EL ROL DE LA VISUALIZACIÓN Y LA INTELIGENCIA ARTIFICIAL EN LA INGENIERÍA DE SOFTWARE

ALEXANDRE BERGEL

Profesor Asociado del Departamento de Ciencias de la Computación de la Universidad de Chile. Doctor en Ciencias de la Computación por la Universidad de Bern, Suiza. Sus intereses de investigación se centran en ingeniería de software, eficiencia del software, visualización, ambientes de programación y aprendizaje de máquina.
abergel@dcc.uchile.cl

DIFICULTAD DE HACER SOFTWARE

"Los programas informáticos son las construcciones más complejas que el ser humano produce", dijo Douglas Crockford, creador del lenguaje de programación JavaScript. Las dificultades de diseñar y construir software son bien conocidas por ambos: desarrolladores y usuarios. Por ejemplo, es frecuente encontrar fallas en el funcionamiento de un sitio web o enfrentar un consumo excesivo de memoria en aplicaciones de dispositivos móviles. Estas malas experiencias que cada uno ve de forma casi cotidiana, reflejan un hecho: la construcción de software puede ser el campo de la ingeniería que el ser humano menos domina. El humano está mejor preparado para construir edificios o barcos que para desarrollar software.

Existen varias causas que explican la pobre calidad del software. En general, un desarrollador de software escribe código usando herramientas muy similares a las que usa un escritor para redactar un documento de texto, como puede ser un libro. Los ambientes de desarrollo de software (IDEs) se parecen mucho a Microsoft Word, excepto que incluyen algunas mejoras. Un desarrollador tiene una visión muy restringida del software, que es lo que muestra la ventana de este "Word". Para asegurar una buena calidad de software, es importante ofrecer una visión global de éste, la que puede además ser muy grande.

Las dificultades de producir software robusto son bien conocidas y la comunidad científica en ingeniería de software invierte mucha de su energía en mejorar las técnicas y herramientas involucradas en el proceso de construcción de software. Mi esfuerzo personal se distingue del esfuerzo del resto de la comunidad, en que se basa el uso de técnicas de visualización y de inteligencia artificial.

VISUALIZACIÓN DE SOFTWARE E INTELIGENCIA ARTIFICIAL

La hipótesis sobre la que se sustenta mi trabajo, sostiene que *las herramientas que se usan para construir software pueden ser mejoradas de forma significativa usando técnicas de visualización y de inteligencia artificial.*

El grupo de investigación ISCLab —que dirijo— nació en 2018 con el objetivo, precisamente, de entregar a ingenieros y desarrolladores técnicas de visualización y de inteligencia artificial, que faciliten la construcción de software.

Las piezas de software están constituidas y caracterizadas por múltiples datos, en diversos formatos. En particular, el código fuente define el software. Una pieza de software tiene la finalidad de ser ejecutada para exhibir sus funcionalidades. La ejecución puede ser medida a través del tiempo de ejecución de cada uno de sus componentes. El punto a tener presente, es que una pieza de software viene acompañada de muchos datos. Y manejar estos datos es complejo. Usamos técnicas de visualización para ver patrones en el código fuente y en la ejecución. Una vez que identificamos estos patrones, los cuales pueden corresponder a anomalías en el funcionamiento de la aplicación, explotamos técnicas de inteligencia artificial para solucionar o caracterizar dicha anomalía.

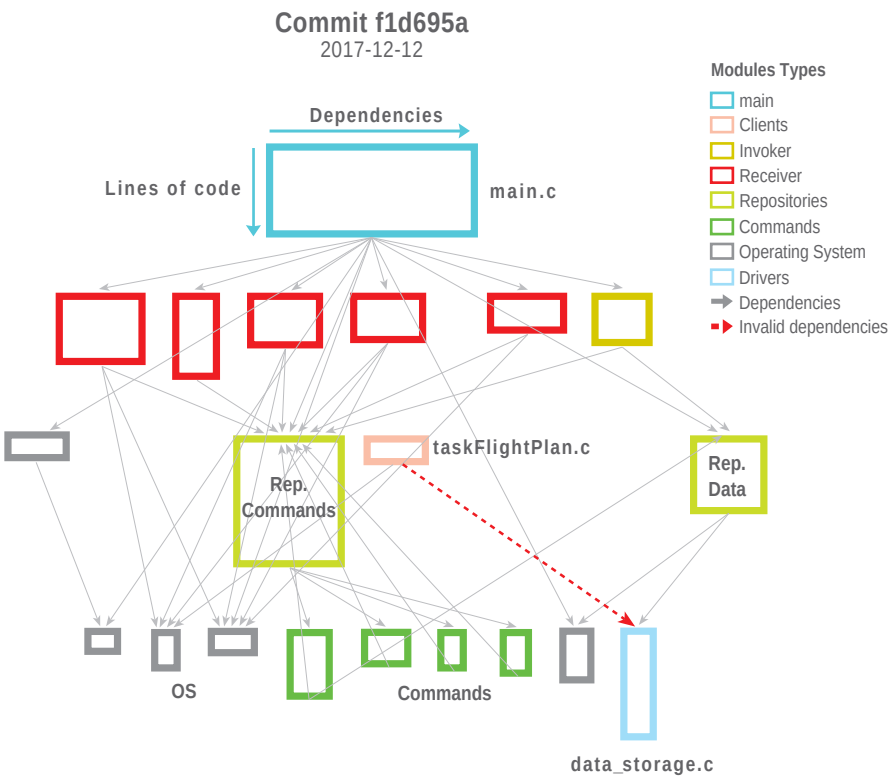

Figura 1. Indica una dependencia incorrecta entre varios componentes del sistema de vuelo de Suchai (imagen extraída del artículo "An Architecture-Tracking Approach to Evaluate a Modular and Extensible Flight Software for CubeSat Nanosatellites", publicado en https://ieeexplore.ieee.org/document/8758807).

VISUALIZACIÓN

Una pieza de software puede ser muy grande. Considere, por ejemplo, el software de vuelo de Suchai, el nanosatélite chileno desarrollado por la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile. El tamaño del sistema de vuelo completo es de 40 mil líneas de código, lo que puede considerarse de tamaño mediano. Teniendo en cuenta

que un libro tiene aproximadamente 35 líneas por página, el sistema de vuelo de Suchai ocuparía 1.142 páginas, lo que corresponde a dos veces el último libro de Harry Potter. Cada palabra y letra de este libro puede ser la fuente de una falla grave, que puede resultar, por ejemplo, en la pérdida del satélite. Una pieza de software de este tamaño tiene que seguir una arquitectura bien definida y documentada. Pero las herramientas de desarrollo de software actualmente usadas, no permiten contrastar la arquitectura deseada del software con la arquitectura realmente implementada. Y no tener claro cuál es la arquitectura de un software, es tan grave como no saber si un edificio responderá o no a un sismo fuerte; cualquier evento no previsto puede tener consecuencias catastróficas.

En ISCLab hemos desarrollado diversos tipos de visualizaciones que permiten apreciar las partes internas de una pieza de software (ver **Figura 1**). Para trazar un paralelismo con, por ejemplo, la medicina, si a una persona le duele una parte de su cuerpo, el médico le indicará tomarse unas radiografías para corroborar los síntomas e indicará así el tratamiento adecuado. En el caso de los sistemas informáticos, estamos siguiendo literalmente la misma metodología.

Hemos desarrollado diversos tipos de visualizaciones que permiten ayudar a los desarrolladores a detectar anomalías en la arquitectura del software. Una pieza de software de gran tamaño, al igual que un edificio, requiere de una arquitectura definida, clara y sin ambigüe-dades. Verificar en papel que una arquitectura de software se adhiere a lo planificado es, lamentablemente, una tarea sumamente difícil, pero, a su vez, indispensable. Nuestras visualizaciones permiten facilitar de forma significativa esta tarea.

ALGORITMO GENÉTICO

Desarrollar software es una actividad sumamente compleja. Frecuentemente, un ingeniero tiene que tomar decisiones difíciles. Un ejemplo típico se da en la labor de testing, la que requiere de un conocimiento profundo del software que se desea testear. Considere, por ejemplo, una pieza de software que tiene como objetivo validar un RUT. Producir un test que sea completo (que no excluya casos probables, pero no previstos por el desarrollador, como dar como *input* un RUT incompleto), es imposible de verificar en general. Para enfrentar estas dificultades en la tarea de testing, usamos técnicas de algoritmos genéticos.

Un algoritmo genético es una metáfora computacional de la evolución biológica de las especies. Considere una población de conejos que viven en un ecosistema con depredadores voraces, como lobos, por ejemplo. Los conejos más rápidos tienen mayor probabilidad de sobrevivir que aquellos más lentos. Al vivir más tiempo, los conejos rápidos tienen, a su vez, mayor probabilidad de reproducirse. Resulta entonces muy probable que la próxima generación de conejos tenga mejores habilidades de supervivencia, al escapar más rápidamente de los depredadores.

Un algoritmo genético opera de forma muy similar. Primero crea una población inicial de individuos, de forma aleatoria. Luego selecciona a los individuos más “fuertes” o “cercanos” a la solución del problema en cuestión. Finalmente crea una nueva generación combinando la información genética de cada individuo, a través de operaciones predefinidas.

Aplicado a la tarea de testing, un individuo puede representar una secuencia de *inputs* que se le dan al programa. Esta secuencia puede ser cualquier valor, elegido de forma aleatoria. Imagínese que al testear el validador de RUT con una secuencia de caracteres aleatoria, éste simplemente se cae. En ese caso hemos detectada una falla grave del programa.

El uso de algoritmos genéticos ayuda a generar tests que permiten identificar fallas que serían significativamente más difíciles de detectar para un humano, debido a los sesgos (*bias*) que tenemos. Por ejemplo, en el caso del validador de RUT sería difícil que un humano generase como *input* un nombre en vez de un RUT. El humano tendería a generar valores que se parecen a un RUT, por ejemplo, números separados mediante un punto. Por el contrario, un algoritmo genético sí sería capaz de generar los inputs que hagan caer al validador.

CHATBOT COMO APOYO

El desarrollo de software requiere de colaboración y discusión entre los distintos individuos involucrados. Por ejemplo, consideremos el desarrollo de un videojuego en 3D. Una situación clásica para un desarrollador es usar una librería que gestione el dibujo 3D. Usar una librería existente permite al desarrollador enfocarse en los aspectos que definen la parte lógica del juego. Una librería 3D, como Unreal o Unity, es compleja de usar. Es muy probable que el desarrollador tenga que buscar documentación en Internet y quizás recurrir a algún chat para obtener respuestas más específicas. En ISCLab hemos desarrollado un chat que responde a preguntas sencillas como “Who is expert in Unity?”. Nuestro chat responde con la lista de expertos, que construye a partir de información recopilada de repositorios de programas, como GitHub.

TRABAJO FUTURO

Es muy probable que en el futuro las piezas de software sean más grandes que en la actualidad. Siempre ha sido así y al día de hoy no hay ningún indicador en la industria del desarrollo de software que señale que esta dinámica cambiará. Al contrario, esta tendencia acentuará la necesidad de medir la calidad de software y de usar un apoyo artificial por parte de los desarrolladores.

Desde un punto de vista académico, el laboratorio ISCLab ha desarrollado técnicas de visualización y de inteligencia artificial. Ahora hemos entrado en una fase de aplicar nuestros resultados a situaciones reales y comprobar si los beneficios de nuestras técnicas —identificados de forma teórica— se corroboran en la práctica, sobre piezas de software grandes y complejas. Actualmente estamos aplicando nuestras técnicas a diferentes sistemas, como el sistema de vuelo de Suchai y sistemas que controlan el comportamiento de robots. ■