La programación como construcción de teoría ~ Peter Naur

La siguiente es una traducción libre del paper *Programming as Theory Building* publicado por Peter Naur en 1985, cuyo texto original puede encontrarse en el siguiente enlace: https://algoritmos-iii.github.io/assets/bibliografia/programming-as-theory-building.pdf.

La misma fue realizada por colaboradores del curso Leveroni de Algoritmos y Programación III, en la Facultad de Ingeniería de la Universidad de Buenos Aires. Comentarios, consultas y reportes de errores pueden enviarse a fiuba-algoritmos-iii-doc@googlegroups.com

Se presentan algunas opiniones sobre la programación, tomada en un sentido amplio y considerándola como una actividad humana. Aceptando que los programas no solo tendrán que ser diseñados y producidos, pero también modificados para atender las demandas cambiantes del mundo. Se concluye que el objetivo principal de la programación no es producir programas, sino que los programadores construyan teorías sobre la forma en que los problemas en cuestión son resueltos mediante la ejecución de programas. Se discute sobre las implicaciones de tal visión de la programación en asuntos tales como la vida y modificación de un programa, métodos de desarrollo de sistemas y el estatus profesional de los programadores.

Palabras clave: General; Términos Generales: Factores Humanos, Teoría, psicología de la programación, metodología de la programación.

1. Introducción

La presente discusión es una contribución a la comprensión de qué es la programación. Sugiere que programar debería ser considerada una actividad por la cual los programadores forman o logran un cierto tipo de intuición, una teoría, de los asuntos que están tratando. Esta propuesta contrasta con lo que parece ser una noción más común, según la cual la programación debe considerarse como la producción de un programa y otros textos relacionados.

Algunos de los antecedentes de las opiniones aquí presentadas se encuentran en ciertas observaciones de lo que en realidad sucede con los programas y los equipos de desarrollo que se ocupan de ellos, particularmente en las situaciones que surgen de ejecuciones o reacciones inesperadas y quizás erróneas de un programa, y en ocasiones en que estos son modificados. La dificultad de acomodar tales observaciones dentro de una visión de la programación como producción de programas sugiere que esa visión es engañosa. Se presenta como alternativa el punto de vista de la construcción de teoría.

Un trasfondo más general de la presentación es la convicción de que es importante tener una comprensión apropiada de qué es la programación. Si nuestra comprensión es inapropiada no podremos comprender las dificultades que surgen en la actividad, y nuestros intentos por superarlas dará lugar a conflictos y frustraciones.

En la presente discusión se comenzará por delinear parte de la experiencia previa crucial. A esto le seguirá la explicación de una teoría sobre qué es programar, denotada como la Visión de Construcción de Teoría.

2. La programación y el conocimiento del programador

Usaré la palabra *programación* para denotar toda la actividad de diseño e implementación de soluciones programadas. Lo que me interesa es la actividad de asociar ciertas partes y aspectos significativos de una actividad en el mundo real a la manipulación formal de símbolos que puede realizar un programa al ejecutarse en una computadora. Con esa noción surge directamente que la actividad de programación de la que estoy hablando debe incluir el desarrollo a lo largo del tiempo correspondiente a los cambios que ocurren en las actividades del mundo real afectadas por la ejecución del programa.

Una forma de expresar el principal punto que quiero desarrollar es que la programación en este sentido debe ser, principalmente, los programadores construyendo conocimiento de cierto tipo, conocimiento que debe ser considerado básicamente una pertenencia inmediata de los programadores, siendo cualquier documentación un producto auxiliar o secundario.

Como antecedente previo a desarrollar la visión dada en las siguientes secciones, lo que resta de esta sección estará dedicado a describir parte de la experiencia real de lidiar con grandes programas, que me parecía cada vez más y más significativa a medida que reflexionaba sobre los problemas. En cualquier caso la experiencia es mía o me fue comunicada por personas que tienen contacto directo con la actividad en cuestión.

El primer caso se refiere a un compilador. Ha sido desarrollado por un grupo A para un lenguaje L, y funcionó muy bien en la computadora X. Ahora otro grupo B tiene la tarea de escribir un compilador para un lenguaje L+M, una modesta extensión de L, para la computadora Y. El grupo B decide que el compilador para L desarrollado por el grupo A será un buen punto de partida para su diseño, y cierran un contrato con el grupo A para obtener de ellos soporte en forma de documentación completa, incluyendo los textos anotados del programa, discusiones por escrito sobre el diseño, y también asesoramiento personal. El acuerdo fue efectivo y el grupo B logró desarrollar el compilador que querían. En este contexto, el tema significativo es la importancia del asesoramiento personal del grupo A en lo que tenía que ver con la implementación de las extensiones M al lenguaje. Durante la fase de diseño el grupo B hizo sugerencias sobre la forma en que las extensiones deberían realizarse, y las envió al grupo A para su revisión. En múltiples casos importantes resultó que las soluciones sugeridas por el grupo B no hacían uso de las facilidades que no sólo eran inherentes a la estructura del compilador ya existente, sino que estaban extensamente discutidas en su documentación y, en cambio, estaban basadas en agregados en forma de parches a esa estructura, en detrimento de su potencia y simplicidad. Los miembros del grupo A pudieron detectar estos casos instantáneamente y pudieron proponer soluciones más simples y efectivas, encuadradas en la estructura ya existente. Este es un ejemplo sobre cómo el texto completo del programa y su documentación adicional no son suficientes para transmitir al altamente motivado grupo B la comprensión más profunda del diseño, esa teoría que los miembros del grupo A tienen inmediatamente presente.

En los años siguientes a estos eventos, el compilador desarrollado por el grupo B fue tomado por otros programadores de la misma organización, sin la orientación del grupo A. Información obtenida por un miembro del grupo A sobre los resultados de las modificaciones realizadas al compilador 10 años después dejó en claro que, en esa etapa posterior, la poderosa estructura original todavía era visible, pero su efectividad fue completamente anulada por agregados amorfos de todo tipo. Así, nuevamente, el texto del programa y su documentación probaron ser insuficientes como portadores de las ideas más importantes del diseño.

El caso 2 se refiere a la instalación y el diagnóstico de fallas de un gran sistema para monitorear en tiempo real actividades de producción industrial. El sistema es comercializado por su productor, siendo cada entrega del sistema adaptada individualmente a su entorno específico de sensores y dispositivos de visualización. El tamaño del programa entregado en cada instalación ronda las 200.000 líneas. La experiencia relevante del modo en que se maneja este tipo de sistema se encuentra en el rol y forma de trabajo del grupo de programadores de instalación y detección

de fallas. Los hechos son, primero, que estos programadores se han dedicado de lleno al sistema, como una ocupación de tiempo completo durante un período de varios años, desde que el sistema estaba siendo diseñado. Segundo, al diagnosticar una falla estos programadores confían casi exclusivamente en su conocimiento inmediato del sistema y el texto del programa anotado, y no pueden concebir que cualquier tipo de documentación adicional les resulte de utilidad. En tercer lugar, otros grupos de programadores que son responsables de operar instalaciones particulares del sistema, y por ende reciben documentación del sistema y orientación para su uso de parte del personal del productor, usualmente encuentran dificultades que tras consultar con los programadores de instalación y detección de fallas del productor resultan provenir de una incomprensión de la documentación existente, que pueden aclararse fácilmente por dichos programadores.

En conclusión, resulta evidente que al menos con ciertos tipos de programas grandes, la continua adaptación, modificación y corrección de errores en ellos depende esencialmente de un cierto tipo de conocimiento que poseen los programadores que están estrecha y continuamente vinculados con ellos.

3. La noción de teoría según Ryle

Si se admite que la programación debe incluir, como parte esencial, una construcción del conocimiento de los programadores, el siguiente tema es caracterizar ese conocimiento con más detalle. Aquí consideraremos la propuesta de que el conocimiento de los programadores debería ser considerado como una teoría, en el sentido de Ryle [11]. Muy brevemente, una persona que tiene o posee una teoría en este sentido, sabe cómo hacer ciertas cosas y además puede sostenerlo con explicaciones, justificaciones y respuestas a consultas sobre la actividad en cuestión. Cabe destacar que la noción de teoría de Ryle aparece como un ejemplo de lo que K. Popper [10] denomina objetos inanimados del Mundo 3 y, por lo tanto, tiene una posición filosófica defendible. En esta sección describiremos en detalle la noción de teoría según Ryle.

Ryle [11] desarrolla su noción de teoría como parte de su análisis de la naturaleza de la actividad intelectual, particularmente la forma en que la actividad intelectual difiere y trasciende a la actividad meramente inteligente. En el comportamiento inteligente un individuo no muestra un conocimiento particular sobre hechos, sino la habilidad de hacer ciertas cosas, como hacer y entender bromas, hablar gramaticalmente o pescar. En particular, el desempeño inteligente se caracteriza, en parte, porque el individuo lo realiza correctamente de acuerdo con ciertos criterios, pero muestra, además, su habilidad para aplicar esos criterios con el fin de detectar

y corregir fallas, aprender del ejemplo de otros, etc. Puede observarse que esta noción de inteligencia no se basa en ninguna noción de que el comportamiento inteligente depende de que la persona siga o adhiera a reglas, prescripciones o métodos. Por el contrario, el solo acto de adherir a reglas puede hacerse más o menos inteligentemente; si el ejercicio de la inteligencia dependiera de seguir reglas, debería haber reglas acerca de cómo seguir reglas, y acerca de cómo seguir las reglas acerca de seguir reglas, etc. en una regresión infinita, lo cual es absurdo.

Lo que caracteriza a la actividad intelectual por sobre la actividad meramente inteligente, es que el individuo construye y se apropia de una teoría, entendida como el conocimiento que una persona debe tener, no solo para realizar ciertas cosas inteligentemente sino también para poder explicarlas, responder consultas acerca de ellas, discutirlas, etc. Una persona que tiene una teoría está preparada para enfrentar esas actividades; mientras construye la teoría, la persona está intentando obtenerla.

La noción de teoría en el sentido usado aquí aplica no solo a las elaboradas construcciones de campos especializados de investigación, sino también a actividades en las que cualquier persona educada participará en ciertas ocasiones. Incluso actividades poco ambiciosas de la vida cotidiana pueden dar lugar a que las personas teoricen, por ejemplo, al planificar cómo acomodar muebles, o cómo llegar a algún lugar mediante ciertos medios de transporte.

La noción de teoría usada aquí es explícitamente *no* limitada a lo que podría llamarse la parte más general o abstracta de la intuición. Por ejemplo, para tener la teoría de la mecánica de Newton como la entendemos aquí, no alcanza con entender las leyes centrales, como que la fuerza es igual al producto entre la masa y la aceleración. Adicionalmente, como se describe más en detalle en Kuhn [4, p. 187ff], la persona que tiene la teoría debe entender la forma en que las leyes centrales aplican a ciertos aspectos de la realidad, como para ser capaz de reconocer y aplicar la teoría sobre otros aspectos similares. Una persona que tiene la teoría de la mecánica de Newton debe entonces entender cómo se aplica a los movimientos pendulares y de los planetas, y debe ser capaz de reconocer fenómenos similares en el mundo, para poder utilizar correctamente las reglas de la teoría expresadas matemáticamente.

El hecho de que una teoría dependa de la comprensión de ciertos tipos de similitudes entre situaciones y acontecimientos del mundo real nos da la razón por la cual el conocimiento de alguien que posee la teoría no puede, en principio, ser expresado en términos de reglas. En efecto, las similitudes en cuestión no son, ni pueden ser, expresadas en términos de criterios, así como tampoco pueden ser

formuladas las similitudes entre muchos otros tipos de objetos, como los rostros humanos, las melodías o los sabores de vino.

4. La teoría a ser construida por el programador

En términos de la noción de teoría de Ryle, lo que el programador debe construir es una teoría de cómo ciertos aspectos de la realidad serán manejados, o representados, por un programa de computadora. En la Visión de Construcción de Teoría (en adelante VCT) de la programación, la teoría que el programador construye tiene más importancia que otros productos como la documentación, las especificaciones, o los comentarios.

En la argumentación a favor de la VCT, el mayor problema es demostrar cómo el conocimiento que posee el programador, en virtud de haber construido la teoría, trasciende aquello escrito en la documentación. La respuesta es que el conocimiento del programador es superior en al menos tres áreas esenciales:

1. El programador que tiene la teoría del programa puede explicar cómo la solución se relaciona con los problemas del mundo real que ayuda a resolver. Esta explicación deberá tener en cuenta la manera en la que los problemas del mundo, tanto en características generales como en detalles, son mapeadas al texto del programa, y a cualquier documentación adicional. De modo que el programador deberá poder explicar, para cada parte del texto del programa, y para cada característica de su estructura, con qué aspecto del mundo se relacionan, y viceversa. Es obvio que la mayor parte de los aspectos del mundo quedarán fuera del alcance del programa, siendo irrelevantes en su contexto. Sin embargo, la decisión de qué partes del mundo son relevantes solo puede ser hecha por alguien que entiende el mundo entero. Y este entendimiento debe ser proporcionado por el programador.
2. El programador que tiene la teoría del programa puede explicar por qué cada parte del programa es como es. En otras palabras, puede defender el texto del programa con algún tipo de justificación. El argumento final de esta justificación es, y siempre debe ser, el conocimiento directo e intuitivo del programador. Esto es cierto incluso cuando la justificación hace uso de la razón, sea mediante la aplicación de reglas de diseño, de estimaciones cuantitativas, de comparaciones con alternativas, etc. El punto es que la elección de los principios y reglas a seguir, y la decisión de que son relevantes a la situación actual, en el análisis final deben mantenerse producto del conocimiento directo del programador.

3. El programador que tiene la teoría del programa puede responder constructivamente a cualquier demanda de modificación del programa, para manejar los problemas del mundo en una manera distinta. El diseño de cómo incorporar una modificación de la mejor forma a un programa ya establecido depende de la percepción de la similitud entre la nueva demanda y las operaciones ya implementadas en el programa. El tipo de similitud que debe ser percibida es una entre aspectos del mundo. Esto solo tiene sentido para el agente con conocimiento del mundo, el programador, y no puede ser reducido a un conjunto limitado de reglas o criterios, por razones similares a las dadas sobre por qué la justificación de un programa no puede ser reducida.

A pesar de que se hayan presentado algunos argumentos básicos a favor de la adopción de la VCT de la programación, una evaluación de la visión debe tener en cuenta hasta qué punto la misma contribuye a un entendimiento coherente de la programación y sus problemas. Estos asuntos se discutirán en las secciones siguientes.

5. Problemas y costos de la modificación de programas

Una de las razones destacadas para proponer la VCT de la programación es el deseo de establecer un entendimiento de la programación que permita comprender mejor las modificaciones a los programas. Esta pregunta será entonces la primera en ser analizada.

Un aspecto en el que todos parecen estar de acuerdo es en que el software será modificado. Sucede invariablemente que una vez en producción, un programa se sentirá solo como una parte de la solución al problema que intenta resolver. Además, el uso mismo del programa servirá de inspiración para nuevas ideas sobre funcionalidades útiles que el programa debe tener. De aquí entonces la necesidad de poder manejar correctamente los cambios.

La cuestión de las modificaciones a un programa está íntimamente relacionada con la de los costos de la programación. Ante la necesidad de un cambio en la forma de operar de un programa, se espera ahorrar costos al modificar el programa existente, en lugar de crear un nuevo programa desde cero. Sin embargo, esa idea llama a un análisis más profundo.

En primera instancia, debe ser señalado que la expectativa de que es posible modificar un programa a bajo costo no puede ser avalada por analogía con las modificaciones a otros tipos de construcciones humanas complejas. Para muchos de estos tipos de construcciones, como los autos o los televisores, las modificaciones están completamente fuera de alcance en la práctica. Y en los

lugares donde ocasionalmente se llevan a cabo modificaciones, como en los edificios, se sabe que suelen ser demasiado costosas, y una demolición completa del edificio seguida de la construcción de uno nuevo es muchas veces económicamente más viable. En segundo lugar, la expectativa de las modificaciones a bajo costo encuentra sustento en el hecho de que un programa es solo texto en un medio que permite su fácil edición. Sin embargo, para que tal argumento sea válido, se debe asumir que el costo dominante es el de la manipulación de texto. Esto estaría en concordancia con una noción de la programación vista como producción de texto. En la Visión de Construcción de Teoría todo este argumento es inválido. La misma no respalda la expectativa de que modificar programas a bajo costo sea, en la mayoría de los casos, posible.

Otra cuestión estrechamente relacionada es la de la flexibilidad de los programas. Al incorporar flexibilidad a un programa estamos incluyendo en su diseño cierta funcionalidad que no es inmediatamente requerida, pero que probablemente resulte útil en el futuro. Entonces un programa flexible es capaz de manejar ciertas clases de cambios en las circunstancias externas sin ser modificado.

A menudo se afirma que los programas deberían ser diseñados para ser tan flexibles como se pueda, con el fin de ser fácilmente adaptables a los cambios. Este consejo es razonable cuando se habla de flexibilidad fácilmente lograda. Sin embargo, la flexibilidad generalmente solo se alcanza a un costo muy grande. Cada aspecto de ella debe ser diseñado, incluyendo qué circunstancias debe cubrir, y bajo qué parámetros debe ser controlada. Luego debe ser implementada, testeada, y descrita. Todo esto para obtener funcionalidad cuya utilidad depende completamente de eventos futuros. Debe ser obvio entonces que la flexibilidad integrada en el programa no es una solución viable a la necesidad de adaptar los programas a las circunstancias cambiantes del mundo.

Cuando se modifica un programa, una solución ya programada debe ser cambiada para reaccionar a una alteración en la actividad del mundo real con la que se relaciona. En una modificación, el primer paso debe ser confrontar la solución existente con las demandas de la modificación deseada. En esta confrontación se determinará el tipo y nivel de similitud entre las nuevas demandas y las capacidades de la solución actual. Es aquí donde se aprecia el mérito de la Visión de Construcción de Teoría. Efectivamente, es durante la determinación de similitudes donde se hacen evidentes las deficiencias de cualquier enfoque que ignore la necesidad de participación directa de las personas que posean el conocimiento apropiado. El punto es que el tipo de similitud que debe ser determinada es accesible al ser humano que posee la teoría del programa, y está completamente

fuera de alcance de lo que puede ser determinado con reglas, ya que hasta el criterio sobre el cual juzgar no puede ser formulado. Luego, a partir del conocimiento de las similitudes entre los nuevos requerimientos y los ya satisfechos por el programa, el programador es capaz de diseñar el cambio en el texto del programa que se debe realizar para implementar la modificación.

En cierto sentido, hablar de una modificación de teoría sería incorrecto, ya que solo se puede hablar de una modificación de programa. Una persona que posee la teoría ya debe estar preparada para responder al tipo de preguntas y demandas que puedan dar lugar a las modificaciones. Esta observación lleva a una conclusión importante: los problemas de modificar un programa surgen al actuar bajo el supuesto de que la programación consiste en producir el texto del programa, en lugar de entenderla como una actividad de construcción de teoría.

Siguiendo la VCT, se hace entendible el deterioro del texto del programa como resultado de las modificaciones hechas por programadores sin una comprensión adecuada de la teoría subyacente. Ciertamente, si se las ve como un mero cambio en el texto del programa y del comportamiento externo de su ejecución, una modificación dada puede realizarse de muchas maneras diferentes, todas correctas. Sin embargo, vistas en relación a la teoría del programa, estas maneras pueden parecer muy diferentes: algunas se adaptan correctamente a la teoría o la extienden de manera natural, y otras son completamente inconsistentes con esa teoría, teniendo la apariencia de parches poco integrados en el programa. Esta desigualdad de carácter de los diferentes cambios solo tiene sentido para un programador con la teoría del programa. Al mismo tiempo, el carácter de los cambios hechos es vital para la viabilidad a largo plazo del programa. Para que un programa mantenga su calidad es necesario que cada modificación esté firmemente arraigada en su teoría. Esto se da porque la noción misma de cualidades como la simpleza, o la buena estructura, solo pueden ser entendidas en términos de la teoría, ya que caracterizan al texto del programa en relación a otros textos posibles que podrían haber sido usados para obtener el mismo comportamiento, pero que existen solo como posibilidades en la mente del programador.

6. La vida, muerte, y resurrección de un programa

Uno de los principales postulados de la VCT de la programación es que una parte esencial de cualquier programa, su teoría, es algo que no puede ser expresado, sino que está intrínsecamente atado a las personas. Se desprende entonces que al describir el estado de un programa sea importante indicar hasta qué punto siguen estando a cargo los programadores que poseen su teoría. Como una forma de

resaltar este hecho, se podría extender la noción de desarrollo de un programa con las nociones de “vida”, “muerte” y “resurrección” del mismo. La construcción de un programa es la construcción de su teoría por parte de un equipo de desarrolladores. Durante la “vida” de un programa, este se mantiene bajo el control activo de un equipo de programadores que poseen su teoría, y que específicamente se hacen cargo de todas sus modificaciones. La “muerte” de un programa ocurre cuando se disuelve el equipo de programadores que tenían su teoría. Un programa muerto puede continuar siendo ejecutado en una computadora, y seguir produciendo resultados útiles. Sin embargo, el estado de muerte se hace evidente cuando las demandas de modificación del programa no pueden ser inteligentemente resueltas. La “resurrección” de un programa es la reconstrucción de su teoría por parte de un nuevo equipo de programadores.

La extensión de la vida de un programa, de acuerdo con estas nociones, depende entonces de cómo la nueva generación de programadores toma posesión de su teoría. Para que un programador nuevo pueda obtener la teoría existente de un programa no alcanza con que tenga la oportunidad de familiarizarse con el texto del programa u otra documentación. Lo que se requiere es que el nuevo programador trabaje en contacto estrecho con programadores que ya tengan la teoría. De esta manera podrá familiarizarse con el lugar que ocupa el programa en el contexto de las situaciones del mundo real, y así adquirir el conocimiento de cómo funciona el programa y de cómo se manejan, dentro de la teoría, las reacciones inusuales del programa y sus modificaciones. Esta problemática de educar a los nuevos programadores en la teoría existente de un programa es similar al problema de educar en otras actividades donde el conocimiento dominante es de carácter práctico, por ejemplo, en el caso de la escritura, o de tocar un instrumento musical. La forma de aprendizaje más importante es que el estudiante haga las actividades relevantes bajo supervisión y guía. En el caso de la programación, la actividad debería incluir una discusión de la relación entre el programa y los aspectos relevantes del mundo real, y de los límites establecidos en los asuntos de la realidad que trata el programa.

Una consecuencia muy importante de la VCT es que la resurrección de un programa, el restablecimiento de su teoría solamente en base a la documentación, es estrictamente imposible. Para que esta afirmación parezca más razonable, hay que tener en cuenta que rara vez surge la necesidad de revivir un programa completamente muerto. Sería extraño que un equipo nuevo de programadores sea asignado a revivir un programa sin tener al menos un poco de conocimiento de la teoría que formó el equipo original. Aun así, la VCT sugiere fuertemente que solo se debe intentar revivir un programa en situaciones excepcionales. En el mejor de los

casos será un proceso muy costoso, y probablemente se genere una teoría diferente a la que tenían los autores originales, por lo que el texto del programa puede contener discrepancias.

En lugar de intentar revivir un programa, la VCT sugiere descartar el programa existente y darle al nuevo equipo la oportunidad de resolver el problema desde cero. Este procedimiento es más probable que genere un programa viable, a un precio similar, o incluso menor, al de intentar revivir el original. El argumento es que construir una teoría para que encaje con un programa existente es una actividad difícil, frustrante, y que consume mucho tiempo. Es probable que el nuevo programador se sienta dividido entre la lealtad al programa original, con cualquier problema o debilidad que pueda llegar a tener, y la teoría nueva que construyó, que para bien o para mal, probablemente difiera de la teoría inicial.

Problemas similares surgen cuando un programa se mantiene continuamente vivo por un equipo de programadores que evoluciona con el tiempo, como resultado de las diferentes competencias y experiencias de quienes lo conforman, ya que el reemplazo de los desarrolladores individuales del equipo se vuelve inevitable a largo plazo.

7. Método y construcción de teoría

En los últimos años se ha visto mucho interés en los métodos de programación. En la presente sección se harán algunos comentarios acerca de la relación entre la Visión de Construcción de Teoría y las nociones detrás de los métodos de programación.

Para empezar, ¿qué es un método de programación? Esto no siempre se aclara, incluso por los autores que recomiendan algún método en particular. Aquí, un método de programación será considerado como un conjunto de reglas de trabajo para los programadores, que les dicen qué cosas deberían hacer, en qué orden, qué notaciones o lenguajes usar, y qué tipos de documentos producir en las varias etapas.

Al comparar esta noción de método con la VCT de la programación, el tema más importante es el de las acciones u operaciones, y su orden. Hablar de un "método" implica que el desarrollo de programas puede y debe proceder como una secuencia de acciones de cierto tipo, cada acción llevando a un tipo particular de resultado documentado. En la VCT lo más importante es la construcción de la teoría, mientras que la producción de otros documentos es secundaria. Al construir la teoría no puede haber una secuencia particular de acciones, ya que la teoría desarrollada por una persona no tiene una división inherente en partes ni un orden específico. Más

bien, la persona que posee una teoría será capaz de producir presentaciones de varios tipos sobre la base de la misma, en respuesta a preguntas o demandas. Con respecto al uso de estilos particulares de notación o de formalización, esto solo puede ser un tema secundario ya que el objeto primario, la teoría, no es, y no puede ser, expresada, por lo que preguntar sobre su forma de expresión no tiene sentido.

Se desprende que, en la Visión de Construcción de Teoría, no puede haber un método correcto para la actividad principal de la programación.

Podría parecer que esta conclusión entra en conflicto con la opinión establecida, en varias maneras, y entonces es posible que se la tome como un argumento en contra de la VCT. Dos contradicciones aparentes serán tratadas a continuación, la primera relacionada con la importancia de los métodos en el ejercicio de la ciencia, y la segunda referida al éxito de los métodos como son realmente usados en el desarrollo de software.

El primer argumento es que el desarrollo de software debe basarse en las costumbres científicas, y por lo tanto debe emplear procedimientos similares a los métodos científicos. La falla de este argumento está en asumir que hay tal cosa como un método científico, y que sea útil para los científicos. Esta pregunta ha sido el sujeto de mucho debate en años recientes, y la conclusión de autores tales como Feyerabend [2], tomando sus ilustraciones de la historia de la física, y Medawar [5], argumentando como un biólogo, es que la noción del método científico como un conjunto de instrucciones es erróneo.

Esta conclusión no es contradicha por trabajos tales como los de Polya [8, 9] sobre la solución de programas. Este trabajo toma sus ilustraciones desde el campo de las matemáticas y conduce a una percepción que también es altamente relevante para la programación. Sin embargo, no se puede afirmar que presenta un método con el cual proceder. Más bien, es una colección de sugerencias que apuntan a estimular la actividad mental de quien resuelve problemas, al señalar diferentes modos de trabajo que pueden ser aplicados en cualquier secuencia.

El segundo argumento que pareciera contradecir la desestimación de los métodos es que el uso de algunos métodos particulares ha sido exitoso, de acuerdo a informes publicados. A este argumento se podría responder que un estudio metódicamente satisfactorio de la eficacia de los métodos de programación no pareciera, hasta ahora, haberse hecho. Tal estudio debería emplear la técnica bien establecida de experimentos controlados (cf. Brooks [1] o Moher y Schneider [6]). La falta de tales estudios es explicable en parte por el alto costo en el cual sin lugar a dudas se incurriría en tales investigaciones si los resultados fueran a ser

significativos, en parte por los problemas de establecer, en una manera operacional, los conceptos que subyacen los llamados métodos en el campo del desarrollo de programas. La mayoría de los reportes publicados acerca de esos métodos meramente describen y recomiendan ciertas técnicas y procedimientos, sin establecer su utilidad o eficacia en alguna forma sistemática. Un estudio elaborado de cinco métodos diferentes por C. Floyd y varios colaboradores [3] concluye que la noción de métodos como sistemas de reglas que en un contexto arbitrario mecánicamente conducirán a buenas soluciones es una ilusión. Lo que resta es el efecto de los métodos en la educación de los programadores. Esta conclusión es totalmente compatible con la VCT de la programación. De hecho, en esta visión la calidad de la teoría construida por el programador dependerá en gran parte de la familiaridad del programador con las soluciones modelo a problemas típicos, con técnicas de descripción y verificación, y con principios de estructuración de sistemas formados por muchas partes en interacciones complicadas. Así, muchos de los elementos de interés de los métodos son relevantes para la construcción de teoría. Donde la VCT difiere con los metodólogos es en el tema de qué técnicas usar y en qué orden. En la VCT esto permanece como una cuestión que el programador debe decidir, teniendo en cuenta el problema específico a resolver.

8. El estatus del programador y la Visión de Construcción de Teoría

Las áreas en las que las consecuencias de la Visión de Construcción de Teoría contrastan con las de las visiones actuales más extendidas son aquellas acerca de la contribución personal de los programadores a la actividad, y del estatus adecuado de los programadores.

La diferencia entre la VCT y la visión más prevaleciente de la contribución personal de los programadores es evidente en gran parte del debate común sobre la programación. Como un simple ejemplo, consideremos el estudio de la capacidad de modificar grandes sistemas de software por Oskarsson [7]. Este estudio brinda información extensiva acerca de un considerable número de modificaciones en una nueva versión de un gran sistema comercial. La descripción cubre el contexto, sustancia, e implementación de cada modificación, con particular atención a la manera en que los cambios son limitados a determinados módulos del programa. Sin embargo, no hay sugerencia alguna de que la implementación de las modificaciones pudiera depender de los antecedentes de los 500 programadores empleados en el proyecto, como el tiempo que llevan trabajando en él, y no hay indicación de la manera en que las decisiones de diseño fueron distribuidas entre los 500 programadores. Aún así, la importancia de una teoría subyacente se admite

indirectamente en afirmaciones como "las decisiones fueron implementadas en el bloque equivocado" y en referencia a una "philosophy of AXE" [se deja la cita en su idioma original]. Sin embargo, por la manera en que el estudio fue conducido, estas afirmaciones solo permanecen como indicaciones aisladas.

En general, mucha de la discusión actual de programación parece asumir que la programación es similar a la producción industrial, donde el programador es considerado como un componente de esa producción, un componente que debe ser controlado por reglas de procedimiento y que puede ser reemplazado fácilmente. Otra visión relacionada es que los seres humanos rinden más si actúan como máquinas, siguiendo reglas, con un énfasis en los modos formales de expresión, lo que permite formular ciertos argumentos en términos de reglas o manipulación formal. Esas reglas están de acuerdo con la noción, aparentemente común entre personas que trabajan con computadoras, de que la mente humana trabaja como una computadora. Al nivel de la gestión industrial, estas visiones apoyan la idea de tratar a los programadores como trabajadores de baja responsabilidad, y de poca educación.

En la Visión de Construcción de Teoría, el resultado primario de la actividad de programar es la teoría mantenida por los programadores. Como esta teoría es, por su propia naturaleza, parte de una posesión mental de cada programador, se implica que la noción del programador como un componente fácilmente reemplazable en la actividad de producción de programas debe ser abandonada. En cambio, el programador debe ser considerado como un desarrollador responsable y un gerente de la actividad en la cual la computadora es una parte. Para llevar a cabo su tarea, a él o a ella se le debe dar una posición permanente y de un estatus similar al de otros profesionales, tales como ingenieros y abogados, cuya contribución activa como empleados surge de su competencia intelectual.

Elevar el estatus de los programadores, tal como sugiere la VCT, debe ser acompañado de una correspondiente reorientación en la educación de los programadores. Si bien habilidades como el dominio de las notaciones, las representaciones de datos, y los procesamiento de datos, siguen siendo importantes, el énfasis debería estar en aumentar el entendimiento y el talento para la formación de teorías. Hasta qué punto esto se puede enseñar, permanece como una pregunta abierta. El enfoque más prometedor sería hacer que el estudiante trabaje en problemas concretos bajo supervision, en un entorno activo y constructivo.

9. Conclusiones

Aceptando que las modificaciones a los programas, demandadas por las circunstancias externas cambiantes, son una parte esencial de la programación, se argumenta que el objetivo principal de la programación es que los programadores construyan una teoría tal que los hechos en cuestión puedan ser respaldados por la ejecución de un programa. Tal visión lleva a una noción de la vida de un programa que depende de su continuo mantenimiento por los programadores que poseen su teoría. Además, siguiendo esta visión, la noción de un método de programación, entendido como un conjunto de reglas de procedimiento a ser seguidas por el programador, está basada en asunciones inválidas, y debe ser rechazada. Otras consecuencias de este enfoque se vinculan con el estatus que debe otorgarse a los programadores como desarrolladores responsables, permanentes y gerentes de la actividad de la cual la computadora es solo un aspecto; y con que su educación debe enfatizar el ejercicio de la construcción de teoría, a la par de la adquisición de conocimiento de procesamiento de datos y notaciones.

Referencias

[1] R.E. Brooks, Studying Programmer Behaviour Experimentally, Commun. ACM 23, no. 4 (1980) 207-213.

[2] P. Feyerabend, Against Method (Verso Editions, London, 1978; ISBN: 86091-700-2).

[3] C. Floyd, Eine Untersuchung von Software-Entwicklungsmethoden, in: H. Morgenbrod and W. Sammer, Eds., Programmierumgebungen und Compiler, Tagung I/1984 des German Chapter of the ACM (Teubner Verlag, Stuttgart, 1984; ISBN: 3-519-02437-3) 248-274.

[4] T.S. Kuhn, The Structure of Scientific Revolutions, Second Edition (University of Chicago Press, Chicago, 1970; ISBN: 0-226-45803-2).

[5] P. Medawar, Pluto's Republic (Oxford University Press, Oxford, 1982; ISBN: 0-19-217726-5).

[6] T. Moher and G.M. Schneider, Methodology and experimental Research in Software Engineering, Int. J. Man-Mach. Stud. 16, (1 jan. 1982) 65-87.

[7] O. Oskarsson, Mechanisms of Modifiability in Large Software Systems (Linköping Studies in Science and Technology, Dissertations, no. 77, Linköping , 1982; ISBN: 91-7372- 527-7).

[8] G. Polya, How To Solve It (Doubleday Anchor Book, New York, 1957).

[9] G. Polya, Mathematics and Plausible Reasoning (Princeton University Press, Princeton, New Jersey, 1954).

[10] K.R. Popper and J.C. Eccles, The Self and lts Brain (Routledge and Kegan Paul, London, 1977).

[11] G. Ryle, The Concept of Mind (Penguin Books Ltd., Harmondsworth, England, 1963).