Forming Software Architects in Early Stages: From Craft to Engineering

Ricardo Gacitúa B., Mauricio Diéguez
Departamento de Ciencias de la Computación e Informática
Universidad de La Frontera
Temuco, Chile
Email: ricardo.gacitua, mauricio.dieguez@ufrontera.cl

Elizabeth Vidal
Universidad LaSalle
Arequipa, Perú
Email: evidal@ulasalle.edu.pe

***Abstract*—Software Architecture is the highest level design of the structure of a program or a computer system. The increase of the complexity of the systems and the major importance that the qualit attributes acquire, needs increasingly to include, in early stages, aspects of modelling and paradigms of programming. Unfortunately, the empirical evidence shows that the initial formation in courses of programming is centred on aspects of language and sintáxis more than in the modeling one of the solutions. The aim of this article is to describe a methodological approach that is being used in the first courses of programming in the University of La Frontera. This approach allows to include the design of a computational solution as principal focus. A problem solving methodology called Rapid & Global (R&G) is used to teach how to program and to facilitate the incorporation of computational concepts and paradigms of programming used in Architecture of Software. Our work shows three important results: (i) The students understand and use safely, in early stages, computational terminology as object oriented, abstraction, classes and methods; (ii) There is recognized and understands the importance of the design (modeling of the solution) in the construction of computational solutions as it increases the complexity and; (iii) The computational solutions obtained are documented, easy to read, to modify and independent from the language. This article shows that it is possible to change the paradigm with which students are formed in programming, changing the installed practice of teaching programming languages as a basis to solve problems, by a practice of modeling the solution and then writing the code.**

***Keywords*—Software Architecture, programming, modeling, non-functional requirements.**

I. Introducción

Arquitectura de Software es el diseño de más alto nivel de la estructura de un programa o de un sistema computacional y se considera uno de los factores claves en el éxito de un sistema [1]. Como el tamaño y la complejidad de los sistemas se incrementa, los algorítmos y las estructuras de datos ya no constituyen los mayores desafíos. Cuando los sistemas son construídos con muchos componentes, la organización de todo el sistema - la arquitectura de software - presenta nuevos problemas de diseño, tales como el modelado de la interacción entre componentes, su secuenciación y la semántica de los modelos [2]. Variados son los conceptos [3] que son requeridos para desarrollar arquitecturas de software, como por ejemplo patrones [4] y abstracciones (modelos) [5], los cuales constituyen elementos relevantes para interactuar con el código fuente del software. La arquitectura se selecciona y diseña en base a objetivos de los que se derivan los requisitos [6]. Pero no solamente son relevantes los requisitos funcionales [7] sino que, también los requisitos no-funcionales [8] tales como, mantenibilidad, seguridad y portabilidad. Todo lo anterior, se constituye en la base de conocimiento especializada cuyo dominio debería comenzar en etapas tempranas de la formación de los ingenieros de software, en particular en los primeros cursos de programación. Desafortunadamente, está documentado que en los primeros cursos de programación el principal foco está centrado sólo en aspectos de lenguaje y sintáxis más que en el modelado de las soluciones e inclusión de paradigmas de computación, elementos constitutivos de una arquitectura de software [9]. En términos de resolución de problemas, se puede describir un interés sólo en el Qué (Problema) y en el Con qué (Herramientas), pero muy poco en el Cómo (proceso). Es reconocido que el principal problema que enfrentan los novatos en programación es su carencia de conocimiento específico de programación y las estrategías de desarrollo [9]. En los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal, debido a la dificultad que implicaba para la mayoría de las personas. Con el paso del tiempo se han ido desarrollando formas y guías generales, que ayudan a resolver los problemas [10] [11] [12]. Según Garlan y Shaw [1] "*más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema*". He aquí, la real importancia de cambiar el foco desde los lenguajes y su sintáxis al modelado de la solución.

Este artículo describe un enfoque metodológico que ha comenzado a ser utilizado en los primeros cursos de programación en la Universidad de La Frontera. Este enfoque permite incluir el diseño de una solución computacional como foco principal, además de atributos de calidad y paradigmas de computación. Para ello, se utiliza una metodología de resolución de problemas denominada Rápida & Global (R&G) para enseñar a programar y mejorar con ello el desarrollo del pensamiento lógico computacional y facilitar la inclusión de conceptos computacionales y paradigmas de programación. Para evidenciar los resultados se realizan encuestas de opinión y de conocimiento, además de grabaciones de videos en que se registran las impresiones de los esudiantes al final del curso.

La estructura del artículo es como sigue: La sección

978-1-5386-3483-7/17/$31.00 ©2017 IEEE

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.

II presenta una breve descripción de la problemática de la formación en programación. La sección III describe la propuesta metodológica. Luego, la sección IV ilustra los resultados obtenidos de las encuestas y de las sesiones de video. La sección V presenta una breve discusión respecto de los resultados y finalmente, la sección VI resume las principales conclusiones y trabajos futuros.

II. Las bases de la Arquitectura: ¿Cómo enseñamos a modelar problemas utilizando computadores?

Mark Gusdial, en su artículo titulado: ¿Cuál es la mejor manera de enseñar ciencias de la computación a los principiantes? [13], establece que se enseña computación solicitando a los estudiantes involucrarse en actividades de profesionales del área, esto es "Programando". Se les entrega algún material de estudio, pero se espera que mucho del aprendizaje ocurra a través de la práctica. Sweller et al. [14] explica su premisa básica: ¿Por qué la mínima guía durante la instrucción no funciona: Un análisis de las fallas del constructivismo, descubrimiento y enseñanza basada en preguntas?. Ellos describen en frases algo que describe muy bien cómo se trabaja en la educación en computación: "*Al parecer hay dos principales supuestos que sustentan los programas de enseñanza utilizando mínimas guías. Primero, se desafía a los estudiantes a resolver problemas reales o adquirir conocimiento complejo en ambientes ricos en información basado en el supuesto de que, teniendo los aprendices que construir sus propias soluciones, esto lleva a experiencias de aprendizaje más efectivas. Segundo, ellos parecen asumir que el conocimiento puede ser adquirido de mejor forma a través de la experiencia basada en los procedimientos de la disciplina*". Esto parece reflejar nuestra práctica. En otras palabras, las personas deberían aprender a programar construyendo programas a partir de información básica sobre el lenguaje, y deberían hacerlo en la misma forma que los expertos lo hacen. Luego de décadas utilizando la instrucción de programación con mínimas guías, a nuestro conocimiento no hay trabajos de investigación que soporten esta técnica. Recientes trabajos en el área de enseñanza de programación apuntan a introducir lenguajes gráficos, tal como Scracth [15]. Sin embargo, aún cuando se cambia el foco desde la sintáxis del lenguaje de programación a estructuras de control gráficas (e.g. repetición, condición, asignación), el problema de no mostrar guías disciplinadas para guiar el proceso se mantiene. La pregunta que surge entonces es: ¿Por qué destacados cientificos e investigadores en el área de computación - quienes demandan rigurosas pruebas para sus afirmaciones en sus investigaciones -, continuan utilizando y defendiendo el sesgo de la intuición y la estrategia de prueba y error como método de enseñanza?, lo que está demostrado que no es lo más efectivo. ¿Será acaso, porque en muchos casos somos científicos o ingenieros inmersos en el mundo de la educación, sin la formación requerida para ello?

Parnas [16] advierte acerca de los peligros del desarrollo de software indisciplinado y establece que los estudiantes de computación no son enseñados para trabajar en forma disciplinada. En efecto, la importancia del análisis disciplinado es raramente mencionado, elemento central de la Arquitectura de Software. Es frecuente escuchar hablar acerca de la "enseñanza creativa" o que el desarrollo de software es un "arte" y, últimamente, acerca de la innovación aplicada al desarrollo de software, ésto como una forma de explicar la falta de procedimientos sistemáticos en la construcción de productos de software en etapas tempranas. Según Parnas [16] a los estudiantes se les debe enseñar "Qué hacer" y "Cómo hacerlo", incluso en un primer curso. Estamos, por tanto, frente a un gran dilema en la enseñanza: formar **Artesanos de Programas** o **Ingenieros de Software**. A nuestro conocimiento no existen trabajos de investigación recientes que se preocupen de este dilema e incorporen metodologías centradas en el diseño de soluciones. Este artículo no pretende hacer frente a la innumerable cantidad de preguntas abiertas que de este tema se derivan, sino proponer un primer enfoque metodológico y disciplinado alineado con lo expuesto por Parnas.

III. Una propuesta metodológica

En nuestra opinión, el diseño disciplinado es enseñable y posible. Éste requiere el uso de lógica básica, por tanto no es requerido lógica temporal [17] o cualquiera de los mejores métodos formales [18]. Procedimientos simples pueden ser significativamente efectivos para entender el problema, modelar una solución, encontrar defectos y mejorar la integridad. Según Parnas [16], un diseño de software disciplinado requiere tres pasos:

1) Determinar y describir el conjunto de posibles entradas al software.
2) Dividir el conjunto de entradas de tal forma que las entradas, dentro de cada partición, sean manejadas con una regla simple.
3) Definir la regla.

La próxima sección describe la metodología Rápida & Global, la cual es consistente con los pasos anteriores.

A. Metodología Rápida & Global (R&G).

La Metodología R&G guía el análisis y solución de problemas de manera sistemática y global, y puede ser utilizada para enseñar a programar. Ésta considera que los problemas computacionales, en general, pueden concebirse como un proceso y, por lo tanto, dividirse en tres principales actividades, (i) Una actividad que describa y procese el conjunto de entradas posibles, denominada "Entrada", (ii) Una actividad encargada de particionar el problema y representar su solución, denominada "Desarrollo" y (iii) Una actividad encargada de presentar los resultados, denominada "Salida". Estas actividades en conjunto describen la solución para un problema, y soporta tanto el enfoque estructurado como el orientado a objeto. Se estructura en 4 premisas básicas que son: DEFINE, RESUELVE, CONVIERTE Y CODIFICA. Esto implica primeramente definir y describir el problema, luego resolverlo utilizando un modelo (con notación gráfica) luego, convertir dicho modelo a una representación más cercana a los lenguajes de programación (pseudo-código) y finalmente, codificarlos en algún lenguaje de programación (e.g. Java, Python). Se define una secuencia lineal de actividades, debido a que el código resultante debería ser generado automáticamente desde los modelos anteriores.

La metodología consta de las siguientes etapas, como es ilustrado en la Figura 1.

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.

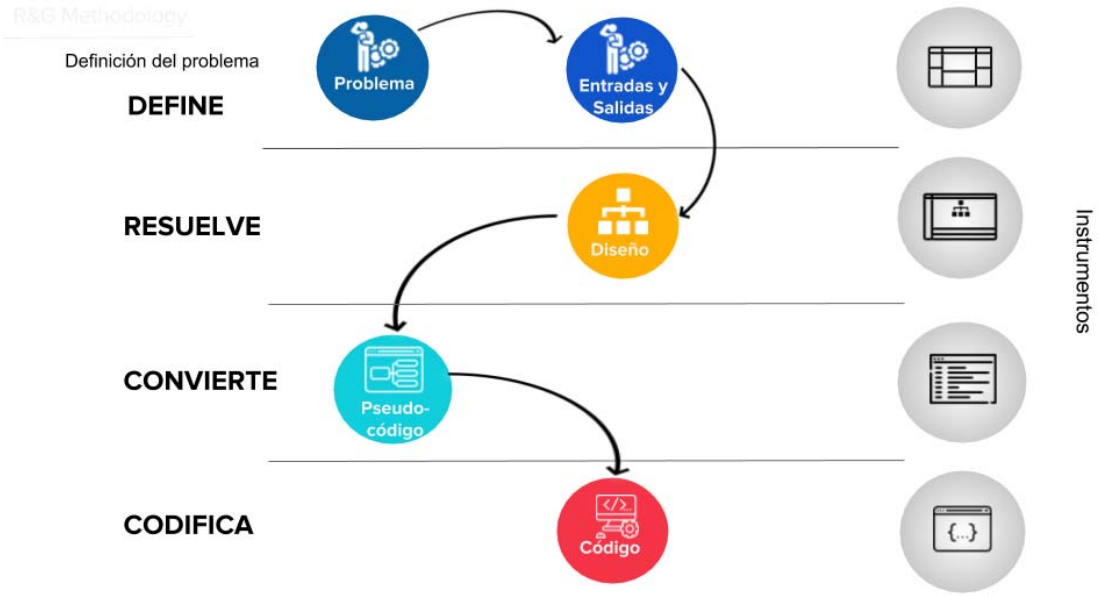

Figure 1: Etapas de Metodología Rápida & Global (R&G).

1) **Definición del Problema** El primer paso que establece la metodología consiste en la identificación y descripción del problema. Se responde a la preguntas (i) ¿Cuál es el problema?, (ii) ¿Cuáles son la entrada?, (iii) ¿Cuáles son sus salidas? y (iv) ¿Cuáles son las principales consideraciones que requiere el problema.
Esta paso obliga al diseñador a entender y describir, en palabras propias, cuál es el problema que se quiere solucionar. Uno de los riesgos importantes en el desarrollo de software es resolver un problema equivocado [19]. Por ello, esta etapa es crítica. Una vez identificado el problema, el diseñador debe identificar las principales condiciones que se han de considerar en la solución propuesta y las principales actividades que se desarrollan (Desarrollo). Por ejemplo en el caso de programación de computadores, se identifican las principales condiciones y repeticiones que un programa de computador debería llevar a cabo. Se determina, además, el posible reuso de funcionalidades, las cuales han sido previamente implementadas y que pueden ayudar a la solución. Estas soluciones ya implementadas, son denominadas funciones; procedimientos; subprogramas o métodos, en el ámbito de programación. Finalmente, se bosqueja de manera preliminar, un diseño de la pantalla la cual será presentada por un computador, tanto para el ingreso de datos (**Entrada**), como para la impresión de resultados (**Salida**). Todas los pasos de la metodología son apoyados con templates predefinidos llamados R&G-Canvas. En este caso es R&G-Canvas-Problema. La Figura 2 presenta una implementación del template R&G-Canvas-Problema implementado utilizando la aplicación draw.io (http://www.draw.io) - también existe una versión en papel - en que se distinguen diversas zonas, siendo las principales la descripción del problema y la definición de entradas y salidas.

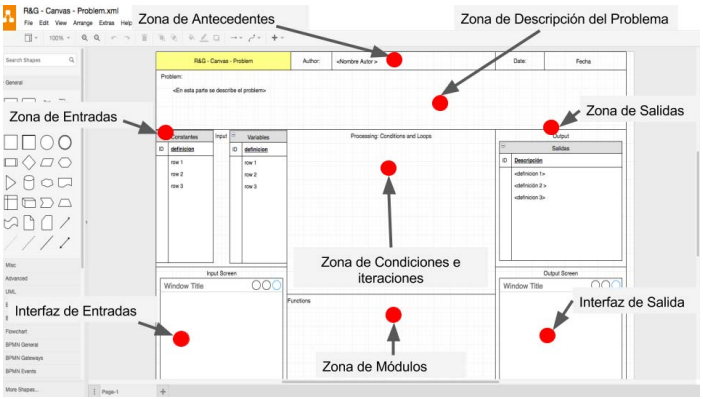

Figure 2: R&G-Canvas-Problema.

2) **Definición de Entradas y Salidas**. Considerando que R&G considera un programa como un proceso, la identificación de entradas y salidas del proceso es un paso clave. En este paso - que se realiza en conjunto con la definición del problema - tiene por objeto identificar las entradas y salidas del proceso. Tanto las entradas como las salidas se representan utilizando nombres y proveyendo su definición en tablas ad-hoc para su descripción. Se define además el tipo de dato que contiene (e.g. entero, booleano y flotante) y si su valor es variable o constante. Por ejemplo: ”N, INTEGER, ”Valor entero que representa el número de elementos”). Esta actividad se documenta en el template R&G-Canvas-Problema, ilustrado en la Figura 2.
3) **Diseño - Refinamiento Conceptual**. Es la tercera

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.

etapa más relevante de la metodología R&G. En ésta se utilizan los principios de abstracción, refinamiento sucesivo y modularidad, y tiene como propósito identificar tareas individuales, denominadas **actividades**. El fundamento de este paso se sustenta en la premisa que "nada es particularmente dificil si se divide en pequeños tareas". Utilizando el principio de diseño de descomposición por refinamiento sucesivo, se divide el problema (actividad principal) en 3 sub-actividades (Entrada, Desarrollo y Salida) y éstas a su vez en sub-actividades más pequeñas hasta lograr representar la actividad de más bajo nivel, llamadas **sentencias**, que son aquellas que un computador ejecuta. En términos de programación las sentencias toman la forma de asignaciones de valores o llamadas a subprogramas. Finalmente, el diseño resultante tiene la estructura de un árbol que comprende padres e hijos.

La Figura 3 ilustra el Canvas de Diseño - llamado R&G-Canvas-Diseño - utilizado para representar la solución enforma gráfica.

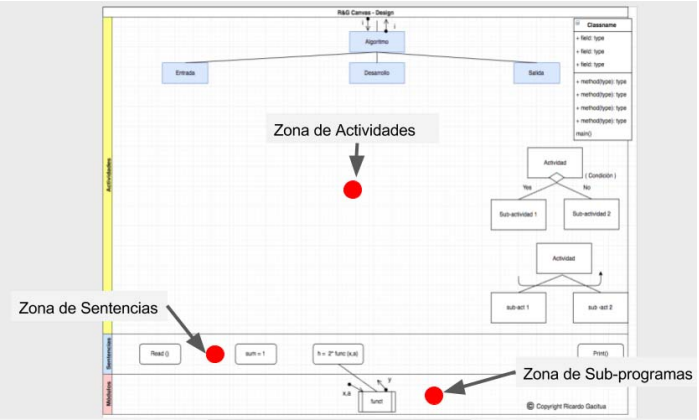

Figure 3: R&G-Canvas-Diseño.

La Figura-4 ilustra además, la notación gráfica utilizada.

4) **Pseudo-código**. Esta etapa concuerda plenamente con los pasos definidos por Parnas en la sección II, en términos de definición de reglas. En este caso, se aplica una regla simple para transformar el árbol generado en el paso de diseño a su equivalente en pseudo-código, la que es definida como sigue:

Result: Recorrer el árbol en in-orden
inOrder(TreeNodeT n){
if n *!= null* **then**
    inOrder(n.getLeft());
    visit(n);
    inOrder(n.getRight());
end
}

Algorithm 1: Desde el diseño al Pseudo-código

Donde *visit(n)* tiene la siguiente condición. Si el nodo es una actividad, su descripción se convierte en comentario en el pseudo-codigo. Si es una sentencia,

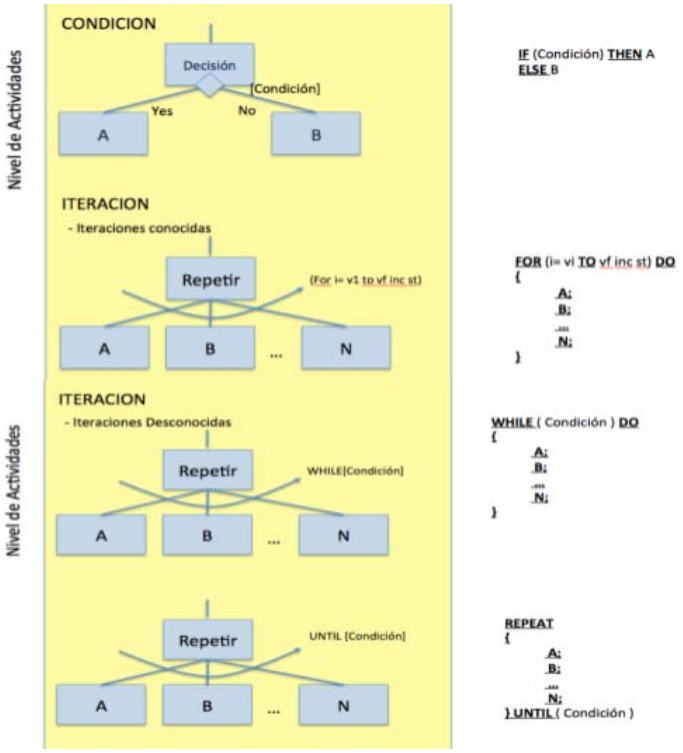

Figure 4: R&G- Notación.

se escribe directamente en el pseudo-codigo. Cada vez que se baja de nivel en el recorrido del árbol, se indenta el código en el pseudo-código. Cuando se encuentra una actividad que representa una estructura de control, ésta se escribe como su equivalente en pseudo-codigo, cómo se muestra en la Figura 5. Esta regla se transformación, muestra que este proceso es completamente automatizable.

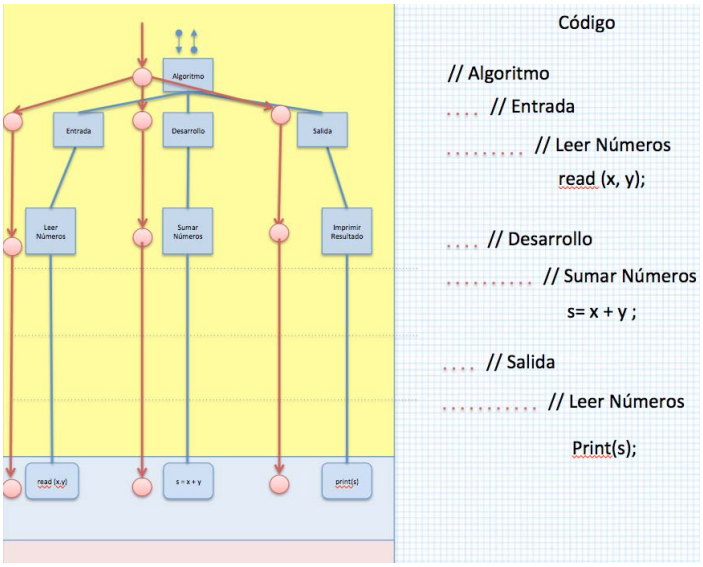

Figure 5: Ejemplo de Generación de Pseudocódigo en R&G.

5) **Codificación**

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.

Una vez generado el pseudo-codigo se escribe (genera) su equivalente escrito en algún lenguaje seleccionado, tal como Java, Phyton, C++ u otro. Al igual que la fse anterior, esta etapa es completamente automatizable.

IV. Resultados

Esta sección presenta los resultados obtenidos de la aplicación de instrumentos de medición para evaluar la aceptación e impacto que tiene la metodología en el proceso de enseñanza de programación (No se incluye las relaciones entre el uso de la metodología y el rendimiento académico, puesto que en un primer curso de Universidad, existen múltiples factores que intervienen en el rendimiento académico, tales como aspectos socio-económicos, psicológicos, entre otros, que escapan del ámbito definido para esta investigación). Los instrumentos de evaluación utilizados en este estudio están basados en el modelo de evaluación de entrenamiento propuesto por Kirkpatrick [20]. Este modelo define 4 niveles de evaluación:

- Nivel 1: Reacción/Motivación. Se refiere al grado en el cual los participantes reaccionan favorablemente al entrenamiento. Se evalúa en base al compromiso de los participantes, contribución a la experiencia de aprendizaje y la habilidad para usar o aplicar el aprendizaje en sus labores.
- Nivel 2: Aprendizaje. Se refiere al grado en el cual los participantes adquieren el conocimiento, habilidades y actitudes, confianza y compromiso durante el entrenamiento. Se pretende medir si los participantes evidencian un incremento en el conocimiento, habilidades o actitudes.
- Nivel 3: Desempeño/Comportamiento. Esto dice relación con el grado en el cual los participantes aplican lo que ellos aprendieron. Se pretende observar si los participantes cambian su comportamiento de regreso en sus lugares de trabajo como resultado del entrenamiento. Se mide basado en la observación.
- Nivel 4: Resultados. El grado en el cual las salidas objetivas ocurren y se refuerzan. Esto se refiere a preguntar si el entrenamiento ha afectado otros procesos o salidas como incremento de productidad, calidad, entre otros. Se pretende medir el impacto de lo aprendido en otras aplicaciones, basado en resultados tangibles , tales como reducción de costos, eficiencia, productividad, retención o incremento de satisfacción.

Para evaluar los resultados de la aplicación de la metodología y visualizar su impacto en la práctica de programación, se utilizó la metodología en un primer curso de programación en la Universidad de La Frontera. Basado en el modelo de Patrick se aplicaron encuestas para medir el Nivel 1, las que dieron cuenta de la evaluación que realizan los estudiantes al final del periodo. Para evaluar el Nivel 2, referente al aprendizaje, se consideró el nivel de complejidad de los problemas que debieron resolver los estudiantes y las encuestas finales del curso que realiza la Universidad. Dicha encuesta contempla, entre otros aspectos, la evaluación del contenido de la asignatura, del desempeño de los profesores, material docente y aspectos relevantes que los alumnos deseen destacar, tales como sugerencias de mejoras, críticas, entre otros. Adicionalmente, se crearon videos con evidencia de su opiniones para evaluar el Nivel 3. Considerando sólo el tiempo de 16 semanas de clases, no fue posible evaluar el Nivel 4 del modelo, pues se requiere hacer un seguimiento de los alumnos en otras asignaturas para determinar el impacto que ha tenido el uso de este enfoque metodológico en su desempeño futuro. la Tabla I presenta un resumen de los instrumentos de evaluación utilizados.

Table I: Instrumento de Medición por Nivel, de acuerdo a [20]

No.	Nivel	Instrumento de Evaluación
1	Reacción / Motivación	Encuesta ad-hoc
2	Aprendizaje	Encuesta Universidad
3	Desempeño / Comportamiento	Video
4	Resultados	-

El curso comprendió 23 estudiantes. Para recoger las percepciones de los estudiantes, en lo referente al impacto de la metodología, se utilizaron encuestas previamente diseñadas por la universidad y, adicionalmente, se diseñó un instrumento de consulta ad-hoc que consta de 7 preguntas respecto de la experiencia de los estudiantes. Fueron grabados además, videos de sus experiencias, tal como se presenta en la Figura 9. Cada estudiante respondió de acuerdo al grado de acuerdo con la afirmación presentada. En la Tabla II se presentan las consultas realizadas y las respuestas de los estudiantes. Las respuestas de los estudiantes fueron grabadas en video, para ser posteriormente analizadas. En la Figura 9 se muestra un cuadro resumen de algunos de los videos realizados.

A. Evaluación de Nivel 1: Reacción / Motivación

Esta evaluación tiene por objeto medir cómo los participantes reaccionan al uso de la metodología. Para ello, se aplicaron 2 encuestas, una general de la universidad y otra ad-hoc para valorar la motivación de la asignatura y de la motivación con respecto del foco en el diseño de soluciones de la metodologia, la cual se evalúa en el Nivel 2. Primeramente, la Figura 6 presenta los resultados de la encuesta aplicada por la Universidad respecto del item motivación de la asignatura.

La encuesta establece una valoración de 0 (mínimo) a 5 (máximo), en relación a la motivación. En la Figura 6 se aprecia que el valor más bajo (3.57) se obtuvo en el item ”en esta asignatura se estimula a los alumnos a participar de las discusiones”. Esto se debe a que un primer curso de programación los alumnos no tiene muchas oportunidades para discutir respecto de una temática en particular, pues el método de enseñanza está basado en enseñar algunos conceptos y poner en práctica la metodología inmediatamente. En efecto, se proponen enunciados de problemas y los alumnos deben seguir la metodología para resolver el problema. Tanto el profesor como los ayudantes están para resolver dudas, guiarles, ayudarles en el análisis del problema y en la codificación posteriormente. Se reconoce, sin embargo, que los problemas entregados son apropiados y desafiantes, siendo la motivación del profesor el item que obtiene la mayor valoración (4.31).

Considerando que la encuesta de la universidad no considera los aspectos particulares de evaluación de la

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.

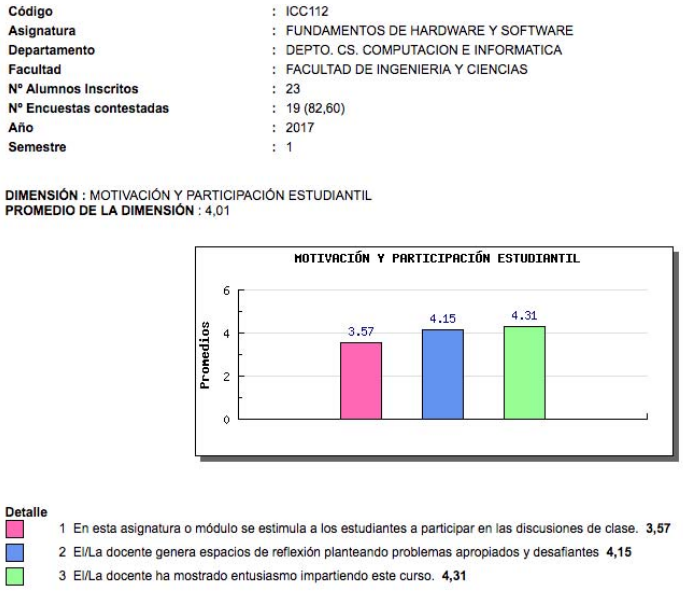

Figure 6: Motivación / Reacción a la enseñanza basada en una metodología (Encuesta Universidad de La Frontera).

metodología R&G, los alumnos de igual forma incluyeron comentarios. Este hecho es significativo, puesto que constituye evidencia de que algún nivel de impacto en los estudiantes fue logrado, como se ilustra en la Figura 7.

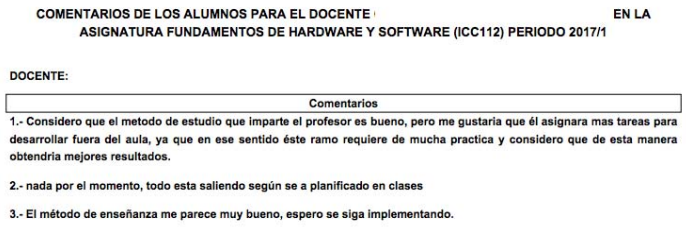

Figure 7: Comentarios incluidos en la encuesta general de la Universidad.

B. Evaluación de Nivel 2: Aprendizaje

Esta evaluación tiene por objeto la medir el incremento de conocimiento (antes y después). Se presentan los resultados del Nivel 2: Aprendizaje. La Universidad de La Frontera utiliza un instrumento de medición del logro de aprendizaje, el cual es utilizado al final del periodo de clases lectivas. La encuesta establece una valoración de 0 (mínimo) a 5 (máximo), en relación al logro de aprendizaje. La Figura 8 muestra el resultado final del curso, donde destaca el valor de 4.42 en el item ”he aprendido cosas que considero importantes para mi profesión”, seguida de un valor 4.21 relativa a la opción ”En esta asignatura he desarrollado habilidades y competencias que pueden ser útiles cuando trabaje”. Todos los itemes evaluados superan los 3 puntos.

Adicional a lo anterior, se aplicó una encuesta ad-hoc para evaluar la metodología, la que contempla siete preguntas respondidas utilizando sólo tres opciones: (a) En desacuerdo,

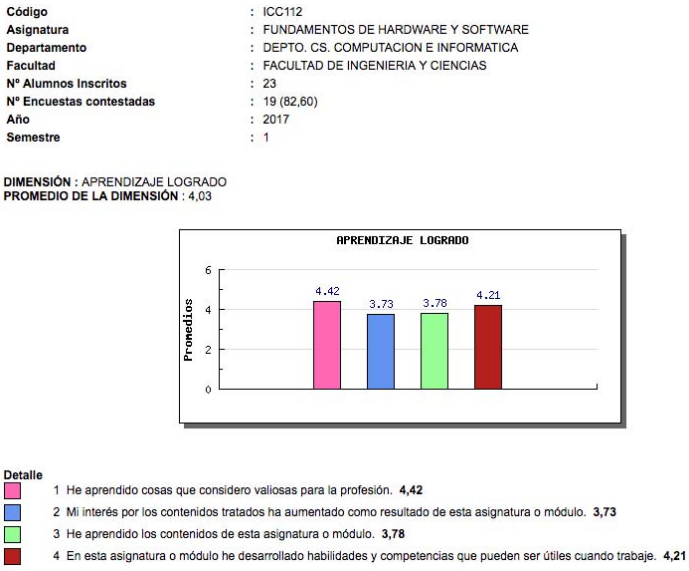

Figure 8: Logros de aprendizajes (Encuesta Universidad de La Frontera).

(b) Ni en desacuerdo / Ni acuerdo, y (c) De acuerdo. La Tabla II presenta los resultados de la encuesta en porcentajes. De un total de 23 alumnos inscritos, sólo 19 respondieron la encuesta al finalizar el curso (82,6%).

Table II: Resultados de Encuesta ad-hoc

ID	Questions	Des - acuerdo	Ni des-acuerdo, Ni acuerdo	Acuerdo
Q1	¿La complejidad del problema es una factor relevante a considerar al resolver un problema?	0%	0%	100%
Q2	¿Puedo escribir código directamente bastando un breve análisis del problema?	0%	10%	90%
Q3	¿El diseño de una solución es fundamental para resolver problemas?	0%	0%	100%
Q4	¿La Metodología R&G es simple y fácil de utilizar?	0%	0%	100%
Q5	¿La Metodología R&G me guía y facilita el análisis de un problema?	0%	0%	100%
Q6	¿En la Metodología R& el programa de computador se deriva completamente del diseño del problema?	0%	0%	100%
Q7	¿La Metodología R&G ha ayudado a mejorar mis prácticas de programación?	0%	5%	95%

La Tabla II muestra que las preguntas Q1,Q3,Q4,Q5 y Q6 alcanzan el 100% de acuerdo. Esto es, los alumnos están de acuerdo en dichas aseveraciones como resultado de la práctica y evaluación que hacen en base a su experiencia. Sólo en las preguntas Q2 y Q7 no existe un total acuerdo. Lo anterior, puede ser el resultado de que algunos estudiantes que cursan la asignatura han tenido experiencia programando previamente y han sido formado centrados en el código. Por tanto, como al inicio del curso se realizan ejercicios muy simples, éstos parecen inquietarse por no ir directamente al código. Por ello, manifiestan algunas opiniones tales como, ”la metodología me

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.

hace más lento llegar al código", o "algunas etapas, como la transformación en pseudo-codigo, no me son necesarias". Futuras evaluaciones realizarán separaciones entre alumnos que saben programar y alumnos que no, desde un inicio.

C. Evaluación de Nivel 3: Desempeño / Comportamiento

Esta evaluación tiene por objeto medir la transferencia de conocimiento y su aplicación en el dominio. En este caso, interesa medir el cambio de prácticas y su aplicación en otras áreas. Sin embargo, medir la aplicación de su utilización en otras áreas en que se desenvuelven los estudiantes es una tarea pendiente, pues requiere de un horizonte de tiempo mayor. El cambio de prácticas debería ser evidenciado sobre todo en aquellos alumnos que cursan por segunda vez la asignatura. Interesa evaluar además, cuánta importancia consideran los alumnos que tiene el diseño por sobre la implementación de una solución computacional y, por sobre todo, cuán útil ha sido la metodología utilizada. En este caso, el método usado para evaluar fueron observaciones y entrevistas, las que fueron registradas en video. La Figura 9 presenta una imagen de un conjunto de videos que fueron grabados.

Figure 9: Ejemplo de Videos que registran el proceso de entrevistas.

Se utilizó un conjunto de cinco preguntas con respuesta mixtas las que se listan a continuación:

1) P1. ¿Considera Ud. que programar es una actividad dificil?
2) P2. ¿Es Ud. capaz de escribir código directamente en el computador, independiente del nivel de complejidad?
3) P3. ¿Cuán importante es el diseño de una solución antes de escribirla en un lenguaje de programación?
4) P4. ¿Cómo la Metodología R&G le ayuda a entender un problema y a escribir su solución en un lenguaje de programación?
5) P5.¿Cuánto han mejorado su habilidades de programación con el uso de la Metodología R&G, en términos de documentación de código, estructuración temprana de funcionalidades, orden y sintáxis de lenguaje, entre otros ?

En algunos caso, la primera parte de la pregunta se responde con una afirmación o negación, y posteriormente se agrega una explicación de la respuesta. La segunda parte de la respuesta es utilizada para evidenciar aspectos tales como, (i) uso de terminología computacional, (ii) uso apropiado del contexto de los términos, (iii) conocimiento de la metodología y (iv) cambios de prácticas.

Las respuestas proporcionadas por los alumnos a las preguntas antes mencionadas se registraron en videos y con ello fueron analizadas, lo que permitió extraer conclusiones basadas en observaciones, como se detalla a continuación:

1) Respuesta 1: Se reconoce que la actividad de programación es dificil, y "depende de la complejidad del problema" (100%).
2) Respuesta 2: La mayoría reconoce que no es capaz de escribir código directamente. Salvo 2 alumnos -de un total de 19 -que dicen que pueden hacerlo, dado el nivel bajo de complejidad de los primeros problemas.
3) Respuesta 3: El total de los alumnos manifiesta que el diseño de la solución es muy relevante (100%).
4) Respuesta 4: Todos reconocen que la Metodología R&G ayuda a entender un problema y facilita la codificación (100%).
5) Respuesta 5: De un total de 19 alumnos, sólo 1 menciona que su nivel de prácticas no ha variado sustancialmente, aun cuándo la evidencia mostraba que en un inicio no había documentación de su código, cómo si había al final del curso. De un total de 19 alumnos, 2 de ellos mencionan que la Metodología R&G les hace más lento el desarrollo. Un tercer alumno en particular, menciona que le resulta más simple utilizar los R&G-Canvas en papel, más que utilizar la aplicación online. Esto es lo que en términos coloquiales nosotros llamamos diseño basado en "servilletas".

V. Discusion

De los resultados presentados en la sección IV se obtiene un amplio acuerdo acerca de que la Metodología R&G es fácil de utilizar y que les ayuda en la resolución de problemas. Tanto en Q1 (Escribir código directamente), como en Q7 (Mejoramiento de prácticas de programación) algunas respuestas fueron influenciadas debido a que los alumnos tenían conocimiento previo de programación y de lenguajes de programación.

De la experiencia de evaluación se evidenció empíricamente cuatro aspectos relevantes:

1) Los estudiantes comprenden y utilizan con seguridad terminología computacional, en etapas tempranas, tales como orientación objeto, clases y métodos, abstracción, cohesión y acoplamiento, refinamiento sucesivo, modularidad, entre otros. En efecto, al término del curso -y se evidencia en los videos- el uso de terminología asociada al modelamiento y su signficado es utilizado ampliamente por los estudiantes.
2) Los estudiantes reconocen y comprenden la importancia del diseño (modelado de la solución) en la construcción de soluciones computacionales a medida que aumenta la complejidad. Todos los estudiantes, reconocen la importancia del diseño de una solución. En aquellos casos en que los alumnos sabían programar, consideraban que no era necesario utilizar todos los pasos de la metodología (por ejemplo, el paso de transformación a pseudo-codigo) puesto que el nivel de complejidad de los ejercicios

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.

iniciales era muy bajo. Ellos eran capaces, por tanto, de escribir el código directamente.

3) Los estudiantes se familiarizan en etapas tempranas con modelos de representación de soluciones y rápidamente comprenden y utilizan modelos de clases UML, como paso posterior a la utilización de R&G.
4) Las programas de computador resultantes, escritos en Java, están documentados con comentarios dentro del texto, indentados, fáciles de leer y de modificar.

VI. Conclusiones

Este trabajó describe la primera parte de una propuesta metodológica que ha comenzado ha ser utilizada en los primeros cursos de programación en la Universidad de La frontera, en particular en la carrera de Ingeniería en Informática. Dicha propuesta permite incluir el diseño de una solución computacional como el foco principal de un proceso por sobre el código, el cual puede ser derivado automáticamente del diseño. Está sustentada en una metodología sistemática denominada Metodología Rápida&Global. El enfoque metodológico global es más amplio que la sóla incorporación de una metodología, pues incluye no sólo la utilización de una metodología de enseñanza, sino también considera aspectos tales como nuevo rol del profesor, nuevo rol de ayudantes de la asignatura, material interactivo de enseñanza, herramientas web de ayuda al modelado de soluciones, y generación automática de código. Este artículo muestra que el diseño disciplinado es enseñable y posible, y que procedimientos simples pueden ser significativamente efectivos para entender el problema, modelar una solución, encontrar defectos y mejorar la integridad de una solución computacional. Trabajo futuro está orientado a construir una aplicación en la Web que permita soportar los Canvas de la metodología, tanto como la definición del problema, el diseño de la solución, asi como la generación automática de código. Posteriormente, se proseguirá con los otros aspectos de la propuesta. Se considera, además, la inclusión de esta metodología en otros ambientes culturales para determinar de qué manera aspectos culturales influyen o no en la aceptación de procedimientos disciplinados en la enseñanza de programación.

Agradecimientos

Este trabajo ha sido financiado por el Proyecto de Innovación Docente DID17-0007 de la Universidad de La Frontera, Chile.

References

[1] D. Garlan and M. Shaw, "An introduction to software architecture," Pittsburgh, PA, USA, Tech. Rep., 1994.

[2] D. Garlan, "Software architecture: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 91–101. [Online]. Available: http://doi.acm.org/10.1145/336512.336537

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.

[4] R. Bijvank, W. Wiersema, and C. Köppe, "Software architecture patterns for system administration support," in *Proceedings of the 20th Conference on Pattern Languages of Programs*, ser. PLoP '13. USA: The Hillside Group, 2013, pp. 1:1–1:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2725669.2725671

[5] R. C. Holt, "Software architecture abstraction and aggregation as algebraic manipulations," in *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research*, ser. CASCON '99. IBM Press, 1999, pp. 5–. [Online]. Available: http://dl.acm.org/citation.cfm?id=781995.782000

[6] J. Castro and J. Kramer, "From software requirements to architectures," in *Proceedings of the 23rd International Conference on Software Engineering*, ser. ICSE '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 764–765. [Online]. Available: http://dl.acm.org/citation.cfm?id=381473.381637

[7] H. Mei, "A complementary approach to requirements engineering—software architecture orientation," *SIGSOFT Softw. Eng. Notes*, vol. 25, no. 2, pp. 40–45, Mar. 2000. [Online]. Available: http://doi.acm.org/10.1145/346057.346070

[8] I. Mistrik, R. Bahsoon, P. Eeles, R. Roshandel, and M. Stal, *Relating System Quality and Software Architecture*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014.

[9] S. Fincher, "What are we doing when we teach programming?" in *Frontiers in Education Conference, 1999. FIE '99. 29th Annual*, vol. 1, Nov 1999, pp. 12A4/1–12A4/5 vol.1.

[10] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972. [Online]. Available: http://doi.acm.org/10.1145/361598.361623

[11] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds., *Structured Programming*. London, UK, UK: Academic Press Ltd., 1972.

[12] T. Rentsch, "Object oriented programming," *SIGPLAN Not.*, vol. 17, no. 9, pp. 51–57, Sep. 1982. [Online]. Available: http://doi.acm.org/10.1145/947955.947961

[13] M. Guzdial, "What's the best way to teach computer science to beginners?" *Commun. ACM*, vol. 58, no. 2, pp. 12–13, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2714488

[14] J. Sweller and G. A. Cooper, "The use of worked examples as a substitute for problem solving in learning algebra," *Cognition and Instruction*, vol. 2, no. 1, pp. 59–89, 1985.

[15] C. Cachero, M. De-Juan-Vigaray, E. González Gascón, and P. Barra Hernández, "Intention to adopt scratch as a teaching tool: An empirical study," in *INTED2017 Proceedings*, ser. 11th International Technology, Education and Development Conference. IATED, 6-8 March, 2017 2017, pp. 9005–9013. [Online]. Available: http://dx.doi.org/10.21125/inted.2017.2130

[16] D. L. Parnas, "Risks of undisciplined development," *Commun. ACM*, vol. 53, no. 10, pp. 25–27, Oct. 2010. [Online]. Available: http://doi.acm.org/10.1145/1831407.1831419

[17] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, ser. SFCS '77. Washington, DC, USA: IEEE Computer Society, 1977, pp. 46–57. [Online]. Available: http://dx.doi.org/10.1109/SFCS.1977.32

[18] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM Comput. Surv.*, vol. 41, no. 4, pp. 19:1–19:36, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1592434.1592436

[19] B. W. Boehm, "Software risk management: Principles and practices," *IEEE Softw.*, vol. 8, no. 1, pp. 32–41, Jan. 1991. [Online]. Available: http://dx.doi.org/10.1109/52.62930

[20] D. L. Kirkpatrick, *Evaluating training programs*. Tata McGraw-Hill Education, 1975.

Authorized licensed use limited to: University of Canberra. Downloaded on March 28,2025 at 05:27:51 UTC from IEEE Xplore. Restrictions apply.