La Programación como Construcción de Teoría

Peter Naur

Introducción

La presente discusión es una contribución a la comprensión de qué es la programación. Se sugiere que la programación, en su esencia, debe considerarse como una actividad mediante la cual los programadores forman o adquieren un cierto tipo de comprensión, una teoría, sobre los asuntos en cuestión. Esta sugerencia contrasta con lo que parece ser una noción más común: que la programación debe verse como la mera producción de un programa y ciertos otros textos.

Parte del trasfondo de las ideas aquí expuestas se encuentra en ciertas observaciones sobre lo que realmente sucede con los programas y los equipos de programadores que trabajan con ellos, especialmente en situaciones que surgen a partir de ejecuciones inesperadas y quizás erróneas de los programas, así como en ocasiones de modificaciones de estos. La dificultad de acomodar tales observaciones dentro de una visión de la programación como mera producción sugiere que esta visión es engañosa. En su lugar, se presenta la visión de la construcción de teoría como una alternativa.

El contexto más general de esta presentación radica en la convicción de que es importante tener una comprensión adecuada de lo que es la programación. Si nuestra comprensión es inadecuada, malinterpretamos las dificultades que surgen en esta actividad, y nuestros intentos por superarlas generarán conflictos y frustraciones. En la discusión actual, primero se esbozará parte de la experiencia clave de fondo. Luego, se explicará una teoría sobre la naturaleza de la programación, denominada la **Visión de Construcción de Teoría**. En las secciones posteriores, se explorarán algunas de las consecuencias de esta perspectiva.

La Programación y el Conocimiento de los Programadores

Usaré la palabra **programación** para referirme a toda la actividad de diseño e implementación de soluciones programadas. Lo que me interesa es la actividad de hacer coincidir una parte y un aspecto significativos de una actividad del mundo real con la manipulación formal de símbolos que puede realizar un programa en ejecución en una computadora.

Bajo esta noción, se sigue directamente que la actividad de programación que estoy describiendo debe incluir el desarrollo en el tiempo correspondiente a los cambios que ocurren en la actividad del mundo real que el programa está modelando, es decir, las modificaciones del programa.

Una forma de expresar el punto principal que quiero destacar es que la programación, en este sentido, debe entenderse principalmente como el proceso mediante el cual los programadores construyen un conocimiento de un tipo particular, un conocimiento que se considera fundamentalmente como posesión inmediata de los programadores, mientras que cualquier documentación es un producto auxiliar y secundario.

Como trasfondo para la posterior elaboración de esta perspectiva en las siguientes secciones, el resto de la presente sección describirá algunas experiencias reales en el manejo de programas de gran tamaño, experiencias que me han parecido cada vez más significativas a medida que he reflexionado sobre estos problemas. En ambos casos, la experiencia es propia o me ha sido comunicada por personas con contacto directo con la actividad en cuestión.

Caso 1: Un compilador

Este caso se refiere a un compilador desarrollado por un grupo **A** para un lenguaje **L**, el cual funcionaba muy bien en la computadora **X**. Luego, otro grupo **B** recibió la tarea de escribir un compilador para un lenguaje **L + M**, una extensión modesta de **L**, para la computadora **Y**. El grupo **B** decidió que el compilador de **L** desarrollado por el grupo **A** sería un buen punto de partida para su diseño, y firmó un contrato con el grupo **A** para recibir apoyo en forma de documentación completa, incluyendo textos del programa con anotaciones y extensas discusiones escritas sobre el diseño, así como asesoramiento personal.

Este acuerdo fue efectivo, y el grupo **B** logró desarrollar el compilador que deseaba. Sin embargo, en el contexto actual, el aspecto más relevante es la importancia del asesoramiento personal del grupo **A** en lo que respecta a la implementación de las extensiones **M** al lenguaje. Durante la fase de diseño, el grupo **B** propuso distintas formas de integrar estas extensiones y las presentó al grupo **A** para su revisión. En varios casos importantes, el grupo **A** descubrió que las soluciones propuestas por el grupo **B** no aprovechaban las facilidades inherentes en la estructura del compilador existente, a pesar de que estas facilidades no solo estaban presentes en el diseño, sino que también se discutían en detalle en la documentación. En cambio, las soluciones de **B** introducían modificaciones a la estructura en forma de parches que destruían su poder y simplicidad.

Los miembros del grupo **A** fueron capaces de detectar estos problemas de inmediato y propusieron soluciones simples y efectivas, enmarcadas completamente dentro de la estructura existente. Esto demuestra cómo el texto completo del programa y la documentación adicional no fueron suficientes para transmitir al grupo **B**, a pesar de su alta motivación, la comprensión más profunda del diseño, esa teoría que estaba presente de manera inmediata en los miembros del grupo **A**.

En los años siguientes, el compilador desarrollado por el grupo **B** pasó a ser manejado por otros programadores de la misma organización, sin la orientación del grupo **A**. Información obtenida por un miembro del grupo **A** sobre el compilador tras aproximadamente 10 años de modificaciones reveló que, aunque la estructura original seguía siendo visible, su efectividad había sido completamente anulada por la acumulación amorfa de numerosas adiciones de distintos tipos. Así, una vez más, el texto del programa y su documentación demostraron ser insuficientes como vehículo para transmitir algunas de las ideas de diseño más importantes.

Caso 2: Un sistema en tiempo real

Este caso trata sobre la instalación y el diagnóstico de fallos en un gran sistema en tiempo real utilizado para monitorear actividades de producción industrial. El sistema es comercializado por su fabricante, y cada entrega del sistema se adapta individualmente a su entorno específico de sensores y dispositivos de visualización. El tamaño del programa en cada instalación es del orden de **200,000 líneas** de código.

La experiencia relevante en la forma en que se maneja este tipo de sistema está relacionada con el rol y la manera de trabajar del grupo de programadores encargados de la instalación y detección de fallos. Los hechos clave son los siguientes:

1. Estos programadores han trabajado estrechamente con el sistema como su ocupación a tiempo completo durante varios años, desde la etapa en que el sistema estaba en fase de diseño.

2. Cuando diagnostican un fallo, estos programadores confían casi exclusivamente en su conocimiento inmediato del sistema y en el texto anotado del programa, sin considerar útil ningún otro tipo de documentación adicional.

3. Otros grupos de programadores, responsables de la operación de instalaciones específicas del sistema y que reciben documentación y orientación completa sobre su uso por parte del personal del fabricante, regularmente encuentran dificultades. Sin embargo, al consultar con los programadores encargados de la instalación y detección de fallos, estas dificultades suelen atribuirse a una comprensión inadecuada de la documentación existente, aunque pueden resolverse fácilmente con la ayuda de estos programadores especializados.

La conclusión parece ineludible: al menos en ciertos tipos de programas grandes, la continua adaptación, modificación y corrección de errores depende esencialmente de un tipo particular de conocimiento poseído por un grupo de programadores que están estrecha y continuamente vinculados con ellos.

La Noción de Teoría según Ryle

Si se acepta que la programación debe implicar, como parte esencial, la construcción del conocimiento del programador, el siguiente paso es caracterizar ese conocimiento con mayor precisión. Lo que se considerará aquí es la sugerencia de que el conocimiento de los programadores debería entenderse propiamente como una **teoría**, en el sentido de **Ryle [1949]**.

De manera muy breve, una persona que posee una teoría en este sentido sabe cómo hacer ciertas cosas y, además, puede respaldar su ejecución con explicaciones, justificaciones y respuestas a preguntas relacionadas con la actividad en cuestión. Cabe señalar que la noción de teoría de Ryle aparece como un ejemplo de lo que **K. Popper** [Popper y Eccles, 1977] llama **objetos no encarnados del Mundo 3**, lo que le otorga una base filosófica defendible.

En esta sección, describiremos la noción de teoría de Ryle con más detalle.

Ryle [1949] desarrolla su concepto de teoría como parte de su análisis sobre la naturaleza de la actividad intelectual, en particular sobre la manera en que la actividad intelectual difiere y va más allá de una actividad meramente inteligente.

En el comportamiento inteligente, una persona no muestra necesariamente un conocimiento específico de hechos, sino la habilidad de hacer ciertas cosas, como contar y entender chistes, hablar con gramática correcta o pescar. Más aún, el desempeño inteligente se caracteriza no solo por hacer estas cosas bien, según ciertos criterios, sino también por la capacidad de aplicar estos criterios para detectar y corregir errores, aprender de los ejemplos de otros, etc.

Es importante destacar que esta noción de inteligencia no se basa en la idea de que el comportamiento inteligente dependa del seguimiento de reglas, prescripciones o métodos. Por el contrario, el acto mismo de seguir reglas puede hacerse de manera más o menos inteligente. Si la inteligencia dependiera de seguir reglas, sería necesario establecer reglas sobre cómo seguir reglas, y reglas sobre cómo seguir esas reglas, lo que conduciría a una **regresión infinita**, lo cual es absurdo.

Lo que distingue la **actividad intelectual** de una actividad meramente inteligente es que la persona desarrolla y posee una teoría. Aquí, la **teoría** se entiende como el conocimiento necesario no solo para realizar ciertas tareas con inteligencia, sino también para **explicarlas, responder preguntas sobre ellas, argumentar sobre ellas**, etc.

Una persona que posee una teoría está preparada para participar en tales actividades, mientras que una persona que la está construyendo aún está en proceso de adquirirla.

La noción de teoría utilizada aquí no se limita únicamente a las construcciones elaboradas de campos especializados de investigación. También se aplica a actividades en las que cualquier persona educada participa en determinadas ocasiones. Incluso actividades cotidianas y poco ambiciosas pueden dar lugar a la **teorización**, por ejemplo, al planificar cómo colocar muebles en una habitación o cómo llegar a un destino utilizando ciertos medios de transporte.

La noción de teoría aquí empleada **no se restringe a la parte más general o abstracta del conocimiento**. Por ejemplo, para poseer la teoría de la mecánica de **Newton**, no basta con comprender las leyes centrales, como la ecuación **Fuerza = Masa × Aceleración**.

Además, como lo describe en más detalle **Kuhn**, quien posee la teoría debe entender **cómo las leyes centrales se aplican a ciertos aspectos de la realidad**, de modo que pueda reconocer y aplicar la teoría en otros contextos similares.

Por ejemplo, alguien que comprende la mecánica de Newton debe entender cómo se aplica a **los movimientos de los péndulos y los planetas**, y debe ser capaz de reconocer fenómenos similares en el mundo, de manera que pueda emplear adecuadamente las reglas matemáticas de la teoría.

La dependencia de una teoría respecto a la **capacidad de identificar similitudes entre situaciones y eventos del mundo real** es la razón por la cual el conocimiento de alguien que posee la teoría **no puede, en principio, expresarse en términos de reglas**.

De hecho, las similitudes en cuestión **no pueden expresarse en términos de criterios**; del mismo modo que muchas otras formas de reconocimiento, como identificar **rostros humanos, melodías o sabores de vino**, tampoco pueden expresarse mediante reglas explícitas.

La Teoría que Debe Construir el Programador

En términos de la noción de teoría de **Ryle**, lo que debe construir el programador es una **teoría sobre cómo ciertos aspectos del mundo serán manejados o respaldados por un programa de computadora**.

Desde la perspectiva de la **construcción de teorías en la programación**, la teoría desarrollada por los programadores tiene **primacía** sobre otros productos, como el código fuente del programa, la documentación para el usuario y cualquier otra documentación adicional, como las especificaciones.

Para argumentar a favor de esta perspectiva, la cuestión fundamental es demostrar cómo el conocimiento del programador, al poseer la teoría, necesariamente y de manera esencial, **trasciende** aquello que está registrado en los documentos producidos.

La respuesta a esta cuestión es que el conocimiento del programador supera lo registrado en la documentación en al menos **tres áreas esenciales**:

1. **Relación entre el programa y el mundo real**
    - Un programador que posee la teoría del programa puede **explicar cómo la solución se relaciona con los aspectos del mundo real que ayuda a manejar**.
    - Esta explicación debe abordar la manera en que los aspectos del mundo real, tanto en sus características generales como en sus detalles, están, de alguna forma, **mapeados** en el código fuente y en cualquier otra documentación adicional.
    - Por lo tanto, el programador debe ser capaz de explicar, para cada parte del código y cada una de sus estructuras generales, **qué aspecto o actividad del mundo representa**.
    - A la inversa, para cualquier aspecto del mundo real, el programador debe poder indicar su forma de mapeo en el código.
    - La mayoría de los aspectos y actividades del mundo, por supuesto, estarán **fuera del alcance del programa**, ya que serán irrelevantes en el contexto. Sin embargo, la **decisión de qué partes del mundo son relevantes solo puede ser tomada por alguien que comprende el mundo en su totalidad**,

es decir, el programador.

2. **Justificación del diseño del programa**

    - Un programador que posee la teoría del programa puede explicar **por qué cada parte del código es como es**, es decir, puede respaldar el código con una justificación.

    - En última instancia, la **base de esta justificación debe ser el conocimiento intuitivo o estimativo del programador**.

    - Incluso cuando la justificación se basa en razonamientos, como la aplicación de reglas de diseño, estimaciones cuantitativas o comparaciones con alternativas, la elección de **qué principios y reglas aplicar, y la decisión de que son relevantes para la situación, sigue dependiendo del conocimiento directo del programador**.

3. **Capacidad de adaptación y modificación del programa**

    - Un programador que posee la teoría del programa puede responder de manera constructiva a cualquier **demanda de modificación**, para que el programa respalde nuevos aspectos del mundo real.

    - Diseñar la mejor forma de incorporar una modificación en un programa ya establecido depende de la **percepción de la similitud entre la nueva demanda y las funcionalidades ya existentes** en el programa.

    - El tipo de **similitud** que debe ser percibido es entre **aspectos del mundo real**.

    - Solo tiene sentido para una persona que posee un conocimiento profundo del mundo, es decir, el programador, y **no puede reducirse a un conjunto limitado de reglas o criterios**.

    - Esto es por las mismas razones por las que la justificación del programa **no puede expresarse completamente en términos de reglas**.

Si bien esta sección presenta algunos argumentos fundamentales para adoptar la perspectiva de la **construcción de teorías en la programación**, una evaluación de esta visión debería considerar en qué medida **contribuye a una comprensión coherente de la programación y sus problemas**.

Estos temas serán discutidos en las siguientes secciones.

Problemas y Costos de las Modificaciones de Programas

Una razón importante para proponer la **visión de construcción de teorías en la programación** es el deseo de desarrollar una comprensión adecuada sobre las modificaciones de programas. Por lo tanto, esta será la primera cuestión que se analizará.

Existe un consenso general: **el software siempre será modificado**. Invariablemente, cuando un programa entra en operación, se percibe como solo una **parte** de la solución a los problemas planteados. Además, el **uso mismo del programa** suele inspirar ideas para nuevos servicios que debería ofrecer. De ahí surge la necesidad de contar con mecanismos para gestionar modificaciones.

La cuestión de las modificaciones está estrechamente ligada a la de los **costos de programación**. Ante la necesidad de cambiar la forma en que opera un programa, se espera lograr **ahorros en los costos** modificando el código existente en lugar de escribir un programa completamente nuevo.

Sin embargo, la expectativa de que las modificaciones de programas **sean económicas** merece un análisis más profundo. En primer lugar, debe notarse que esta expectativa **no puede justificarse por analogía** con modificaciones en otras construcciones complejas hechas por el ser humano. En muchos casos, como en la arquitectura, las modificaciones pueden ser **costosas**, al punto de que, económicamente, a menudo resulta preferible **demoler y reconstruir desde cero** en lugar de adaptar la estructura existente.

En segundo lugar, la idea de que modificar programas **es barato** podría basarse en el hecho de que un programa es un **texto** almacenado en un medio que permite **ediciones sencillas**. Para que este argumento fuera válido, debería asumirse que el costo dominante de la programación es **la manipulación del texto**. Esto coincidiría con la idea de que programar es solo **producir código**. Sin embargo, **desde la visión de construcción de teorías, este argumento es falso**. Esta perspectiva no respalda la expectativa de que las modificaciones de programas sean, en general, económicas.

La Flexibilidad en los Programas

Otro tema estrechamente relacionado es el de la **flexibilidad del programa**. Al incluir flexibilidad en un software, se agregan **capacidades operativas** que no son estrictamente necesarias en el presente, pero que podrían ser útiles en el futuro. Así, un programa flexible puede manejar **ciertos cambios en las condiciones externas** sin requerir modificaciones.

Frecuentemente se afirma que los programas **deberían diseñarse con un alto grado de flexibilidad** para que sean fácilmente **adaptables** a circunstancias cambiantes. Este consejo puede ser razonable cuando la flexibilidad se logra **sin gran esfuerzo**, pero en términos generales, **la flexibilidad conlleva un costo significativo**.

- Cada característica flexible debe **diseñarse**, lo que implica determinar qué circunstancias debe cubrir y qué parámetros la controlarán.

- Luego, debe **implementarse, probarse y documentarse**.

- Este costo se incurre en función de una característica cuya **utilidad dependerá enteramente de eventos futuros**.

Por lo tanto, debe quedar claro que la **flexibilidad integrada en un programa no es una solución general** para la necesidad de adaptación del software a un mundo en constante cambio.

Modificación de Programas y la Construcción de Teorías

Modificar un programa implica **ajustar una solución existente** para que pueda **abordar un cambio en la actividad del mundo real** que debe reflejar.

Lo primero que se necesita en una modificación es **confrontar la solución existente con las nuevas exigencias**. En este proceso, es necesario **determinar el grado y tipo de similitud** entre las capacidades del programa actual y los nuevos requerimientos.

Esta necesidad de **evaluar similitudes** resalta el valor de la **visión de construcción de teorías**. De hecho, es precisamente en la determinación de similitudes donde se evidencian las deficiencias de cualquier enfoque que **ignore la importancia de la participación directa de personas con el conocimiento adecuado**.

El punto clave es que **el tipo de similitud que debe reconocerse sólo puede ser comprendido por quienes poseen la teoría del programa**. No puede reducirse a reglas fijas, ya que **ni siquiera los criterios para evaluarla pueden ser completamente formulados**.

A partir de la comprensión de la similitud entre los nuevos requerimientos y los ya cubiertos por el programa, el programador puede **diseñar los cambios necesarios en el código** para implementar la modificación de manera efectiva.

En cierto sentido, no puede hablarse de una modificación de la teoría, sino únicamente de una modificación del programa. De hecho, una persona que posee la teoría debe estar ya preparada para responder a los tipos de preguntas y demandas que pueden dar lugar a modificaciones del programa. Esta observación conduce a la importante conclusión de que los problemas de la modificación de programas surgen de actuar bajo la suposición de que programar consiste en la producción de texto del programa, en lugar de reconocer la programación como una actividad de construcción de teorías.

Sobre la base de la visión de construcción de teorías, la degradación de un texto de programa como resultado de modificaciones realizadas por programadores que no comprenden adecuadamente la teoría subyacente se vuelve comprensible. De hecho, si se considera simplemente como un cambio en el texto del programa y en el comportamiento externo de la ejecución, una modificación deseada puede, por lo general, realizarse de muchas maneras diferentes, todas correctas. Al mismo tiempo, si se analiza en relación con la teoría del programa, estas maneras pueden parecer muy distintas: algunas pueden ajustarse a la teoría o extenderla de manera natural, mientras que otras pueden ser completamente inconsistentes con ella, teniendo quizás el carácter de parches no integrados en la parte principal del programa.

Esta diferencia en la naturaleza de los diversos cambios solo puede tener sentido para el programador que posee la teoría del programa. Al mismo tiempo, la naturaleza de los cambios realizados en un texto de programa es crucial para la viabilidad a largo plazo del programa. Para que un programa conserve su calidad, es imprescindible que cada modificación se base firmemente en la teoría del mismo. De hecho, la propia noción de cualidades como simplicidad y buena estructura solo puede entenderse en términos de la teoría del programa, ya que caracterizan el texto del programa en relación con otros textos de programa que podrían haberse escrito para lograr el mismo comportamiento de ejecución, pero que solo existen como posibilidades en la comprensión del programador.

Vida, Muerte y Resurgimiento de un Programa

Una afirmación central de la visión de construcción de teorías en la programación es que una parte esencial de cualquier programa, su teoría, es algo que no puede expresarse de manera concebible y está inextricablemente ligada a los seres humanos. De ello se desprende que, al describir el estado de un programa, es importante indicar en qué medida los programadores que poseen su teoría siguen a cargo de él. Para enfatizar esta circunstancia, se podría extender la noción de construcción de programas a las nociones de vida, muerte y resurgimiento de un programa. La construcción de un programa es equivalente a la construcción de su teoría por y dentro del equipo de programadores. Durante la vida del programa, un equipo de programadores que posee su teoría mantiene un control activo sobre él y, en particular, conserva el control sobre todas sus modificaciones. La muerte de un programa ocurre cuando el equipo de programadores que posee su teoría se disuelve. Un programa muerto puede seguir ejecutándose en una computadora y produciendo resultados útiles. El estado real de muerte se hace visible cuando no se pueden atender de manera inteligente las demandas de modificación del programa. El resurgimiento de un programa consiste en la reconstrucción de su teoría por un nuevo equipo de programadores.

La vida extendida de un programa, según estas nociones, depende de que nuevas generaciones de programadores adopten la teoría del programa. Para que un nuevo programador llegue a poseer la teoría existente de un programa, no es suficiente que tenga la oportunidad de familiarizarse con el texto del programa y otra documentación. Lo que se requiere es que el nuevo programador tenga la oportunidad de trabajar en estrecho contacto con los programadores que ya poseen la teoría, de modo que pueda familiarizarse con el lugar del programa en el contexto más amplio de las situaciones relevantes del mundo real y adquirir el conocimiento de cómo funciona el programa y cómo se manejan dentro de la teoría del programa las reacciones inusuales y las modificaciones. Este problema de educación de nuevos programadores en la teoría existente de un programa es bastante similar al problema educativo en otras actividades donde el conocimiento de cómo hacer ciertas cosas predomina sobre el conocimiento de que ciertas cosas son de determinada manera, como la escritura o la interpretación de un instrumento musical. La actividad educativa más importante es que el estudiante realice las tareas pertinentes bajo la supervisión y guía adecuadas. En el caso de la programación, la actividad debería incluir discusiones sobre la relación entre el programa y los aspectos y actividades relevantes del mundo real, así como sobre los límites impuestos a los asuntos del mundo real abordados por el programa.

Una consecuencia muy importante de la visión de construcción de teorías es que el resurgimiento de un programa, es decir, el restablecimiento de su teoría únicamente a partir de la documentación, es estrictamente imposible. Para evitar que esta consecuencia parezca irrazonable, cabe señalar que probablemente rara vez surgirá la necesidad de resucitar un programa completamente muerto, ya que difícilmente se asignaría el resurgimiento a nuevos programadores sin que al menos alguno tuviera algún conocimiento de la teoría original. Aun así, la visión de construcción de teorías sugiere firmemente que el resurgimiento de un programa solo debería intentarse en situaciones excepcionales y con plena conciencia de que, en el mejor de los casos, será costoso y podría dar lugar a una teoría revivida que difiera de la que originalmente tenían los autores del programa, lo que podría generar discrepancias con el texto del programa.

En lugar del resurgimiento de un programa, la visión de construcción de teorías sugiere que el texto del programa existente debería ser descartado y que el nuevo equipo de programadores debería tener la oportunidad de resolver el problema desde cero. Un procedimiento de este tipo es más probable que genere un programa viable que el resurgimiento del programa, y a un costo no mayor, e incluso posiblemente menor. El punto es que construir una teoría que se ajuste y sostenga un texto de programa existente es una actividad difícil, frustrante y que consume mucho tiempo. El nuevo programador probablemente se sentirá dividido entre la lealtad al texto del programa existente, con todas las posibles oscuridades y debilidades que pueda contener, y la nueva teoría que debe desarrollar, la cual, para bien o para mal, probablemente diferirá de la teoría original detrás del texto del programa.

Problemas similares pueden surgir incluso cuando un programa se mantiene continuamente activo mediante un equipo de programadores en evolución, debido a las diferencias de competencia y experiencia previa entre los miembros del equipo, especialmente cuando la operatividad del equipo se mantiene a través de la inevitable rotación de sus integrantes.

Método y Construcción de Teorías

En los últimos años ha habido un gran interés por los métodos de programación. En esta sección se harán algunos comentarios sobre la relación entre la visión de construcción de teorías y las nociones detrás de los métodos de programación.

Para empezar, ¿qué es un método de programación? Esto no siempre se deja en claro, incluso por los autores que recomiendan un método en particular. Aquí se considerará un método de programación como un conjunto de reglas de trabajo para programadores, que indican qué tipo de cosas deben hacer, en qué orden, qué notaciones o lenguajes utilizar y qué tipos de documentos producir en distintas etapas.

Al comparar esta noción de método con la visión de construcción de teorías en la programación, la cuestión más importante es la de las acciones u operaciones y su orden. Un método implica la afirmación de que el desarrollo de un programa puede y debe avanzar como una secuencia de acciones de ciertos tipos, en la que cada acción conduce a un resultado documentado particular. En la construcción de una teoría, no puede haber una secuencia particular de acciones, ya que una teoría sostenida por una persona no tiene una división intrínseca en partes ni un orden inherente. Más bien, la persona que posee una

teoría será capaz de generar distintas presentaciones a partir de ella, en respuesta a preguntas o requerimientos.

En cuanto al uso de notaciones o formalizaciones específicas, nuevamente, esto solo puede ser una cuestión secundaria, ya que el elemento principal, la teoría, no se expresa ni puede expresarse, por lo que no surge la cuestión de la forma de su expresión.

De ello se desprende que, según la visión de construcción de teorías, en la actividad principal de la programación no puede existir un método correcto.

Esta conclusión puede parecer estar en conflicto con la opinión establecida de varias maneras y, por lo tanto, podría considerarse un argumento en contra de la visión de construcción de teorías. Aquí se abordarán dos aparentes contradicciones: la primera se relaciona con la importancia del método en la búsqueda científica y la segunda con el éxito de los métodos tal como se aplican en el desarrollo de software.

El primer argumento sostiene que el desarrollo de software debe basarse en métodos científicos y, por lo tanto, debe emplear procedimientos similares a los de la ciencia. El error de este argumento es la suposición de que existe un "método científico" y que este es útil para los científicos. Esta cuestión ha sido objeto de mucho debate en los últimos años, y la conclusión de autores como Feyerabend (1978), basándose en ejemplos de la historia de la física, y Medawar (1982), argumentando desde la biología, es que la noción de un método científico como un conjunto de directrices para el científico en ejercicio es errónea.

Esta conclusión no se contradice con trabajos como los de Polya (1954, 1957) sobre resolución de problemas. Estos estudios toman sus ejemplos del campo de las matemáticas y ofrecen ideas altamente relevantes también para la programación. Sin embargo, no pueden considerarse como la presentación de un método a seguir. Más bien, constituyen una colección de sugerencias destinadas a estimular la actividad mental del solucionador de problemas, señalando distintos modos de trabajo que pueden aplicarse en cualquier secuencia.

El segundo argumento que podría parecer contradecir el rechazo del método en la Visión de Construcción de Teorías es que el uso de ciertos métodos ha sido exitoso, según informes publicados. A este argumento se le puede responder que hasta el momento no parece haberse realizado un estudio metodológicamente satisfactorio sobre la eficacia de los métodos de programación. Un estudio de este tipo debería emplear la técnica bien establecida de experimentos controlados (cf. [Brooks, 1980] o [Moher y Schneider, 1982]).

La ausencia de tales estudios puede explicarse en parte por el alto costo que sin duda implicarían estas investigaciones si los resultados fueran significativos, y en parte por los problemas de establecer de manera operativa los conceptos subyacentes a lo que en el ámbito del desarrollo de programas se denomina "métodos". La mayoría de los informes publicados sobre estos métodos simplemente describen y recomiendan ciertas técnicas y procedimientos, sin establecer su utilidad o eficacia de manera sistemática.

Un estudio detallado de cinco métodos diferentes realizado por C. Floyd y varios coautores [Floyd, 1984] concluyó que la noción de los métodos como sistemas de reglas que, en

cualquier contexto y de manera mecánica, conducirán a buenas soluciones es una ilusión. Lo que permanece es el efecto de los métodos en la educación de los programadores.

Esta conclusión es completamente compatible con la Visión de Construcción de Teorías en la programación. De hecho, desde esta perspectiva, la calidad de la teoría construida por el programador dependerá en gran medida de su familiaridad con soluciones modelo de problemas típicos, con técnicas de descripción y verificación, y con principios de estructuración de sistemas compuestos por muchas partes en interacción compleja. Así, muchos de los aspectos de interés en los métodos son relevantes para la construcción de teorías.

Donde la Visión de Construcción de Teorías se aparta de la postura de los metodólogos es en la cuestión de qué técnicas utilizar y en qué orden. Según esta visión, esto debe seguir siendo enteramente una decisión del programador, considerando el problema específico que se debe resolver.

El Estatus de los Programadores y la Visión de Construcción de Teorías

Las áreas donde las consecuencias de la Visión de Construcción de Teorías contrastan más notablemente con las de las visiones más prevalentes en la actualidad son las relacionadas con la contribución personal de los programadores a la actividad y su estatus adecuado.

El contraste entre la Visión de Construcción de Teorías y la visión más común sobre la contribución personal de los programadores es evidente en gran parte de la discusión común sobre la programación. Como solo un ejemplo, considere el estudio de la modificabilidad de grandes sistemas de software realizado por Oskarsson [1982]. Este estudio proporciona información extensa sobre una considerable cantidad de modificaciones en una versión de un gran sistema comercial. La descripción cubre el contexto, el contenido y la implementación de cada modificación, prestando especial atención a la forma en que los cambios en el programa se limitan a módulos específicos. Sin embargo, no hay sugerencia alguna de que la implementación de las modificaciones podría depender del trasfondo de los 500 programadores empleados en el proyecto, como la cantidad de tiempo que han estado trabajando en él, y no hay indicios sobre cómo se distribuyen las decisiones de diseño entre los 500 programadores.

A pesar de esto, la importancia de una teoría subyacente es reconocida indirectamente en afirmaciones como "las decisiones se implementaron en el bloque equivocado" y en una referencia a "una filosofía de AXE". Sin embargo, debido a la forma en que se llevó a cabo el estudio, estas admisiones solo pueden quedar como indicaciones aisladas.

Más generalmente, gran parte de la discusión actual sobre programación parece asumir que la programación es similar a la producción industrial, considerando al programador 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 mejor si actúan como máquinas, siguiendo reglas, con un énfasis consecuente en los modos formales de expresión, que permiten formular ciertos argumentos en términos de reglas de manipulación formal. Estas visiones coinciden bien con la noción, aparentemente común entre las personas que trabajan con computadoras, de que la mente humana funciona como una computadora. A nivel de gestión industrial, estas

visiones respaldan tratar a los programadores como trabajadores de responsabilidad relativamente baja y con una educación breve.

En la Visión de Construcción de Teorías, el resultado principal de la actividad de programación es la teoría que poseen los programadores. Dado que esta teoría, por su propia naturaleza, es parte de la posesión mental de cada programador, se sigue que la noción del programador como un componente fácilmente reemplazable en la actividad de producción del programa debe ser abandonada. En su lugar, el programador debe ser considerado como un desarrollador responsable y un gerente de la actividad en la que la computadora es una parte. Para ocupar esta posición, debe ser otorgado un puesto permanente, con un estatus similar al de otros profesionales, como ingenieros y abogados, cuya contribución activa como empleadores de empresas se basa en su competencia intelectual.

El aumento del estatus de los programadores sugerido por la Visión de Construcción de Teorías tendrá que ser respaldado por una reorientación correspondiente en la educación de programadores. Si bien habilidades como el dominio de notaciones, representaciones de datos y procesos de datos siguen siendo importantes, el énfasis principal debe girar hacia el fomento de la comprensión y el talento para la formación de teorías. Hasta qué punto esto pueda ser enseñado debe seguir siendo una pregunta abierta. El enfoque más prometedor sería hacer que el estudiante trabaje en problemas concretos bajo orientación, en un ambiente activo y constructivo.

Conclusiones

Aceptando que las modificaciones de los programas demandadas por circunstancias externas cambiantes son una parte esencial de la programación, se argumenta que el objetivo principal de la programación es hacer que los programadores construyan una teoría sobre la forma en que los asuntos en cuestión pueden ser respaldados por la ejecución de un programa. Esta visión lleva a una noción de la vida del programa que depende del apoyo continuo del programa por parte de programadores que posean su teoría. Además, en esta visión, la noción de un método de programación, entendido como un conjunto de reglas de procedimiento que debe seguir el programador, se basa en suposiciones inválidas y, por lo tanto, debe ser rechazada. Como consecuencias adicionales de esta visión, los programadores deben ser reconocidos con el estatus de desarrolladores y gerentes responsables y permanentes de la actividad de la cual la computadora es una parte, y su educación debe enfatizar el ejercicio de la construcción de teorías, junto con la adquisición de conocimientos sobre procesamiento de datos y notaciones.

RESUMEN:

El documento "La Programación como Construcción de Teorías" de Peter Naur argumenta que la programación debe entenderse principalmente como la actividad mediante la cual los programadores forman o alcanzan un cierto tipo de comprensión, una teoría, sobre los asuntos que manejan. Esta perspectiva contrasta con la noción más común de que la programación es la producción de un programa y documentación asociada.

Naur basa su argumento en observaciones de situaciones reales, especialmente aquellas relacionadas con errores inesperados, modificaciones de programas y la dificultad de mantener la integridad de grandes sistemas a lo largo del tiempo. Estas observaciones sugieren que la documentación, aunque útil, es secundaria y no logra transmitir la comprensión profunda del diseño que poseen los programadores originales. Un ejemplo de esto se observa en el Caso 1, donde un segundo grupo tuvo dificultades para extender un compilador a pesar de contar con documentación exhaustiva, necesitando el conocimiento tácito del grupo original para evitar soluciones ineficientes.

Para fundamentar su visión, Naur se apoya en la noción de "teoría" propuesta por Ryle, que describe como el conocimiento que una persona debe poseer no solo para hacer ciertas cosas de manera inteligente, sino también para explicarlas, justificarlas y responder preguntas sobre ellas. Esta teoría no se limita a reglas explícitas, sino que incluye la capacidad de reconocer similitudes entre situaciones del mundo real, una habilidad inherentemente humana y difícil de formalizar.

En el contexto de la programación, la teoría que deben construir los programadores es una comprensión de cómo ciertos aspectos del mundo serán manejados o soportados por un programa de computadora. Esta teoría tiene primacía sobre el código, la documentación de usuario y otras especificaciones. La posesión de esta teoría permite al programador:

- Explicar cómo la solución del programa se relaciona con los asuntos del mundo que aborda, detallando la correspondencia entre el código y los aspectos del mundo real.
- Justificar por qué cada parte del programa es como es, basándose en su conocimiento intuitivo y su comprensión del diseño, incluso cuando se aplican reglas o comparaciones.
- Responder constructivamente a las demandas de modificación del programa, percibiendo la similitud entre las nuevas necesidades y las funcionalidades existentes, una evaluación que requiere comprensión del mundo real y no puede reducirse a reglas.

La visión de la programación como construcción de teorías ofrece una perspectiva valiosa sobre los problemas y costos de las modificaciones de programas. Contrario a la expectativa de modificaciones de bajo costo basadas en la facilidad de edición de texto, la teoría subraya que la modificación requiere una comprensión profunda de la solución existente y su relación con el mundo real. La falta de esta comprensión puede llevar a modificaciones que deterioran la estructura y la calidad del programa a largo plazo.

Naur introduce los conceptos de vida, muerte y resurrección de un programa para enfatizar la dependencia de la teoría en los seres humanos. Un programa vive mientras el equipo de programadores que posee su teoría lo mantiene y controla las modificaciones. Muere cuando este equipo se disuelve, dificultando o imposibilitando la respuesta inteligente a nuevas demandas. La resurrección de un programa, intentando reconstruir su teoría a partir de la documentación, se considera estrictamente imposible o extremadamente costosa y arriesgada. En muchos casos, sugiere Naur, es preferible descartar el código existente y resolver el problema de nuevo

.

En relación con los métodos de programación, la visión de la construcción de teorías sugiere que no existe un "método correcto" como un conjunto de reglas de procedimiento a seguir mecánicamente. La construcción de la teoría no sigue una secuencia predefinida de acciones. Si bien las técnicas y principios que promueven los métodos pueden ser relevantes para la formación de la teoría, la decisión de qué usar y cuándo debe recaer en el programador, considerando el problema específico.

Finalmente, la Teoría de la Construcción enfatiza la contribución personal del programador y aboga por una revalorización de su estatus. En lugar de ser visto como un componente fácilmente reemplazable en una línea de producción industrial, el programador debe ser considerado un desarrollador y gestor responsable cuya actividad se basa en su competencia intelectual. Esto implica también una reorientación de la educación de los programadores, poniendo mayor énfasis en el desarrollo de la comprensión y el talento para la formación de teorías, además de la adquisición de conocimientos técnicos.

En resumen, Peter Naur propone una comprensión de la programación centrada en el desarrollo de una teoría profunda por parte de los programadores, una visión que tiene implicaciones significativas para la gestión de proyectos, la modificación de software, la educación de programadores y su rol dentro de las organizaciones.