicionales—No puedes aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros/as de hacer cualquier cosa que la licencia permita.

Avisos:

- No tienes que cumplir con la licencia para aquellos elementos del material que son de dominio público o cuando su uso esté permitido por una excepción o limitación aplicable.

[1] https://creativecommons.org/licenses/by-nc/4.0/

- No se otorgan garantías. Es posible que la licencia no otorgue todos los permisos necesarios para el uso que se pretende dar al material. Por ejemplo, derechos relacionados a la publicidad, privacidad o derechos morales pueden limitar la forma en la que puedes usar el material.

B

Código de conducta

Con el objetivo de fomentar un ambiente abierto y amigable, las personas encargadas del proyecto, colaboradoras/es y personas de soporte , nos comprometemos a hacer de la participación en nuestro proyecto y en nuestra comunidad una experiencia libre de acoso para todas las personas, independientemente de edad, tamaño corporal, discapacidad, etnia, identidad y expresión de género, nivel de experiencia, educación, nivel socioeconómico, nacionalidad, apariencia personal, raza, religión o identidad y orientación sexual.

Nuestros estándares

Ejemplos de comportamiento que contribuye a crear un ambiente positivo para nuestra comunidad:

- utilizar un lenguaje amigable e inclusivo,
- respetar diferentes puntos de vista y experiencias,
- aceptar adecuadamente la crítica constructiva,
- enfocarse en lo que es mejor para la comunidad y
- mostrar empatía hacia otros miembros de la comunidad.

Ejemplos de comportamiento inaceptable:

- el uso de lenguaje o imágenes sexualizadas así como dar atención o generar avances sexuales no deseados,
- ofender o provocar de modo malintencionado (*trolling*), comentarios despectivos, insultantes y ataques personales o políticos,
- acoso público o privado,
- publicar información privada de otras personas, tales como direcciones físicas o de correo electrónico, sin su permiso explícito, y
- otras conductas que puedan ser razonablemente consideradas como inapropiadas en un entorno profesional

Nuestras responsabilidades

Las personas encargadas del proyecto somos responsables de aclarar los estándares de comportamiento aceptable y se espera que tomem medidas de acción correctivas apropiadas y justas en respuesta a cualquier caso de comportamiento inaceptable.

Las personas encargadas del proyecto tienen el derecho y la responsabilidad de eliminar, editar o rechazar comentarios, *commits*, código, ediciones en la wiki, *issues* y otras contribuciones que no estén alineadas con este Código de Conducta. También pueden prohibir la participación temporal o permanente de cualquier persona por comportamientos que sean considerados inapropiados, amenazantes, ofensivos o dañinos.

Alcance

Este Código de Conducta aplica tanto en espacios dentro del proyecto como en espacios públicos, mientras una persona represente al proyecto o a la comunidad. Ejemplos de representación del proyecto o la comunidad incluyen el uso de una dirección de correo electrónico oficial del proyecto, realizar publicaciones a través de una cuenta oficial de redes sociales o actuar como representante designada/o en cualquier evento presencial o en línea. La representación del proyecto puede ser aclarada y definida en más detalles por las personas encargadas.

Aplicación

Los casos de comportamiento abusivo, acosador o inaceptable pueden ser denunciados enviando un correo electrónico a la persona encargada del proyecto a la dirección `gvwilson@third-bit.com`. Todas las quejas serán revisadas e investigadas y darán como resultado una respuesta que se considere necesaria y apropiada a las circunstancias. El equipo encargado del proyecto está obligado a mantener la confidencialidad de quien reporte un incidente. Se pueden publicar por separado más detalles de políticas de aplicación específicas.

Aquellas personas encargadas del proyecto que no cumplan o apliquen este código de conducta de buena fé pueden enfrentar repercusiones temporales o permanentes determinadas por el resto del equipo encargado del proyecto.

Atribución

Este código de conducta es una adaptación del Contributor Covenant[1] version 1.4

[1] https://www.contributor-covenant.org

C

Unirse a nuestra comunidad

Esperamos que elijas ayudarnos a hacer lo mismo para este libro. Si esta forma de trabajo es nueva para ti, consulta Appendix B nuestro código de conducta, y luego:

Empieza pequeño. Arregla un error tipográfico, aclara la redacción de un ejercicio, corrige o actualiza una cita, o sugiere un mejor ejemplo o analogía para ilustrar algún punto.

Únete a la conversación. Mira los *issues* y los cambios propuestos por otras personas y añádeles tus comentarios. A menudo es posible mejorar las mejoras, y es una buena manera de presentarte a la comunidad y hacer nuevas amistades.

Discute, luego edita. Si quieres proponer un gran cambio, como reorganizar o dividir un capítulo completo, por favor, completa un *issue* que describa tu propuesta y tu razonamiento y etiquétalo como *"Proposal"* (propuesta en inglés). Te alentamos a que agregues comentarios a estos *issues* para que toda la discusión sobre *qué* y *por qué* esté abierta y se pueda archivar. Si se acepta la propuesta, el trabajo real puede dividirse en varios problemas o cambios más pequeños que se pueden abordar de forma independiente.

C.1. Usando este material

Como se declaró en Chapter 1, todo este material puede distribuirse y reutilizarse libremente bajo la licencia Creative Commons Atribución – No Comercial 4.0 (Appendix A). Puedes usar la versión en línea en http://teachtogether.tech/ en cualquier clase (gratuita o de pago), y puedes citar extractos breves bajo las disposiciones de uso justo[1], pero no puedes volver a publicar fragmentos grandes en obras comerciales sin permiso previo.

Este material ha sido usado de muchas maneras, desde una clase en línea de varias semanas hasta un taller intensivo en persona. Por lo general, es posible cubrir grandes partes de los capítulos Chapter 2 a Chapter 6, Chapter 8, y Chapter 10 en dos días de jornada completa.

[1] https://es.wikipedia.org/wiki/Uso_justo

En persona

Esta es la forma más efectiva de impartir esta capacitación, pero también la más exigente. Las personas que participan están físicamente en el mismo lugar. Cuando necesitan practicar cómo enseñar en pequeños grupos, parte de la clase o toda la clase va a espacios de descanso cercanos. Cada participante usa su propia tableta o computadora portátil para ver material en línea durante la clase y para tomar notas compartidas (Section 9.7), y usa lápiz y papel o pizarras para otros ejercicios. Las preguntas y la discusión se hacen en voz alta.

Si estás enseñando en este formato, debes usar notas adhesivas como indicadores de estado para que puedas ver quién necesita ayuda, quién tiene preguntas y quién está listo/a para seguir adelante (Section 9.8). También debes usarlos para distribuir la atención, para que todos obtengan tu atención y tiempo de forma justa, como tarjetas de minutos para alentar a tus estudiantes a reflexionar sobre lo que acaban de aprender y para darte retroalimentación procesable mientras todavía tienes tiempo para actuar en consecuencia.

En línea en grupos

En este formato, 10 a 40 estudiantes se juntan en 2 a 6 grupos de 4 a 12 personas, pero esos grupos están distribuidos geográficamente. Cada grupo usa una cámara y un micrófono para conectarse a la videollamada, en lugar de que cada persona esté en la llamada por separado. Un buen audio es más importante que un buen video en ambas direcciones: una voz sin imágenes (como la radio) es mucho más fácil de entender que las imágenes sin narrativa, y los/as instructores/as no necesitan poder ver personas para responder preguntas, siempre y cuando esas preguntas se puedan escuchar con claridad. Dicho esto, si una lección no es accesible, entonces no es útil (Section 10.3): proporcionar texto descriptivo es una ayuda cuando la calidad del audio es deficiente, e incluso si el audio es bueno resulta importante para aquellas personas con dificultades auditivas.

Toda la clase toma notas compartidas, y también usa las notas compartidas para hacer y responder preguntas. Tener varias decenas de personas tratando de hablar en una llamada no funciona bien, así que en la mayoría de las sesiones el/la profesor/a habla y sus estudiantes responden a través del chat de la herramienta para tomar notas.

En línea de forma individual

La extensión natural de estar en línea en grupos es estar en línea en forma individual. Al igual que con los grupos en línea, el/la docente hablará la mayoría de las veces y los/las estudiantes participarán principalmente a través del chat de texto. También en este caso, un buen audio es más importante que un buen video, y quienes participan deberían usar el chat de texto para indicar que quieren hablar(Appendix E).

Tener participantes en línea individualmente hace que sea más difícil dibujar y compartir mapas conceptuales (Section 3.4) o dar retroalimentación sobre la ense-

ñanza (Section 8.5). Por lo tanto, quienes enseñen deberán confiar más en el uso de ejercicios con resultados escritos que se puedan poner en las notas compartidas, como por ejemplo dar una devolución sobre videos de personas enseñando.

Multi-semana en línea

La clase se reúne todas las semanas durante una hora a través de videoconferencia. Cada reunión puede realizarse dos veces para acomodar las zonas horarias y los horarios de los/las estudiantes. Los/las participantes toman notas compartidas como se describió anteriormente para las clases grupales en línea, para publicar tareas en línea entre clases, y para comentar sobre el trabajo de los demás. En la práctica, los comentarios son relativamente raros: la gente prefiere discutir el material en las reuniones semanales.

Este fue el primer formato utilizado, y ya no lo recomiendo: mientras que extender la clase les da a las personas tiempo para reflexionar y abordar ejercicios más extensos, también aumenta en gran medida las probabilidades de que tengan que abandonar debido a otras demandas de su tiempo.

C.2. Contribuyendo y manteniendo

Contribuciones de todo tipo son bienvenidas, desde sugerencias para mejoras hasta erratas y nuevo material. Todas las personas que contribuyan deben cumplir con nuestro Código de Conducta (Appendix B); al enviar tu trabajo, aceptas que pueda incorporarse tanto en forma original como editada y que pueda ser publicado bajo la misma licencia que el resto de este material (Appendix A).

Si tu material es incorporado, te agregaremos a los agradecimientos (Section 1.3) a menos que solicites lo contrario.

La fuente de la versión original de este libro se almacena en GitHub en:

https://github.com/gvwilson/teachtogether.tech/

Si sabes cómo usar *Git* y GitHub y deseas cambiar, arreglar o agregar algo, por favor envía un ***pull request*** que modifique la fuente del LaTeX. Si deseas obtener una vista previa de tus cambios, por favor ejecuta `make pdf` o `make html` en la línea de comandos.

Si quieres reportar un error, hacer una pregunta, o hacer una sugerencia, presenta un *issue* en el repositorio. Necesitas tener una cuenta de GitHub para hacer esto, pero no necesitas saber cómo usar *Git*.

Si no deseas crear una cuenta de GitHub, envía tu contribución por correo electrónico a `gvwilson@third-bit.com` con "T3" o "Teaching Tech Together" en algún lugar del asunto. Intentaremos responder en una semana.

Finalmente, siempre nos gusta escuchar cómo se ha usado este material, y estamos *siempre* agradecidos/as por el aporte de más diagramas.

D

Glosario

Agotamiento del ego El deterioro del autocontrol debido al uso prolongado o intensivo. Trabajos recientes no pudieron corroborar su existencia.

Amenaza del estereotipo Una situación en la que las personas sienten que corren el riesgo de ser sometidas a los estereotipos de su grupo social.

Andamiaje Se proporciona material adicional a las/los estudiantes en etapa inicial para ayudarlas/os a resolver problemas.

Aprendizaje activo Un enfoque de la enseñanza en el que las/los estudiantes se involucran con el material a través de la discusión, resolución de problemas, estudios de casos, y otras actividades que requieren que reflexionen y usen nueva información en tiempo real. Ver también **aprendizaje pasivo**.

Aprendizaje basado en la indagación La práctica de permitir que las/os estudiantes hagan sus propias preguntas, establezcan sus propios objetivos y encuentren su propio camino a través de un tema.

Aprendizaje cognitivo Una teoría de aprendizaje que enfatiza el proceso del/de la docente que transmite habilidades e ideas situacionalmente a su estudiante.

Aprendizaje pasivo Un enfoque de la enseñanza en el que las/os estudiantes leen, escuchan o miran sin utilizar inmediatamente nuevos conocimientos. El aprendizaje pasivo es menos efectivo que el **aprendizaje activo**.

Aprendizaje personalizado Adaptación automática de lecciones para satisfacer las necesidades de las/los estudiantes individuales.

Aprendizaje situado Un modelo de aprendizaje que se centra en la transición de las personas de ser recién llegadas a ser miembros aceptados de una **comunidad de práctica**.

Artefacto tangible Algo en lo que una/un estudiante puede trabajar y cuyo estado le proporciona retroalimentación sobre su progreso y ayuda a la/el estudiante a diagnosticar errores.

Aula invertida Una clase en la que las/los alumnos ven lecciones grabadas en su propio tiempo, mientras que el tiempo de clase se utiliza para resolver conjuntos de problemas y responder preguntas.

Automaticidad La capacidad de hacer una tarea sin concentrarse en sus detalles de bajo nivel.

Carga cognitiva El esfuerzo mental necesario para resolver un problema. La teoría de la carga cognitiva lo divide en **carga intrínseca**, **carga pertinente** y **carga extrínseca**. Sostiene que las personas aprenden más rápido cuando las cargas pertinente y extrínseca son reducidas

Carga extrínseca Cualquier **carga cognitiva** que distrae del aprendizaje.

Carga instrínseca La **carga cognitiva** requerida para absorber nueva información.

Carga pertinente La **carga cognitiva** requerida para vincular la nueva información con la antigua.

Ciencia de implementación El estudio de cómo traducir los hallazgos de la investigación a la práctica clínica diaria.

Ciencias de la computación acústico Un estilo de enseñanza que introduce conceptos informáticos utilizando ejemplos y artefactos que no son de programación.

Co-enseñanza Enseñar con otra/otro docente en el salón de clases.

Cognición externalizada El uso de ayuda gráfica, física o verbal para aumentar el pensamiento.

Cognitivismo Una teoría del aprendizaje que sostiene que los estados y procesos mentales pueden y deben incluirse en modelos de aprendizaje. Ver también **conductismo**.

Los Comunes algo gestionado conjuntamente por una comunidad de acuerdo con las reglas que la misma comunidad ha desarrollado y adoptado.

Comunidad de práctica Un grupo de personas que se perpetúan a sí mismas y comparten y desarrollan un oficio como tejedoras/es, músicas/os o programadoras/es. Ver también **participación inicial legítima**.

Conductismo Una teoría del aprendizaje cuyo principio central es estímulo y respuesta, y cuyo objetivo es explicar el comportamiento sin recurrir a estados mentales internos u otros inobservables. Ver además **cognitivismo**.

Conectivismo Una teoría del aprendizaje que sostiene que el conocimiento se distribuye, que el aprendizaje es el proceso de navegación, crecimiento y poda de conexiones, y que enfatiza los aspectos sociales del aprendizaje hechos posibles por Internet.

Conocimiento de contenido pedagógico (*PCK*, por sus siglas en Inglés) La comprensión de cómo enseñar un tema en particular, es decir, el mejor orden en el cual introducir temas y qué ejemplos usar. Ver también **conocimiento del contenido** y **conocimiento pedagógico general**.

Conocimiento del contenido La comprensión de una persona de un tema. Ver también **conocimiento pedagógico general** y **conocimiento de contenido pedagógico**.

Conocimiento pedagógico general La comprensión de una persona de los principios generales de la enseñanza. Ver también **conocimiento del contenido** y **conocimiento de contenido pedagógico**.

Constructivismo Una teoría del aprendizaje que considera que las/los estudiantes construyen activamente el conocimiento.

Contribuyendo a la pedagogía estudiantil Tener a las/los estudiantes produciendo artefactos para contribuir al aprendizaje de otros.

CS0. Introducción a las ciencias de la computación Un curso introductorio de nivel universitario sobre computación dirigido a estudiantes no avanzadas/os con poca o ninguna experiencia previa en programación.

CS1. ciencias de la computación I Un curso introductorio de ciencias de la computación a nivel universitario, generalmente de un semestre, que se enfoca en variables, bucles, funciones y otras mecánicas básicas.

CS2. ciencias de la computación II Un segundo curso de ciencias de la computación de nivel universitario que generalmente presenta estructuras de datos básicas como pilas, colas y diccionarios.

Cursos on-line masivos y abiertos (*MOOC*, por sus siglas en Inglés) Un curso en línea diseñado para la inscripción masiva y el estudio asíncrono, que generalmente usa videos grabados y calificaciones automáticas.

Desarrollo basado en test Una práctica de desarrollo de software en la que las/los programadoras/es escriben primero los tests para darse objetivos concretos y aclarar su comprensión de cómo se ve "terminado".

Directorio o junta de servicio Una junta cuyos miembros asumen roles de trabajo en la organización.

Directorio. Junta Una junta cuya responsabilidad principal es contratar, supervisar y, si es necesario, despedir al director.

Discurso breve de presentación Una breve descripción de una idea, proyecto, producto o persona que se puede dar y comprender en solo unos segundos.

Diseño instruccional El arte de crear y evaluar lecciones específicas para audiencias específicas. Ver también **psicología educacional**.

Distractor pausible Una respuesta incorrecta, que parece que podría ser correcta, en una pregunta de opción múltiple. Ver también **poder de diagnóstico**.

Efecto de atención dividida La disminución que ocurre en el aprendizaje cuando las/os estudiantes deben dividir su atención entre múltiples presentaciones concurrentes de la misma información (por ejemplo, subtítulos y una voz en off).

Efecto de hipercorrección Cuanto más cree alguien que su respuesta en un examen era correcta, más probabilidades hay de que no repita el error una vez que descubre que, de hecho, estaba equivocada/do..

Efecto Dunning-Kruger La tendencia de las personas que solo saben un poco sobre un tema, a estimar incorrectamente su comprensión del mismo.

Efecto inverso de la experiencia La forma en que la instrucción que es efectiva para los novatos se vuelve ineficaz para los profesionales competentes o expertos.

Ejemplos desvanecidos Una serie de ejemplos en los que se borra un número cada vez mayor de pasos clave. Ver también **andamiaje**.

Enseñando para el examen Cualquier método de "educación" que se centre en preparar a las/los estudiantes para aprobar los exámenes estandarizados, en lugar de aprender realmente.

Enseñanza activa Un enfoque de la instrucción en el que la/el docente actúa sobre la nueva información adquirida de los alumnos mientras enseña (ej. cambiando dinámicamente un ejemplo o reorganizando el orden previsto del contenido). Ver también **enseñanza pasiva**.

Enseñanza pasiva Un enfoque a la enseñanza en el que la/el docente no ajusta el ritmo o los ejemplos, o no actúa de acuerdo con los comentarios de las/los estudiantes, durante la lección. Ver también **enseñanza activa**.

Estudiante free-range. Estudiante de rango libre. Alguien que aprende fuera de un aula institucional con un plan de estudios y tareas obligatorias. (Quienes usan este término, ocasionalmente se refieren a los estudiantes en las aulas como "estudiantes battery-farmed", pero nosotros no lo hacemos porque sería grosero).

Estudiante tipo Una breve descripción de una/un estudiante objetivo tipo para una lección que incluye: sus antecedentes generales, lo que ya sabe, lo que quiere hacer, cómo la lección le ayudará y cualquier necesidad especial que puedan tener.

Etiquetado de submetas Dar nombres a cada paso en una descripción paso a paso de un proceso de resolución de problemas.

Evaluación formativa Evaluación que se lleva a cabo durante una clase para dar retroalimentación tanto a la/el estudiante como a la/el docente sobre la comprensión real. Ver también **evaluación sumativa**.

Evaluación sumativa Evaluación que se realiza al final de una lección para determinar si se ha realizado el aprendizaje deseado.

Experta/o Alguien que puede diagnosticar y manejar situaciones inusuales, sabe cuándo no se aplican las reglas habituales y tiende a reconocer soluciones en lugar de razonarlas. Ver también **practicante competente** y **novata/o**.

Falla productiva Una situación en la que a las/los estudiantes se les dan deliberadamente, problemas que no se pueden resolver con el conocimiento que tienen

y deben salir y adquirir nueva información para progresar. Ver también **Zona de Desarrollo Proximal**.

Falso principiante Alguien que ha estudiado un idioma antes pero lo está aprendiendo nuevamente. Los falsos principiantes comienzan en el mismo punto que los principiantes verdaderos (es decir, en una evaluación inicial mostrarán el mismo nivel de competencia) pero pueden avanzar mucho más rápidamente.

Flujo La sensación de estar completamente inmerso en una actividad, frecuentemente asociada con una alta productividad.

Fuzz testing Una técnica de prueba de software basada en generar y enviar datos aleatorios.

Hashing Generar una clave digital pseudoaleatoria condensada a partir de datos; cualquier entrada específica produce la misma salida, pero es muy probable que diferentes entradas produzcan diferentes salidas.

Impotencia aprendida Una situación en la que las personas que son sometidas repetidamente a comentarios negativos de los cuales no tienen forma de escapar aprenden a ni siquiera intentar escapar cuando pueden.

Inclusión Trabajar activamente para incluir personas con diversos antecedentes y necesidades.

Instrucción directa Un método de enseñanza centrado en un diseño curricular meticuloso dictado a través de guiones pre-escritos.

Instrucción por pares Un método de enseñanza en el que la/el docente hace una pregunta y luego las/los estudiantes se comprometen con una primera respuesta, discuten las respuestas con sus compañeras/os y se comprometen con una respuesta revisada.

Integración Computacional Usar la informática para volver a implementar artefactos culturales preexistentes, por ejemplo, crear variantes de diseños tradicionales usando herramientas de dibujo por computadora.

Intuición La capacidad de comprender algo de inmediato, sin necesidad aparente de razonamiento consciente.

Inventario de conceptos Una prueba diseñada para determinar qué tan bien una/un alumna/o comprende un dominio. A diferencia de la mayoría de las pruebas realizadas por instructores, los inventarios de conceptos se basan en una extensa investigación y validación.

Jugyokenkyu Literalmente "estudio de lección", un conjunto de prácticas que incluye hacer que las/os docentes se observen rutinariamente entre sí y discutan las lecciones que imparten para compartir conocimientos y mejorar habilidades.

Lección de demostración Una lección dictada por una/un docente a estudiantes

reales mientras otras/os docentes observan para aprender nuevas técnicas de enseñanza.

Leer-cubrir-recuperar Una práctica de estudio en la que la/el estudiante cubre hechos o términos clave durante un primer paso por el material y luego verifica cuánto recuerda en un segundo paso.

Manual mínimo Un enfoque de capacitación que divide cada tarea en instrucciones de una sola página que también explican cómo diagnosticar y corregir errores comunes.

Manual Material de referencia, destinado a ayudar a alguien que ya comprende un tema, a completar (o recordar) detalles.

Mapa conceptual Una imagen de un modelo mental en el que los conceptos son nodos en un gráfico y las relaciones entre esos conceptos son arcos (etiquetados).

Máquina nocional Un modelo general simplificado de cómo se ejecuta una familia particular de programas.

Marca Las asociaciones que las personas tienen con el nombre de un producto o identidad.

Marketing El arte de ver las cosas desde la perspectiva de otras personas, comprender sus deseos y necesidades y encontrar formas de satisfacerlas.

Memoria de corto plazo La parte de la memoria que almacena brevemente información a la que puede acceder directamente la conciencia.

Memoria de largo plazo La parte de la memoria que almacena información durante largos períodos de tiempo. La memoria a largo plazo es muy grande, pero lenta. Ver también **memoria de corto plazo**.

Memoria de trabajo Ver**memoria de corto plazo**.

Memoria persistente Ver **memoria de largo plazo**.

Mentalidad de crecimiento La creencia de que la habilidad viene con la práctica. Ver también **mentalidad fija**.

Mentalidad fija La creencia de que una habilidad es innata y que el fracaso se debe a la falta de algún atributo necesario. Ver también **mentalidad de crecimiento**.

Metacognición Pensar acerca de pensar.

Modelo deficitario La idea de que algunos grupos están subrepresentados en informática (o algún otro campo) porque sus miembros carecen de algún atributo o calidad.

Modelo mental Una representación simplificada de los elementos clave y las relaciones de algunos dominios de problemas que es lo suficientemente buena como para apoyar la resolución de problemas.

Motivación extrínseca Ser impulsada/o por recompensas externas como el pago o el miedo al castigo. Ver también **motivación intrínseca**.

Motivación intrínseca Ser impulsada/o por el disfrute o la satisfacción de hacer una tarea como fin en sí mismo. Ver también **motivación extrínseca**.

Motor de optimización de posicionamiento en buscadores (*SEO*, por sus siglas en Inglés) Aumentar la cantidad y la calidad del tráfico del sitio web al hacer que las páginas sean más fáciles de encontrar o parezcan más importantes para los motores de búsqueda.

Notas guiadas Notas preparadas por la/el docente que indican a las/os estudiantes que respondan a la información clave en una conferencia o discusión.

Novata/o. Persona novata. Principiante Alguien que aún no ha construido un modelo mental utilizable de un dominio. Ver también **Practicantes competentes** y **experta/o**.

Objetivo de aprendizaje Qué está intentando lograr enseñar la lección.

Paradoja de la reusabilidad Sostiene que cuanto más reutilizable es una lección, es menos efectiva pedagógicamente.

Fragmentación El acto de agrupar conceptos relacionados juntos que pueden almacenarse y procesarse como una sola unidad.

Participación inicial legítima La participación de las/os recién llegadas/os en tareas simples y de bajo riesgo que una **comunidad de práctica** reconoce como contribuciones válidas.

Pensamiento computacional Pensar la resolución de problemas en formas inspiradas en la programación (aunque el término se usa de muchas otras maneras).

Piensa-trabaja en pareja-comparte Un método de colaboración en el que cada persona piensa individualmente sobre una pregunta o problema, luego se junta con otra/o compañera/o para compartir ideas, y luego una persona de cada pareja presenta para todo el grupo.

Poder de diagnóstico El grado en que una respuesta incorrecta a una pregunta o ejercicio le dice a la/el docente qué conceptos erróneos tiene una/un estudiante en particular.

Posicionamiento Lo que diferencia a una marca de otras marcas similares.

Práctica deliberada El acto de observar el desempeño de una tarea mientras se realiza para mejorar la capacidad.

Práctica reflexiva Ver **práctica deliberada**.

Practicante competente Alguien que puede realizar tareas normales con un esfuerzo normal en circunstancias normales. Ver también **principiante** and **experta/o**.

Primero los objetos Un enfoque para enseñar programación donde objetos y clases se introducen temprano.

Principiante absoluto Alguien que nunca se ha encontrado con los conceptos o material antes. El término se usa en distinción para **falso principiante**.

Privilegio preparatorio La ventaja de provenir de un entorno que proporciona más preparación para una tarea de aprendizaje en particular que otras.

Problemas de Parson Una técnica de evaluación desarrollada por Dale Parsons y otros en la que los alumnos reorganizan el material dado para construir una respuesta correcta a una pregunta. [Pars2006].

Programación en pareja Una práctica de desarrollo de software en la que dos programadores comparten una computadora. Un programador (el piloto) escribe, mientras que el otro (el navegante) ofrece comentarios y sugerencias en tiempo real. La programación en pareja a menudo se usa como práctica docente en las clases de programación.

Programación en vivo El acto de enseñar programación escribiendo software frente a los alumnos a medida que avanza la lección.

Programadora conversacional Alguien que necesita saber lo suficiente sobre computación para tener una conversación significativa con un programador, pero no que va a programar por sí mismo.

Psicología Educacional El estudio de cómo la gente aprende. Ver también **diseño instruccional**.

Pull request Un conjunto de cambios propuestos a un repositorio de GitHub que pueden revisarse, actualizarse y, finalmente, agregarse al repositorio.

Punto ciego del experto La incapacidad de las personas expertas para empatizar con las personas novatas que se encuentran por primera vez con conceptos o prácticas.

Reingeniería Un método de diseño instruccional que trabaja hacia atrás desde una evaluación sumativa hasta evaluaciones formativas y desde allí al contenido de la lección.

Representación de la comunidad Usar el capital cultural para resaltar las identidades sociales, las historias y las redes comunitarias de las/los estudiantes en las actividades de aprendizaje.

Representación fluida La capacidad de moverse rápidamente entre diferentes modelos de un problema.

Resultado de aprendizaje Qué es lo que la lección realmente logra enseñar.

Revisión por pares calibrada Hacer que alumnas y alumnos comparen sus revisiones del trabajo de ejemplo con las de un maestro o una maestra antes de que se les permita revisar el trabajo de sus pares.

Síndrome del impostor Un sentimiento de inseguridad sobre los logros propios, que se manifiesta como un miedo a ser expuesta/o como un fraude.

Sistema de gestión de aprendizaje (*LMS*, por sus siglas en Inglés): Una aplicación para registrar la inscripción a cursos, presentaciones de ejercicios, calificaciones y otros aspectos burocráticos del aprendizaje formal en el aula.

Tarea auténtica Una tarea que contiene elementos importantes de cosas que los alumnos harían en situaciones reales (fuera del aula). Para ser auténtica, una tarea debe requerir que los alumnos construyan sus propias respuestas en lugar de elegir entre las respuestas proporcionadas, y trabajar con las mismas herramientas y datos que usarían en la vida real.

Tarjetas de minutos Una técnica de retroalimentación en la que las/los estudiantes pasan un minuto escribiendo una cosa positiva sobre una lección (por ejemplo, una cosa que han aprendido) y una cosa negativa (por ejemplo, una pregunta que aún no ha sido respondida).

Taxonomía de Bloom Una clasificación jerárquica, ampliamente adoptada, de seis etapas de comprensión y cuyos niveles son *conocimiento*, *comprensión*, *aplicación*, *análisis*, *síntesis* y *evaluación*. Ver también **Taxonomía de Fink**.

Taxonomía de Fink Una clasificación de comprensión no jerárquica de seis partes, propuesta por primera vez en [Fink2013] cuyas categorías son *conocimiento fundamental*, *aplicación*, *integración*, *dimensión humana*, *cuidado* y *aprender a aprender*. Ver también **Taxonomía de Bloom**.

Transferencia apropiada de procesamiento La mejora en recordar qué ocurre cuando la práctica utiliza actividades similares a las utilizadas en los tests.

Transferencia cercana **Transferencia de aprendizaje** entre dominios estrechamente relacionados, por ejemplo, mejora en la comprensión de decimales como resultado de hacer ejercicios con fracciones.

Transferencia de aprendizaje Aplicar el conocimiento aprendido en un contexto a problemas en otro contexto. Ver también **transferencia cercana** y **transferencia lejana**.

Transferencia lejana La **transferencia de aprendizaje** entre dominios ampliamente separados, por ejemplo, mejora en las habilidades matemáticas como resultado de jugar al ajedrez.

Tutorial Una lección destinada a ayudar a alguien a mejorar su comprensión general de un tema.

Twitch coding Hacer que un grupo de personas decida momento a momento o línea por línea qué agregarle a un programa a continuación.

Usuario final docente Por analogía con **usuario final programador**, alguien que enseña con frecuencia, pero cuya ocupación principal no es la enseñanza, que

tiene poca o ninguna experiencia en pedagogía y que puede trabajar fuera de las aulas institucionales.

Usuario final programador Alguien que no se considera un programador, pero que, sin embargo, escribe y depura software, como por ejemplo, un artista que crea macros complejas para una herramienta de dibujo.

Zona de Desarrollo Proximal (*ZPD*, por sus siglas en Inglés) Incluye el conjunto de problemas que las personas aún no pueden resolver por sí mismas pero que pueden resolver con la ayuda de un mentor más experimentado. Ver también **falla productiva**.

E

Reuniones, reuniones, reuniones

La mayoría de la gente es muy mala al organizar reuniones: no llevan una agenda, no se toman unos minutos, hablan vagamente o se desvían en irrelevancias, dicen algo trivial o repiten lo que otros han dicho sólo para decir algo, y mantienen conversaciones paralelas (lo cual garantiza que la reunión será una pérdida de tiempo). Saber cómo organizar una reunión de manera eficiente es una habilidad central para cualquiera que desee terminar bien el trabajo; saber cómo participar en la reunión de otra persona es igual de importante (y aunque recibe mucha menos atención, como dijo una colega una vez: todos ofrecen entrenamiento de líderes pero nadie ofrece entrenamiento de seguidores).

Las reglas más importantes para hacer que las reuniones sean eficientes no son secretas, pero rara vez se siguen:

Decide si realmente se necesita una reunión. Si el único propósito es compartir información, envía un breve correo electrónico en su lugar. Recuerda, puedes leer más rápido que cualquiera pueda hablar: si alguien tiene datos para que el resto del equipo los asimile, la forma más educada de comunicarlos es escribirlos.

Escribe una agenda. Si a nadie le importa lo suficiente la reunión como para escribir una lista de puntos de lo que se discutirá, la reunión probablemente no se necesita.

Incluye horarios en la agenda. Las agendas también pueden ayudarte para evitar que los primeros puntos le roben tiempo a los últimos si incluyes el tiempo que le dedicarás a cada punto en la agenda. Tus primeras estimaciones con cualquier grupo nuevo serán tremendamente optimistas, así que revísalas nuevamente para las siguientes reuniones. Sin embargo, no deberías planear una segunda o tercera reunión porque no alcanzó el tiempo: en cambio, trata de averiguar por qué ocupaste tiempo extra y arregla el problema que lo originó.

Prioriza. Cada reunión es un microproyecto, por lo tanto el trabajo debería priorizarse de la misma manera que se hace para otros proyectos: aquello que tendrá alto impacto pero lleva poco tiempo debería realizarse primero, y aquello que tomará mucho tiempo pero tiene bajo impacto debería omitirse.

Haz a una persona responsable de mantener las cosas en movimiento. Una persona debería tener la tarea de mantener los puntos a tiempo, llamando la atención a la gente que esté revisando correo electrónico o teniendo conversaciones paralelas, pidiendo a aquellos que están hablando mucho que lleguen al punto, e invitando a

gente que no interviene que exprese su opinión. Esta persona *no* debería hacer toda la charla; en realidad, en una reunión bien armada aquel que esté a cargo hablará menos que los otros participantes.

Pide amabilidad. Que nadie llegue a ser grosero, que nadie empiece a divagar, y si alguien se sale del tema es tanto el derecho como la responsabilidad del moderador decir "Discutamos eso en otro lado".

Sin interrupciones. Los participantes deben levantar la mano o poner una nota adhesiva si quieren hablar después. Si la persona que está hablando no los nota, quien modera la reunión debería hacerlo.

Sin tecnología A menos que sea necesario por razones de accesibilidad insistir amablemente que todos guarden sus teléfonos, tabletas y computadoras. (p.ej. Por favor, cierren sus aparatos electrónicos).

Registro de minutas. Alguna otra persona que no sea quien modere debería tomar notas de forma puntual sobre los fragmentos más importantes de información compartida, todas las decisiones tomadas y todas las tareas que se asignaron a alguien.

Toma notas. Mientras otras personas están hablando, los participantes deberían tomar notas de preguntas que quieran hacer o de observaciones que quieran realizar (te sorprenderás qué inteligente parecerás cuando llegue tu turno para hablar).

Termina temprano. Si tu reunión está programada de 10:00 a 11:00, debes intentar terminar a las 10:50 para dar tiempo a la gente de pasar por el baño en su camino a donde vayan luego.

Tan pronto termina la reunión, envía a todos un correo electrónico con la minuta o publícala en la web:

La gente que no estuvo en la reunión puede mantenerse al tanto de lo que ocurrió. Una página web o un mensaje de correo electrónico es una forma mucho más eficiente de ponerse al día que preguntarle a un compañero de equipo qué te perdiste.

Cualquiera puede comprobar lo que realmente se dijo o prometió. Más de una vez he revisado la minuta de una reunión en la que estuve y pensé: "Yo dije eso?" o "Espera un minuto, yo no prometí tenerlo listo para entonces!" Accidentalmente o no, muchas veces la gente recordará las cosas de manera diferente; escribirlo da la oportunidad a los miembros del equipo de corregir errores, lo que puede ahorrar muchos malos entendidos más tarde.

Las personas pueden ser responsables en reuniones posteriores. No tiene sentido hacer listas de preguntas y puntos de acción si después no los sigues. Si estás utilizando algún tipo de sistema de seguimiento de temas, crea un tema por cada pregunta o tarea justo después de la reunión y actualiza los que se cumplieron, luego comienza cada reunión pasando por una lista de esos temas.

[Brow2007; Broo2016; Roge2018] tienen muchos consejos para organizar

reuniones. Según mi experiencia, una hora de entrenamiento en cómo ser moderador es una de las mejores inversiones que harás.

Notas Adhesivas y Bingo para Interrupción

Algunas personas están tan acostumbradas al sonido de su propia voz que insistirán en hablar la mitad del tiempo sin importar cuántas personas haya en la habitación. Para evitar esto entrega a todos tres notas adhesivas al comienzo de la reunión. Cada vez que hablen tienen que sacar una nota adhesiva. Cuando se queden sin notas no se les permitirá hablar hasta que todos hayan usado al menos una. En ese momento todos recuperan sus tres notas adhesivas. Esto asegura que nadie hable más de tres veces que la persona más callada de la reunión, y cambia completamente la dinámica de la mayoría de los grupos: personas que dejan de intentar ser escuchadas porque siempre son tapadas de repente tienen espacio para contribuir, y aquellas que hablaban con demasiada frecuencia se dan cuenta lo injustos que han sido[1].

Otra técnica es un bingo de interrupción. Dibuja una tabla y etiqueta las filas y columnas con los nombres de los participantes. Agrega en la celda apropiada una marca para contar cada vez que alguien interrumpa a otro, y toma un momento para compartir los resultados a la mitad de la reunión. En la mayoría de los casos verás que una o dos personas son las que interrumpen siempre, a menudo sin ser conscientes de ello. Eso solo muchas veces es suficiente para detenerlas. Nota que esta técnica está destinada a manejar las interrupciones, no el tiempo de conversación: puede ser apropiado que las personas con más conocimiento de un tema hablen sobre él con más frecuencia en una reunión, pero nunca es apropiado cortar repetidamente a las personas.

E.1. Las reglas de Martha

Las organizaciones de todo el mundo realizan sus reuniones de acuerdo a Reglas de Orden de Roberto[2], pero son mucho más formales que lo requerido para proyectos pequeños. Una ligera alternativa conocida como "Las reglas de Martha" puede que sea mucho mejor para la toma de decisiones por consenso [Mina1986]:

1. Antes de cada reunión cualquiera que lo desee puede patrocinar una propuesta compartiéndola con el grupo. Las propuestas deben ser archivadas al menos 24 horas antes de una reunión para ser consideradas en esa reunión, y deben incluir:
   - un resumen de una línea;
   - el texto completo de la propuesta;

[1] Yo ciertamente lo hice cuando me hicieron esto...
[2] https://en.wikipedia.org/wiki/Robert%27s_Rules_of_Order

- cualquier información de antecedentes requerida;
- pros y contras; y
- posibles alternativas

Las propuestas deberían ser a lo sumo de 2 páginas.

2. Se establece un quórum en una reunión si la mitad o más de los miembros votantes están presentes.
3. Una vez que una persona patrocina una propuesta es responsable de ella. El grupo no puede discutir o votar sobre el tema a menos que quien patrocina o su delegado esté presente. La persona patrocinadora también es responsable de presentar el tema al grupo.
4. Después que la persona patrocinadora presente la propuesta se emite un voto preliminar para la propuesta antes de cualquier discusión:
   - ¿A quién le gusta la propuesta?
   - ¿A quién le parece razonable la propuesta?
   - ¿Quién se siente incómodo con la propuesta?

   Los votos preliminares se pueden hacer con el pulgar hacia arriba, el pulgar hacia los lados o el pulgar hacia abajo (en persona) o escribiendo +1, 0 o -1 en el chat en línea (en reuniones virtuales).
5. Si a todos o a la mayoría del grupo le gusta o resulta razonable la propuesta, se pasa inmediatamente a una votación formal sin más discusión.
6. Si la mayoría del grupo está disconforme con la propuesta se pospone para que la persona patrocinadora pueda volver a trabajar sobre ella.
7. Si algunos miembros se sienten disconformes pueden expresar brevemente sus objeciones. Luego se establece un temporizador para una breve discusión moderada por una persona facilitadora. Después de diez minutos o cuando nadie más tenga algo que agregar (lo que ocurra primero), quien facilita llama a una votación sí-o-no sobre la pregunta: "¿Deberíamos implementar esta decisión aun con las objeciones establecidas?" Si la mayoría vota "sí" la propuesta se implementa. De lo contrario, la propuesta se devuelve a la persona patrocinadora para trabajarla más.

E.2. Reuniones en línea

Discusión de Chelsea Troy[3] de por qué las reuniones en línea son a menudo frustrantes e improductivas resulta un punto importante: en la mayoría de las reuniones

[3] https://chelseatroy.com/2018/03/29/why-do-remote-meetings-suck-so-much/

en línea la primera persona en hablar durante una pausa toma la palabra. ¿El resultado? "Si tienes algo que quieres decir, tienes que dejar de escuchar a la persona que está hablando actualmente y en lugar de eso, enfócate en cuándo van a detenerse o terminar, para que puedas saltar sobre ese nanosegundo de silencio y ser el primero en pronunciar algo. El formato… alienta a los participantes que deseen contribuir a decir más y escuchar menos."

La solución es chatear (charla en texto) a la par de la videoconferencia donde las personas pueden indicar que quieren hablar. Quien modere entonces selecciona personas de la lista de espera. Si la reunión es grande o argumentativa, mantener a todos silenciados y solo permitir a quien modere liberar el micrófono a las personas.

E.3. La autopsia

Cada proyecto debe terminar con una autopsia en la que los participantes reflexionan sobre lo que acaban de lograr y qué podrían mejorar la próxima vez. Su objetivo es *no* señalar con el dedo de la vergüenza a las personas, aunque si eso tiene que suceder, la autopsia es el mejor lugar para ello.

Una autopsia se realiza como cualquier otra reunión con algunas pautas adicionales [Derb2006]:

Conseguir una persona que modere y que no sea parte del proyecto y no tenga interés en serlo.

Reservar una hora y solo una hora. Según mi experiencia, nada útil se dice en los primeros diez minutos de la primera autopsia de alguien, dado que las personas son naturalmente un poco tímidas para alabar o condenar su propio trabajo. Igualmente, no se dice nada útil después de la primera hora: si aún sigues hablando, probablemente sea porque una o dos personas tienen cosas que quieren sacarse del pecho en lugar de dar sugerencias para poder mejorar.

Requerir asistencia. Todos los que formaron parte del proyecto deben estar en la sala para la autopsia. Esto es más importante de lo que piensas: las personas que tienen más que aprender de la autopsia en general son menos propensas a presentarse si la reunión es opcional.

Confeccionar dos listas. Cuando estoy moderando pongo los encabezados "Hazlo otra vez" y "Hazlo diferente" en la pizarra, luego pido a cada persona que me dé una respuesta para cada lista, en orden y sin repetir nada que ya se haya dicho.

Comentar sobre acciones en lugar de individuos. Para cuando el proyecto esté terminado es posible que algunas personas ya no sean amigas. No dejes que esto desvíe la reunión: si alguien tiene una queja específica sobre otro miembro del equipo, pídeles que critiquen un evento o decisión en particular. "Tiene una mala actitud" *no* ayuda a nadie a mejorar.

Priorizar las recomendaciones. Una vez que los pensamientos de todos estén al descubierto ordénalos según cuáles son los más importantes de mantener y cuáles son los más importantes para cambiar. Probablemente solo podrás abordar uno o dos de cada lista en tu próximo proyecto, pero si haces eso cada vez tu vida mejorará rápidamente.

F

Listas de verificación y plantillas

[Gawa2007] hizo popular la idea de que usar listas de verificación puede salvar vidas, y estudios más recientes apoyan su efectividad [Avel2013; Urba2014; Rams2019]. Encontramos útiles las listas de verificación, en particular cuando hay docentes que recién se incorporan al equipo. Los ejemplos a continuación pueden servirte como material inicial a partir del cual desarrollar tus propias listas de verificación.

F.1. Enseñando a evaluar

Esta rúbrica fue diseñada para evaluar lo enseñado durante 5 a 10 minutos con diapositivas, programación en vivo o una combinación de ambas estrategias. Valora cada ítem como "sí," "Más o menos," "No," or "No corresponde (N/A)."

Inicio	Presente (usa N/A para otras respuestas) Adecuada duración (10 a 30 segundos) Se presenta Presenta el tema que se trabajará Describe los requisitos
Contenido	Objetivos claros/narrativa fluida Lenguaje inclusivo Ejemplos y tareas reales Enseña buenas prácticas/utiliza el idioma del código Señala un camino intermedio entre la Escila de la jerga y la Caribdis de la sobresimplificación
Dando la lección	Voz clara y entendible (usa "Más o menos" o "No" para acentos muy marcados) Ritmo: ni muy rápido ni muy lento, no realiza pausas largas o se interrumpe, no aparenta estar leyendo sus notas Seguridad: no se pierde en el pozo de alquitrán de la incertidumbre ni tampoco en las colinas de estiércol de la condescendencia

Diapositivas	Usa diapositivas (completa con N/A el resto de las respuestas si no usa diapositivas) Diapositivas y discurso se complementan uno al otro (programación dual) Fuentes y colores legibles/sin bloques de texto abrumadores por su tamaño Pantalla: cambia frecuentemente (algo cada 30 segundos) Adecuado uso de figuras
Programación en vivo	Usa programación en vivo (completa con N/A el resto de las respuestas si no usa programación en vivo) Código y discurso se complementan uno al otro Fuentes y colores legibles/adecuada cantidad de código en pantalla Uso de herramientas de forma adecuada Resalta elementos clave del código Analiza los errores
Cierre	Presente (valora N/A para otras respuestas) Adecuada duración (10 a 30 segundos) Resume puntos clave Presenta un esquema general de los próximos pasos
En general	Puntos claramente conectados/flujo lógico Hace que el tema sea interesante (i.e. no aburrido) Comprende el tema

F.2. Evaluación del grupo docente

Esta rúbrica fue diseñada para evaluar el desempeño de individuos dentro de un grupo. Los ejemplos a continuación pueden servirte como material inicial a partir del cual desarrollar tus propias rúbricas. Valora cada ítem como “sí,” “Más o menos,” “No,” or “No corresponde (N/A).”

Comunicación	Escucha atentamente y sin interrumpir Aclara lo que se ha dicho para asegurar la comprensión Articula ideas en forma clara y concisa Argumenta adecuadamente sus ideas Obtiene el apoyo de otros miembros del equipo
Toma de decisiones	Analiza los problemas desde diferentes puntos de vista Aplica lógica para resolver problemas

	Propone soluciones basadas en hechos y no en “corazonadas” o intuición Invita a los miembros del equipo a proponer nuevas ideas Genera nuevas ideas Acepta cambios
Colaboración	Reconoce los problemas que el equipo necesita enfrentar y resolver Trabaja para hallar soluciones que sean aceptables para todas las partes involucradas Comparte el crédito del éxito con otros miembros del equipo Promueve la participación entre todos los miembros del equipo Acepta la crítica abiertamente y sin “ponerse a la defensiva” Coopera con el equipo
Autogestión	Monitorea sus avances para asegurar que se alcancen los objetivos Le da máxima prioridad a obtener resultados Define tareas prioritarias para los encuentros de trabajo Promueve que otros miembros del equipo manifiesten sus opiniones, incluso si no coinciden con las propias Mantiene la atención durante la reunión Usa eficientemente el tiempo de reunión Sugiere formas de trabajar en las reuniones

F.3. Organización de eventos

Las listas de verificación a continuación pueden usarse antes, durante y después de un evento.

Programar el evento

- Decidir si será presencial, virtual para un lugar, o virtual para más de un lugar.
- Conversar con la/el disertante? sobre sus expectativas y asegurarse que están de acuerdo en cuanto a quién cubrirá los costos de traslado.
- Definir quiénes podrán participar: ¿será el evento abierto a todas las personas? ¿restringido a miembros de una organización? ¿una situación intermedia?
- Organizar quiénes serán docentes.
- Organizar el espacio, incluyendo *breakout rooms* si fuera necesario.

- Definir la fecha. Si fuera presencia, reservar lo relativo al viaje.
- Conseguir nombres y direcciones de e-mail de participantes a través de la/el disertante.
- Asegurarse que la totalidad de las y los participantes esté registrada.

Construcción del evento

- Crea una página web con los detalles del taller, que incluya fecha, lugar, y lo que las y los participantes deben traer consigo.
- Confirma las necesidades especiales de las/los participantes.
- Si el evento es virtual prueba el modo de videoconferencia, dos veces.
- Asegúrate que las/los participantes tengan acceso a internet.
- Crea un espacio para compartir apuntes y soluciones a los ejercicios (p.ej. un documento Google Doc).
- Establece contacto con las/los asistentes por correo electrónico con un mensaje de bienvenida que contenga el link a la página del taller, lecturas sobre la temática, la descripción de la configuración que deba hacer, una lista de los elementos requeridos para el taller, y un mecanismo para establecer contacto con la/el disertante o docente durante el día.

Al comienzo del evento

- Recuerda a las y los asistentes el código de conducta.
- Toma lista y crea una lista de nombres para pegar en la página compartida para tomar notas.
- Reparte pequeñas notas adhesivas.
- Asegúrate que tengan acceso a internet.
- Asegúrate que pueden acceder a la página compartida.
- Registra información relevante sobre la identificación de las/los asistentes en sus perfiles online.

Al finalizar el evento

- Actualiza la lista de participantes.
- Lleva un registro de la retroalimentación brindada por las/los participantes.
- Haz una copia de la página compartida.

Equipo de viaje

Aquí algunas cosas que las/los docentes llevan consigo a los talleres:

notas adhesivas y caramelos para suavizar la garganta
zapatos cómodos y pequeña libreta de notas
adaptador de corriente eléctrica de repuesto y camisa de repuesto
desodorante y adaptadores para video
pegatinas (*stickers*) para computadoras y tus notas (impresas o en una tableta)
barrita de cereal o similar y antiácido (problema de comer al paso)
tarjeta de presentación y anteojos/lentes de contacto de repuesto
libreta y bolígrafo, y puntero láser
vaso térmico para té/café y marcadores de pizarra adicionales
cepillo de dientes o enjuague bucal y toallitas húmedas descartables (puede volcarse algo encima de tu ropa)

Al viajar muchas/os docentes llevan además zapatos deportivos, traje de baño, mat de yoga o el material que necesiten para hacer actividad física. También una conexión WiFi portátil por si la de la habitación no funciona, y alguna memoria USB con los instaladores del software que las/los estudiantes aprenderán.

F.4. Diseño de lecciones

Esta sección resume el diseño de lecciones por el método hacia atrás o *backward* en inglés, que fue desarrollado independientemente por [Wigg2005; Bigg2011; Fink2013]. Propone una progresión paso a paso para ayudarte a pensar en qué hacer en cada uno y en el orden adecuado y proporciona ejercicios breves espaciados para que puedas reorientar o redirigir tu esfuerzo sin demasiadas sorpresas desagradables.

Del paso 2 en adelante será considerado en tu lección final por lo que no se trata de un desperdicio de esfuerzo: como se describió en el Chapter 6, construir ejercicios de práctica desde el comienzo te ayuda a asegurarte que todo lo que preguntes a las/los estudiantes contribuirá a los objetivos de la lección y que todo lo que necesitan saber está cubierto.

Los pasos se describen en orden creciente de detalle pero el proceso en sí es siempre iterativo. Con frecuencia y a medida que resuelvas preguntas más avanzadas, volverás a revisar tus respuestas en trabajos anteriores, y te darás cuenta que tu plan inicial no iba a funcionar como pensaste originalmente.

¿Para quién es esta lección?

Crea algunas/os estudiantes tipo (Section 6.1) o (mejor aún) elige entre los que tú y tus colegas han creado para uso general. Cada estudiante tipo debe tener:

1. un contexto general,
2. lo que ya sabe,
3. lo que cree que quiere saber y
4. qué necesidades especiales tiene.

Ejercicio breve: resumen breve de a quién estás intentando ayudar.

¿Cuál es la idea principal?

Responde tres o cuatro de las preguntas a continuación sólo enumerando elementos para ayudarte a descifrar el enfoque de la lección. No necesitas responder todas las preguntas, y puedes plantear y responder otras preguntas si creer que ayudarán, pero debes incluir sí o sí un par de respuestas a la primera pregunta. Además, en esta etapa puedes crear un mapa conceptual (Section 3.1).

- ¿Qué problemas aprenderán a resolver?
- ¿Cuáles conceptos y técnicas aprenderán?
- ¿Cuáles herramientas tecnológicas, paquetes y funciones usarán?
- ¿Qué términos de la jerga definirás?
- ¿Qué analogías usarás para explicar conceptos?
- ¿Qué errores o conceptos equivocados esperas encontrar?
- ¿Cuáles grupos de datos utilizarás?

Ejercicio breve enfoque general y sin detalles de la lección. Compártelo con una/un colega — una breve devolución en esta instancia puede ahorrar horas de esfuerzo más tarde.

¿Qué harán las/los estudiantes durante la lección?

Establezca los objetivos del Paso 2 escribiendo descripciones detalladas de algunos ejercicios que las/los estudiantes serán capaces de resolver al final de la lección. Hacer esto es análogo a test-driven development[1]: en vez de trabajar en función de un conjunto de objetivos de aprendizaje (probablemente ambiguos), hazlo "hacia atrás": elabora ejemplos concretos que quieres que puedan resolver tus estudiantes. Esto además permite dejar en evidencia requisitos técnicos necesarios que de otro modo podrían no descubrirse hasta que fuera demasiado tarde.

Para complementar la descripción detallada de los ejercicios escribe la descripción de uno o dos ejercicios para cada hora de lección como una lista de conceptos breve para mostrar qué tan rápido esperas que las/los estudiantes avancen. De nuevo, esto permitirá tener una visión realista sobre lo que asumiste de las/los estudiantes y ayudará a hacer evidentes los requisitos técnicos necesarios para resolver el ejercicio. Una manera de elaborar estos ejercicios adicionales es hacer una lista con las habilidades que necesitan para resolver los ejercicios principales y crear un ejercicio que aborde cada una.

Ejercicio breve: 1–2 ejercicios explicados de principio a fin que usen las habilidades que las/los estudiantes van a aprender, y una media docena de ejercicios con su solución esquematizada. Incluye soluciones completas para que puedas asegurarte que el programa que usen funciona.

¿Cómo están conectados los conceptos

Coloca los ejercicios que creaste en un orden lógico y a partir de ellos deriva el esquema general de una lección. El esquema debe tener 3–4 ítems por hora de clase con una evaluación formativa para cada uno. En esta etapa es común que modifiques las evaluaciones de forma que puedan basarse sobre las anteriores.

Ejercicio breve: el esquema de una lección. Es muy probable que te encuentres con que te habías olvidado de algunos elementos y que no están incluidos en tu trabajo hasta aquí, así que no te sorprendas si debes ir y venir varias veces.

Descripción general de la lección

Ahora puedes escribir la descripción general de la lección que incluya:

- un párrafo de descripción (i.e. un discurso de venta para tus estudiantes),
- media docena de objetivos de aprendizaje y
- un resumen de los requisitos.

[1] https://en.wikipedia.org/wiki/Test-driven_development

Hacer esto antes suele ser un esfuerzo inútil ya que el material que compone la lección aumenta, se recorta o cambia de lugar en las etapas anteriores.

Ejercicio breve: descripción del curso, objetivos de aprendizaje y requisitos.

F.5. Cuestionario pre-evaluación

Este cuestionario ayuda a las/los docentes a estimar el conocimiento previo sobre programación de las/los participantes de un taller introductorio a JavaScript. Las preguntas y respuestas son concretas y el cuestionario es corto, para que no resulte intimidante.

1. ¿Cuál de estas opciones describe mejor tu experiencia con la programación en general?

   - No tengo ninguna experiencia.
   - He escrito unas pocas líneas de código alguna vez.
   - He escrito programas para uso personal de un par de páginas de extensión.
   - He escrito y mantenido porciones grandes de programas.

2. ¿Cuál de estas opciones describe mejor tu experiencia con la programación en JavaScript?

   - No tengo ninguna experiencia.
   - He escrito unas pocas líneas de código alguna vez.
   - He escrito programas para uso personal de un par de páginas de extensión.
   - He escrito y mantenido porciones grandes de programas.

3. ¿Cuál de estas opciones describe mejor cuán fácil te resultaría escribir un programa en el lenguaje de programación que prefieras para hallar el número más alto en una lista?

   - No sabría por dónde comenzar.
   - Podría resolverlo con prueba y error y realizando bastantes búsquedas en internet.
   - Lo resolvería rápido con poco o nada de ayuda externa.

4. ¿Cuál de estas opciones describe mejor cuán fácil te resultaría escribir un programa en JavaScript para hallar y cambiar a mayúscula todos los títulos de una página web?

   - No sabría por dónde comenzar.
   - Podría resolverlo con prueba y error y realizando bastantes búsquedas en internet.
   - Lo resolvería rápido con poco o nada de ayuda externa.

5. ¿Qué te gustaría saber o poder hacer al finalizar esta clase que no sabes o puedes hacer ahora?

G

Ejemplos de mapas conceptuales

Estos mapas conceptuales fueron creados por Amy Hodge de la Universidad de Stanford y se reutilizan con permiso.

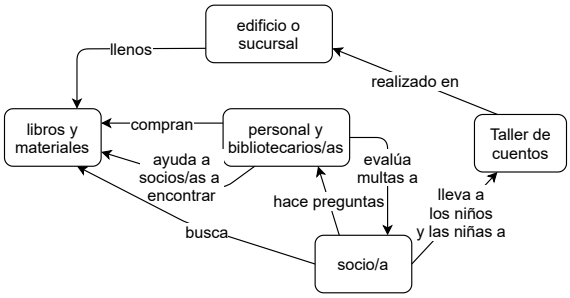

Figura G.1: Mapa conceptual desde el punto de vista los/las socios/as de la biblioteca

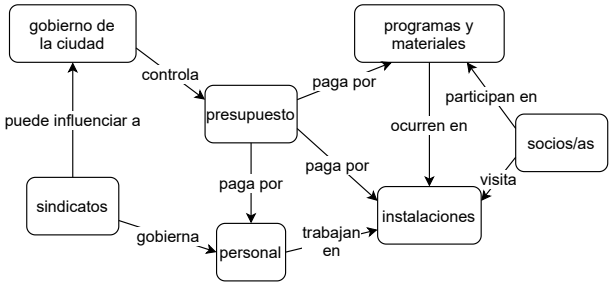

Figura G.2: Mapa conceptual desde el punto de vista de la dirección de la biblitoteca

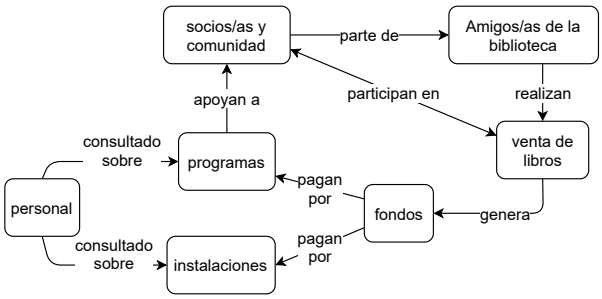

Figura G.3: Mapa conceptual desde el punto de vista de los/las amigos/as de la biblioteca

H

Solución del ejercicio de particionar

Mira el último ejercicio en Chapter 3 para la representación completa de estos símbolos.

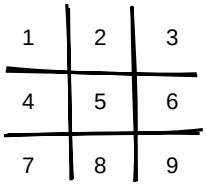

Figura H.1: Representación particionada

Índice alfabético