Conformidad Estructural de Arquitecturas combinado con Análisis de Impacto de Cambios

Miguel Armentano[3], Luis Soldavini[3], J. Andrés Díaz Pace[1], Santiago Vidal[1], and Claudia Marcos[2]

[1] ISISTAN-CONICET, Argentina
[2] ISISTAN-CIC, Argentina
[3] Facultad de Ciencias Exactas, UNICEN, Argentina

Abstract. La conformidad de arquitecturas de software es una práctica que permite mantener la estructura arquitectónica alineada y consistente con su implementación en código. Entre otros beneficios, este alineamiento permite a los arquitectos y desarrolladores realizar distintos análisis de la solución desde etapas tempranas (por ej., de performance, o de modificabilidad, entre otros). Para esto, deben verificarse periódicamente las relaciones entre los elementos arquitectónicos y sus contrapartes en el código fuente, a fin de detectar posibles violaciones de la arquitectura. Las técnicas y herramientas existentes para conformidad arquitectónica proveen un buen soporte para verificar relaciones de tipo estructural. Sin embargo, ciertos análisis que son útiles a nivel arquitectónico, como es el caso del impacto de cambios (CIA, Change Impact Analysis), son difíciles de verificar en relación al código fuente, lo cuál genera una brecha entre las suposiciones y conclusiones que se hacen a nivel arquitectónica y la implementación actual del sistema. Este trabajo presenta un enfoque que extiende las reglas básicas de conformidad estructural con reglas que permiten contemplar suposiciones de CIA con el fin de validarlas en la implementación de la arquitectura. En particular, se propone una herramienta que integra un CIA en base a escenarios de modificabilidad con el modelo de reflexión de Murphy & Notkin. Los resultados, si bien son preliminares, indican que este enfoque permite identificar distintos tipos de violaciones arquitectónicas.

1 Introducción

Las arquitecturas de software son modelos de alto nivel de sistemas de software que hacen explícitas las principales decisiones de diseño respecto a la satisfacción de ciertos atributos de calidad clave (por ej., performance, disponibilidad, modificabilidad, etc.) [5]. Una arquitectura permite distintos tipos de análisis y revisiones de diseño, para mitigar riesgos del sistema desde etapas tempranas del ciclo de vida. No obstante, para poder capitalizar estos beneficios, el sistema debe implementarse de acuerdo a las decisiones y reglas de la arquitectura.

Se dice que la implementación está en *conformidad* con la arquitectura, si dicha implementación respeta los principios de diseño y restricciones prescriptas por la arquitectura [5, 17]. Por ejemplo, si la arquitectura ha sido diseñada siguiendo un estilo de capas en el cual cada capa k solo puede invocar servicios

provistos por su capa inferior (*k-1*) y ser accedida por funciones de su capa superior (*k+1*), entonces los módulos y clases que materializan dichas capas deben respetar sus restricciones de uso.

Debido a la evolución del sistema, es común que aparezcan discrepancias entre la arquitectura "documentada" (u originalmente diseñada) y su implementación en código [19]. La sucesión de cambios (típicamente, en la implementación) tiende a "erosionar" la arquitectura inicial. Este problema impacta negativamente sobre los análisis y razonamientos que pueden llevarse a cabo a nivel arquitectónico, y si estos no son gestionados adecuadamente, puede perjudicar la calidad del desarrollo y ocasionar serios problemas de mantenimiento en el sistema [18].

Los enfoques y herramientas existentes para análisis de conformidad (CA, por sus siglas en inglés) [6, 20, 9] se basan mayormente en analizar diferencias sobre estructuras estáticas, por ejemplo, contrastar una vista arquitectónica de módulos (y sus relaciones de uso) contra el grafo de clases y dependencias del sistema. En este sentido, el modelo de reflexión de Murphy & Notkin [17] permite detectar convergencias, divergencias, y ausencias en la implementación de un sistema, en relación a su arquitectura de módulos original. Sin embargo, existen ciertos análisis de atributos de calidad (por ej., para performance, o para modificabilidad, entre otros), que si bien son relevantes para asegurar conformidad, no están cubiertos actualmente por las herramientas de conformidad (por ej., LattixDSM[4], SonarJ[5], Structure101[6], o ConQAT[7]) [21, 7] y los desarrolladores deben asegurar dichos atributos en la implementación (por ej. via testing) [10].

En particular, este trabajo está focalizado en el análisis de propiedades de modificabilidad mediante técnicas de impacto de cambios. El cambio constante de los sistemas, ya sea por mejoras o por corrección de errores, hace que sea importante identificar el conjunto de componentes afectados por un requerimiento dado, lo que se conoce como Análisis de Impacto de Cambios [3, 13] (CIA, por sus siglas en inglés). Por ejemplo, si un sistema bancario posee una arquitectura en capas, y se requiere agregar un nuevo mecanismo de ingreso en una capa determinada, este cambio no solo va a afectar al módulo de *Ingreso* sino que muy probablemente la modificación se propague a otros módulos (en la misma o en otras capas). En términos de conformidad arquitectónica, es posible verificar (mediante herramientas como las mencionadas anteriormente) que la implementación adhiere a las reglas estructurales, pero esto no necesariamente sucede con los análisis de tipo CIA. Dado que verificar un CIA de forma manual es generalmente costoso y tedioso, es importante automatizar dicho proceso.

En este contexto, este trabajo explora técnicas de conformidad que extienden los análisis estructurales convencionales entre arquitectura e implementación con un análisis de escenarios de modificabilidad en base a CIA. Específicamente, se presenta una herramienta llamada ArchGuru que permite registrar las suposiciones realizadas por un CIA arquitectonico y evaluarlas a nivel código fuente.

El resto del artículo se estructura de la siguiente manera. En la Sección 2 se presentan los conceptos básicos de CIA y CA. La Sección 3 presenta el enfoque

[4] http://www.lattix.com/
[5] https://www.hello2morrow.com/products/sonarj
[6] https://structure101.com/
[7] https://www.conqat.org/

y la herramienta ArchGuru. En la Sección 4, se presenta el caso de estudio. En la Sección 5, se discuten trabajos relacionados. Finalmente, en la Sección 6, se dan las conclusiones y algunos trabajos futuros.

2 Análisis de Impacto de Cambios y de Conformidad

Para que la arquitectura pueda ser utilizada eficazmente durante el desarrollo del software, la misma es representada utilizando distintos tipos de vistas [4, 11], como (i) vista de módulos; (ii) vista de componentes y conectores, y (iii) vista de deployment, entre otras. Por ejemplo, la vista de módulos divide las funciones del software agrupándolas en unidades de código llamadas *módulos*. Cada módulo representa las responsabilidades que el sistema debe cumplir de acuerdo a la especificación de requerimientos. Las relaciones que se establecen entre estos módulos pueden ser de uso, de descomposición, y de jerarquía.

Adicionalmente, una arquitectura debe cumplir con determinados atributos de calidad tales como performance, seguridad, mantenibilidad, o modificabilidad. Estos atributos pueden expresarse a través de escenarios de calidad. Por ejemplo, un escenario de modificabilidad especifica un tipo de cambio que el sistema debe contemplar. De esta forma, el escenario puede estimar la capacidad de modificabilidad de un sistema (para un cambio dado) en términos del tiempo y/o esfuerzo que demandará implementar dicho cambio. Un escenario podria especificar cambios en la interfaz gráfica como: "Un desarrollador desea cambiar el algoritmo de layout con el cual se muestra un gráfico. Este cambio deberá realizarse en el código, en tiempo de (re-)diseño de la solución. El cambio deberá tomar menos de una hora para aplicarse y testearse, y no deberán producirse efectos secundarios en el comportamiento de la aplicación".

Para propiedades de modificabilidad, los modelos de conformidad arquitectónica y CIA son útiles para analizar la calidad (o "salud") de una arquitectura con respecto a su implementación, según se explica a continuación.

2.1 Análisis de Conformidad (CA)

El CA de una arquitectura dada permite determinar el grado en que la implementación es consistente con los documentos de arquitectura.

De esta forma, se asegura que los desarrolladores implementen un sistema basándose en lo que esta definido en la arquitectura (por ej., módulos, interfaces, responsabilidades, y dependencias permitidas entre dichos módulos).

Para facilitar el CA, Murphy & Notkin desarrollaron el concepto de *modelos de reflexión* [17]. Este enfoque permite definir un modelo de alto nivel de referencia para la estructura del sistema (por ej., una vista de módulos) que captura las reglas estructurales que deben cumplirse en un segundo modelo (de mas bajo nivel), que generalmente se corresponde con la implementación del sistema. Cuando analizan las diferencias entre el modelo de alto nivel y el model de bajo nivel, existen 3 tipos de resultados que permiten analizar la conformidad:

- Convergencia: es una relación permitida entre dos elementos del modelo de alto nivel (por ej., una relación de uso entre dos módulos A y B de la arquitectura), que se encuentra efectivamente implementada en el modelo de bajo

nivel (por ej., relaciones de dependencia entre las clases Java que se corresponden con los módulos A y B). Los casos de convergencia no representan problemas, desde la perspectiva de CA.
- Divergencia: es una relación no permitida entre dos elementos del modelo de alto nivel (por ej., no existen relaciones de uso entre dos módulos A y B de la arquitectura), que sin embargo aparece en el modelo de bajo nivel (por ej., las clases que se corresponden con A y B posee dependencias de uso entre ellas). Los casos de divergencia se consideran errores (o violaciones), desde la perspectiva de CA.
- Ausencia: es una relación entre dos elementos del modelo de alto nivel (por ej., una relación de uso entre dos módulos A y B de la arquitectura) que, sin embargo, no esta todavía reflejada en el modelo de bajo nivel (por ej., existen clases que se corresponden con los módulos A y B, pero dichas clases no evidencian dependencias entre ellas). Los casos de ausencia también se consideran un problema , aunque no tan grave como las divergencias.

Un pre-requisito para analizar conformidad es la existencia de mapeos entre los elementos de la arquitectura (por ej., módulos, responsabilidades, escenarios) y los elementos de la implementación (por ej., paquetes, clases, métodos). Estos mapeos pueden ser definidos por expertos, o basarse en convenciones de implementación del sistema.

2.2 Análisis de Impacto de Cambio (CIA)

El CIA fue definido por Bohner como "el proceso que predice y determina qué partes de un sistema de software pueden ser afectados por un cambio y cuáles son sus potenciales consecuencias" [3]. El CIA es un enfoque sistemático para comprender los efectos de un tipo de cambio, en términos de: (i) identificar las partes que deben volver a testearse, (ii) mejorar la estimación de tiempo, y esfuerzo que se requiere para el mantenimiento, y (iii) reducir los errores potenciales debido a impactos del cambio no evidentes.

Naturalmente, el CIA puede aplicarse sobre una vista arquitectónica de módulos, partiendo de un escenario de modificabilidad y estimando luego la propagación de los cambios expresados por el escenario sobre los distintos módulos. Más aún, puede utilizarse una métrica de costo que cuantifique el impacto de dicho escenario sobre el sistema.

3 ArchGuru

Si bien los modelos de reflexión permiten analizar la conformidad del código fuente respecto a la arquitectura, no contemplan aspectos de (i) complejidad de la implementación de los módulos, y (ii) grado de acoplamiento entre las clases que implementan pares de módulos. Estos dos aspectos son vitales para asegurar que una estimación a nivel arquitectural de un CIA se preserva en la implementación. Por ejemplo, un enfoque basado en modelos de reflexión puede arrojar como resultado que un sistema no posee divergencias ni ausencias respecto a la arquitectura original, pero aún así pueden existir problemas de erosión en el código. En otras palabras, si un CIA para un escenario dado M se basa en la suposición de que un módulo X tiene una complejidad media, el

código de las clases que materializan dicho módulo debiera corresponderse con dicha suposición. Alternativamente, si un CIA para el mismo escenario M asume que el acoplamiento entre dos módulos A y B es bajo (y por ende, el cambio es poco factible que se propague entre ambos módulos), debiera suceder que las clases que implementan A y B no posean un alto nivel de acoplamiento.

En esta línea, este trabajo presenta un enfoque llamado ArchGuru que permite extender los modelos de reflexión de Murphy & Notkin con la capacidad de identificar problemas de erosión arquitectónica. Este enfoque fue implementado como un plugin del ambiente Eclipse.

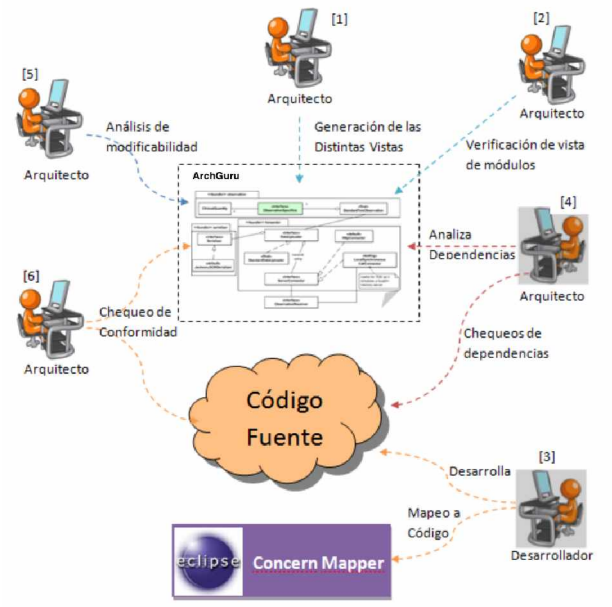

Fig. 1: Enfoque ArchGuru

La Fig. 1 presenta el enfoque ArchGuru con sus principales actores y actividades. El flujo de trabajo comienza a partir del modelado de 2 tipos de información: (i) vista de escenarios, y (ii) vista de módulos del sistema (1). Una vez definida la vista de módulos, el arquitecto puede realizar una validación inicial de la misma (2). Luego, el desarrollador deberá mapear los módulos indicados en la vista de módulos al código fuente que los implementa (3). Después de realizar el mapeo de código, es posible hacer un primer CA en base al modelo de reflexión, a fin de detectar divergencias y ausencias (4) – esta verificación se denomina *conformidad estructural*. En caso de existir violaciones estructurales, éstas deben subsanarse.

Por otro lado, una vez construida la vista de módulos, puede realizarse el CIA sobre dicha vista (5). Este análisis puede realizarse en cualquier momento, y solo es necesario contar con los escenarios de modificabilidad a explorar y vincularlos con las responsabilidades de los módulos. Un punto importante del CIA es que genera un *conjunto de suposiciones* sobre la complejidad de los módulos y su grado de acoplamiento. Por último, el arquitecto puede analizar la

conformidad entre dichas suposiciones y la implementación actual del sistema (6) – esta verificación se denomina *conformidad de propiedades de modificabilidad*, y es la principal contribución de ArchGuru. En caso de encontrarse violaciones relacionadas con modificabilidad, éstas deben ser subsanadas.

En las siguientes sub-secciones se dan detalles de las partes del enfoque.

3.1 Escenarios de Modificabilidad

La vista de escenarios de modificabilidad del enfoque está basada en el template propuesto por el SEI [5] para la definición de los mismos. La Fig. 2 presenta un posible escenario de modificabilidad que describe el cambio de dimensiones de un tablero para un juego de Sudoku.

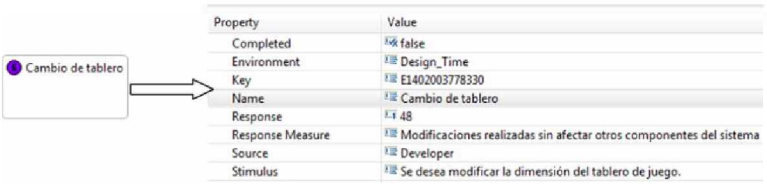

Fig. 2: Vista de escenario de modificabilidad

A partir de esta vista, será posible ejecutar el algoritmo de CIA sobre la vista de módulos para estimar (y de ser posible, cuantificar) los efectos para cumplir con uno o más escenarios.

3.2 Vista de Módulos

La vista de módulos del enfoque está compuesta por diversos elementos (Fig. 3):

- Módulo: es una unidad de implementación que comprende un conjunto de responsabilidades. Un módulo puede implementar interfaces a otros módulos.
- Responsabilidad: pertenecen a los módulos y son capturadas como requerimientos internos del mismo. Cada responsabilidad tiene una descripción.
- Interface: es un módulo *abstracto*, que define un conjunto de operaciones (pero no proporciona su implementación). Cada operación debiera tener correspondencia con una de las responsabilidades del módulo que la implementa.
- Operación: pertenecen a las interfaces y son capturadas como requerimientos internos de las mismas.
- Relaciones de dependencia/uso: denota que un módulo requiere la correcta implementación de otro módulo para su correcto funcionamiento. Los módulos pueden comunicarse directamente entre sí, o por medio de sus interfaces.

Adicionalmente, por cada módulo, el arquitecto deberá asignar una propiedad denominada "Complexity Index" utilizando una vista de propiedades provista por el plugin (Fig. 4). Este valor indica la complejidad en una escala simple, de la percepción que posee el arquitecto sobre la complejidad esperada de dicho módulo. Los valores que puede tomar "Complexity Index" se encuentran definidos por una escala ordinal: Very Easy, Easy, Normal, Difficult y Very Difficult. Esta propiedad será utilizada posteriormente durante el CIA, tanto a nivel arquitectónico como de implementación. Cada valor de la escala se representa

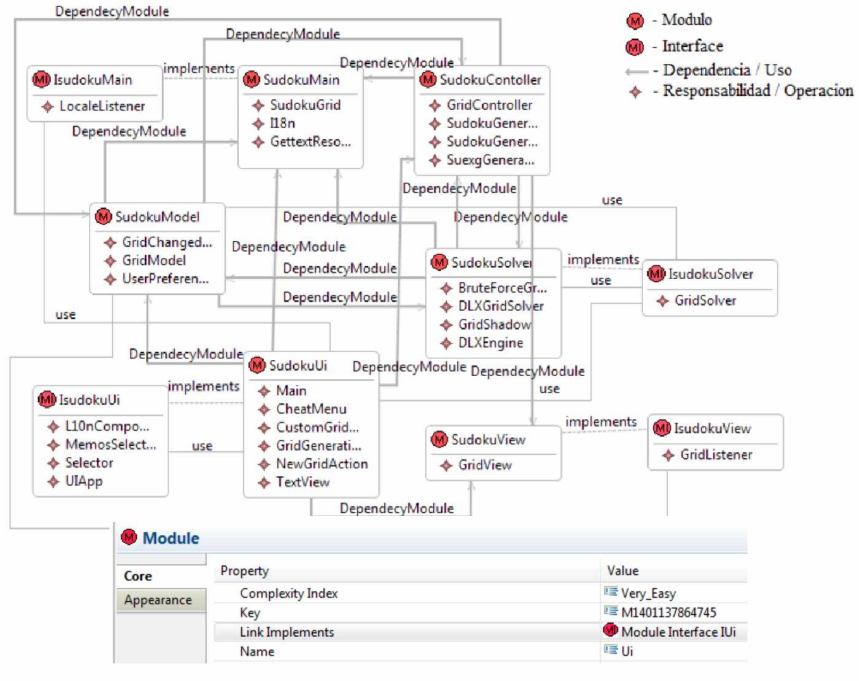

Fig. 4: Vista de propiedades de módulos

con 3 métricas subyacentes. Para cada métrica, se define un umbral que la implementación del módulo no deberia superar. Estos umbrales son configurados por un arquitecto, dado que son propios de cada sistema. Específicamente, se utilizaron métricas de acoplamiento de un módulo, líneas de código por módulo, y números de métodos que forman parte de un módulo.

El acoplamiento de un módulo hace referencia al número de dependencias existentes entre las clases de un módulo con respecto a las clases que conforman el resto de los módulos. Por ejemplo, si una clase A (perteneciente al módulo Y) tiene dependencias con dos clases B y C (pertenecientes a un módulo X), el acoplamiento del módulo Y es 2. Las líneas de código por módulo se calculan contando la cantidad de líneas que se encuentran dentro de cada clase perteneciente a un módulo. Por ejemplo, si un módulo tiene dos clases (A y B) mapeados, la primera con 200 líneas de código y la segunda con 350 líneas de código, la cantidad de líneas de código del módulo va a ser 550. El número de métodos hace referencia a la suma de todos los métodos que poseen las clases mapeadas dentro de un módulo. Por ejemplo, si un módulo mapea a las clases A (con 30 métodos) y B (con 25 métodos), el número de métodos del módulo es de 55.

3.3 Análisis de Impacto de Cambio (a nivel arquitectónico)

Una vez creadas las vistas de escenarios y módulos, ArchGuru permite realizar un CIA para cada escenario. Para ello, se deben especificar las responsabilidades (y/o módulos) iniciales que involucra un escenario M, y luego CIA automáticamente ejecuta un proceso de propagación a través de los elementos de la vista de módulos, para finalmente retornar el conjunto de módulos afectados directa e indirectamente por el cambio. Se implementó un algoritmo de propagación de cambios probabilístico, en base a los lineamientos de [13].

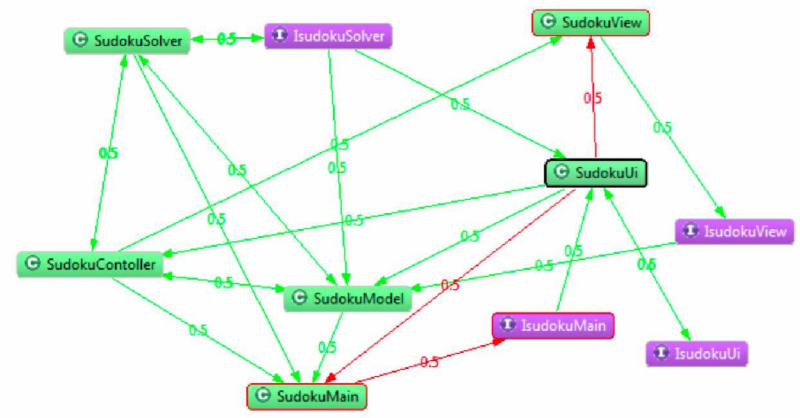

Fig. 5: Visualización algoritmo CIA

ArchGuru permite visualizar la ejecución del algoritmo CIA mediante un grafo, cuyos nodos corresponden a los módulos e interfaces y sus arcos representan las dependencias entre módulos e interfaces (Fig. 5). Además de estimar el número de módulos afectados por un escenario de modificabilidad, es posible estimar el costo total del cambio en base a costos individuales de módulos. Si el costo deseado en el escenario (*response measure* según la terminología del SEI) es mayor que el costo total calculado por el algoritmo de CIA, el escenario se marca como "satisfecho", sino se marca como "insatisfecho". En este punto, el algoritmo registra en forma de suposiciones (denominadas "assumptions" en ArchGuru) el costo de cada módulo (o interface) y la probabilidad de propagación (o grado de acoplamiento) entre cada par de módulos, que serán utilizadas luego para extender el proceso de conformidad estructural. Estas suposiciones determinan el contexto bajo el cual se realizó el CIA para el escenario de modificabilidad analizado. Si este contexto cambiase, es necesario revisar las suposiciones y re-ejecutar el CIA para determinar si el escenario se satisface.

Por defecto, ArchGuru permite configurar estos costos y probabilidades en un archivo. Para obtener el costo de realizar una modificación en un módulo o interface, se utiliza una matriz de doble entrada. La primer entrada es la complejidad del módulo y la segunda, el número de responsabilidades/operaciones que contiene dicho módulo. Para obtener el costo total de realizar el cambio descripto por un escenario de modificabilidad, se suman los costos individuales de los módulos afectados.

3.4 Validaciones de conformidad (estructural y de escenarios)

ArchGuru permite realizar dos chequeos durante el ciclo de vida de un sistema: (i) conformidad estructural de la arquitectura del sistema, y (ii) chequeo de las suposiciones generadas por el algoritmo de CIA. A continuación se detallan las validaciones realizadas en cada uno de los chequeos.

Conformidad de la Arquitectura del Sistema Esta validación consiste en detectar diferencias estructurales entre el código fuente implementado y la vista de módulos diseñada, siguiendo los lineamientos del modelo de reflexión. En realidad, se trabaja con un grafo de dependencias de la implementación extraído mediante un análisis estático. Dicho grafo contiene las clases del sistema y sus

dependencias de uso. Al realizar el análisis de conformidad estructural de dicho grafo contra la vista de módulos como referencia, la herramienta reporta las ausencias como advertencias, y reporta las divergencias como errores.

Chequeo de las Suposiciones Generadas por CIA Durante la ejecución del algoritmo CIA, en tiempo de diseño, se generaron una serie de suposiciones sobre los elementos de la vista de módulos, que dieron sustento al análisis de los escenarios de modificabilidad. El chequeo de suposiciones se realiza una vez obtenido el grafo del CIA. La validación cobra sentido una vez que el sistema ha sido implementado porque el resultado original del CIA podria no ser correcto, si es que existen síntomas de erosión en el código.

El análisis consiste en comprobar que los módulos y el código fuente se mantengan dentro de las métricas establecidas. Si la comprobación del grafo de CIA sobre la vista de módulos no concuerda con el código fuente, debe revisarse el análisis de CIA ajustando las suposiciones y re-computando los valores de los escenarios de modificabilidad.

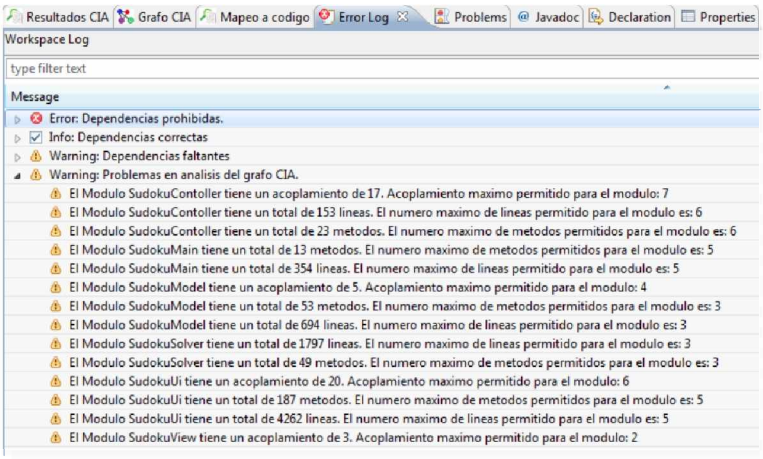

Fig. 6: Validación de las suposiciones generadas por el CIA

Por ejemplo, si se supone un módulo A cuya configuración es "medium", y en el archivo de configuración las métricas establecidas para un módulo "medium" son: líneas de código = 350, número de métodos = 50 y acoplamiento = 10; ArchGuru calculará las métricas para el módulo A en base al código fuente. Suponiendo que se obtienen los siguientes valores: líneas de código = 450, número de métodos = 23, y acoplamiento = 12; ArchGuru determinará que las métricas *líneas de código* y *acoplamiento* exceden los valores permitidos. Esta situación será informada al arquitecto para que la analice y realice las acciones necesarias para restaurar la consistencia. La Fig. 6 muestra la salida de ArchGuru para la validación de las suposiciones generadas por CIA.

4 Caso de Estudio

En esta sección, se presenta la evaluación del enfoque. La misma fue realizada sobre el sistema Health Watcher System (HWS), un sistema que recolecta y

gestiona quejas y notificaciones relacionadas con la salud pública. El objetivo fue analizar la aplicabilidad de ArchGuru y su capacidad para mostrar sintomas de disconformidades al usuario, más que realizar una evaluación de la efectividad de la herramienta para reducir los esfuerzos en chequeos de conformidad.

Para poder evaluar la herramienta ArchGuru sobre la aplicación HWS, se contó con la colaboración de dos expertos en la aplicación, que jugaron los roles de arquitecto (para las actividades 1-2 de la Figura 1) y de desarrollador (para la actividad 3). Los expertos crearon las vistas de módulos y escenarios de modificabilidad; y además, mapearon los distintos módulos de la vista al código fuente de la aplicación. Por último, los expertos ajustaron los distintos parámetros utilizados en la configuración del CIA (actividad 5), como las probabilidades de propagación, y la complejidad de los módulos.

4.1 Análisis de CIA

Para determinar la precisión del algoritmo CIA se realizaron distintas pruebas sobre la última versión de HWS (v. 10). Para esto, los expertos definieron la vista de módulos del sistema y 4 escenarios de modificabilidad:

- Escenario 1: Modificar los datos de un empleado.
- Escenario 2: Administrar la información del sistema.
- Escenario 3: Permitir al ciudadano registrar un nuevo tipo de queja.
- Escenario 4: Agregar un tipo nuevo de consulta que pueda ser realizada por un ciudadano.

Se realizaron 100 ejecuciones de CIA sobre cada escenario de modificabilidad. Todas las ejecuciones se realizaron con la misma configuración arquitectónica (determinada por los expertos). Dado que el algoritmo CIA se basa en probabilidades para determinar si un cambio debe propagarse de un módulo hacia otro, es posible que en distintas ejecuciones se obtengan resultados diferentes. Por este motivo se realizaron varias ejecuciones tomando el mejor y peor caso para cada escenario de modificabilidad.

	Módulos afectados (reales)		Módulos afectados (ArchGuru)	Precisión	Recall
Escenario 1	4	Mejor caso	5	0,8	1
		Peor caso	5	0,6	0,75
Escenario 2	1	Mejor caso	4	0,25	1
		Peor caso	7	0,14	1
Escenario 3	3	Mejor caso	6	0,5	1
		Peor caso	4	0,25	0,33
Escenario 4	2	Mejor caso	4	0,5	1
		Peor caso	7	0,29	1

Table 1: Resultados algoritmo CIA

La Tabla 1 presenta los resultados para cada escenario sobre el que se ejecutó CIA. Como se puede observar, la precisión para el mejor caso varía entre 25% y 80%; y para el peor caso varía entre un 14% y un 60%. Esta variación se debe a que la cantidad de módulos afectados "reales" para cada escenario son muy

pocos (entre 1 y 4 módulos). De esta forma, la precisión del algoritmo CIA se ve afectada dado que teniendo 1 o más falsos positivos la precisión disminuye de forma notoria. Es decir, a mayor cantidad de módulos afectados (reales) la precisión mejora. Esto puede verse en el caso del escenario 1 que es el que tiene un mayor número de módulos afectados "reales" y una mayor precisión. También puede observarse que la precisión para el escenario 2 es relativamente baja con respecto a los otros escenarios. Esto se debe a que existe un solo módulo afectado para este escenario. Con respecto al recall, se obtuvieron buenos valores (100%) en la mayoría de los casos. Solo existió un caso con recall bajo para el peor caso.

4.2 Análisis de CA

Para evaluar el CA del enfoque, se utilizaron 10 versiones de HWS (enumeradas de 1 a 10). Conjuntamente los expertos crearon 4 versiones de la vista de módulos para las versiones 1, 3, 6 y 9. Con el fin detectar problemas estructurales entre la arquitectura y su implementación se realizaron pruebas sobre una versión de la arquitectura y diferentes versiones del código, es decir, utilizando la versión X de la arquitectura, se ejecutó el análisis de conformidad sobre la versión X+1 del código fuente y la versión X+2. Esto puede verse ejemplificado en la Fig. 7. Por ejemplo, la versión de la arquitectura se creó a partir de la versión 3 del código fuente. De esta forma, se llevo a cabo el análisis de CA con esta arquitectura en las versiones 3, 4 y 5 del código fuente. Se analizaron dos situaciones: erosión y corrimiento arquitectónico.

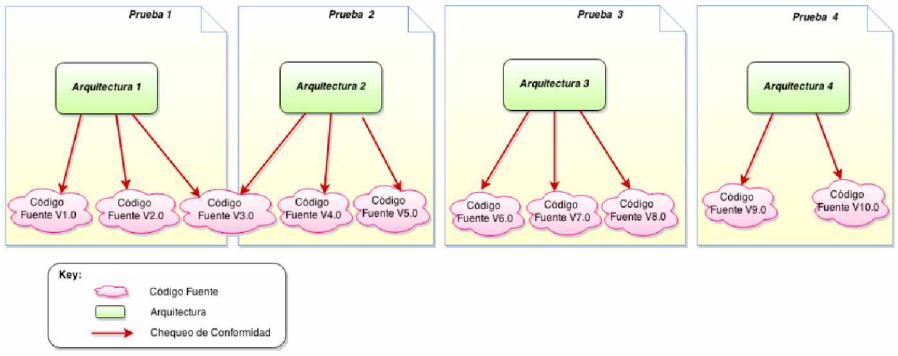

Fig. 7: Diseño experimentación CA

Análisis de Erosión de la Arquitectura El análisis de erosión de la arquitectura indica cúan desgastada se encuentra la arquitectura con respecto a su implementación. Por ejemplo, la Tabla 2 muestra los datos obtenidos para la evaluación de CA de la Arquitectura 1 con las versiones de código fuente 1, 2 y 3. Se tomó como punto de partida la arquitectura 1 con la versión nro. 1 de código fuente. En dicho análisis los resultados son los ideales ya que el código se corresponde con la arquitectura.

Para la versión nro. 2 de código fuente, el análisis de CA detectó algunas diferencias con respecto a la versión de código anterior. Los valores en color

	Código v. 1			Código v. 2			Código v. 3		
Módulo	Acoplamiento	Líneas de código	Número de métodos	Acoplamiento	Líneas de código	Número de métodos	Acoplamiento	Líneas de código	Número de métodos
View	0	1306	26	0	1732	56	0	1732	56
Bussiness	4	715	63	2	715	63	5	715	63
Data	14	2366	123	14	2366	123	14	2366	123
Model	5	707	124	2	707	124	2	685	131
Exceptions	0	137	22	0	137	22	0	137	22
Persistence	1	231	12	0	231	12	0	231	12
Concurrency	0	40	3	0	40	3	0	40	3
Distribution	0	151	10	0	151	10	1	151	10
Util	5	1157	71	4	1157	71	4	1157	71

Table 2: Resultados algoritmo CA para arquitectura 1

azul indican una variación en las métricas dentro de los parámetros establecidos (por los expertos) en el archivo de configuración para dicha complejidad. En este caso las métricas que fluctuaron dentro de los límites establecidos para una determinada complejidad son el acoplamiento y el número de métodos. Los datos que se encuentran en color rojo son métricas en las que el análisis de CA también detectó variación con respecto a la versión de la arquitectura 1 pero con la diferencia de que se superaron los umbrales máximos permitidos configurados en ArchGuru. El módulo *View* es el único que posee una métrica que supera el límite máximo configurado (líneas de código). Luego de analizar manualmente el código fuente se encontró que los cambios introducidos en la versión 2 corresponden a la implementación del patrón Command, siendo la mayoría de las clases afectadas pertenecientes al módulo *View*. Esta situación provocó un incremento en el número de líneas de código superando así, el límite máximo establecido para dicho módulo. A su vez, la refactorización del código e implementación del patrón Command derivó en la disminución del acoplamiento entre los módulos.

Para las arquitecturas 2, 3 y 4 si bien se encontraron variaciones, no se detectó ningún módulo que viole las métricas configuradas para la arquitectura.

Versión arquitectura	Versión código	Convergencia	Divergencia	Ausencia
1	1	39	2	0
	2	39	2	0
	3	39	173	0
2	3	42	2	0
	4	42	20	0
	5	42	58	0
3	6	55	4	0
	7	54	130	1
	8	54	148	1
4	9	56	4	0
	10	53	59	1

Table 3: Convergencia, divergencias y ausencias.

Análisis de Corrimiento Arquitectónico Además de erosión arquitectónica, ArchGuru también detecta corrimiento entre la arquitectura y el código fuente. Para esto se utilizaron los mismos análisis, entre arquitecturas y versiones de

código, mencionados anteriormente. La Tabla 3 muestra los valores de convergencia, ausencia y divergencia obtenidos para el análisis de las distintas arquitecturas. En la tabla puede observarse que no se detectaron ausencias para la arquitectura 1. Con respecto a las divergencias que se detectaron en las versiones de código 1 y 2, las mismas corresponden a dos dependencias entre interfaces las cuales no soporta el enfoque. En la versión 3 del código fuente se observan 173 divergencias, las cuales 2 son las mismas que se mencionaron anteriormente y las 171 restantes corresponden a clases nuevas que fueron creadas y las cuales no se encuentran mapeadas a ningún módulo de la arquitectura. Las 171 divergencias no significan que se hayan agregado 171 clases nuevas, sino que el enfoque toma en cuenta todas las dependencias que posean las clases no mapeadas. Las clases mencionadas corresponden a un nuevo módulo que aún no existe en la arquitectura. El módulo se agregó en la versión 2 de la misma. Cuando se realiza el mapeo de código a la nueva versión de la arquitectura las 171 se subsanan. En base a este resultado se puede decir que la arquitectura se encontraba desfasada con respecto al código fuente (corrimiento arquitectónico).

Con respecto a la versión 2 de la arquitectura, de igual manera que en el análisis anterior, no se encuentran ausencias, y para la versión 3 del código fuente se detectaron 2 divergencias (relaciones entre interfaces no soportadas por el enfoque). Para el análisis de la versión 4 y 5 de código fuente se detectaron 20 y 58 divergencias respectivamente. Son casos similares al detallado en el análisis de la arquitectura 1, dos de estos casos corresponden a relaciones entre interfaces y las restantes pertenecen a clases nuevas las cuales no fueron mapeadas a elementos arquitectónicos porque los mismos aún no están definidos en la arquitectura. Para ambos casos (análisis de conformidad sobre el código fuente versión 4 y código fuente versión 5) se debe llevar a cabo una revisión de la arquitectura ya que ArchGuru detectó corrimiento arquitectónico entre la versión de la arquitectura analizada y las versiones de código fuente correspondientes. Análisis similares pueden inferirse de las arquitecturas 3 y 4.

5 Trabajos Relacionados

El problema de conformidad arquitectónica ha sido explorado mediante técnicas de ingeniería reversa [15, 8] y lenguajes de descripción arquitectónica (ADLs) [1, 16]. Por ejemplo, en [1] se propone un enfoque que busca mantener sincronía entre la especificación arquitectónica y el código fuente. Sin embargo, esta sincronización no aborda el dinamismo arquitectónico ni la correspondencia de comportamiento entre la arquitectura y la implementación. En [17] se presenta una herramienta llamada RMTool que permite analizar diferencias entre el diseño arquitectónico y su implementación. En [6] se presenta ArchSync, la cual es una herramienta que ayuda a los arquitectos a chequear la conformidad entre una descripción arquitectónica basada en escenarios y la implementación. Los escenarios representan comportamientos arquitecturales del sistema y están modelados con mapas de casos de uso. Al igual que ArchGuru, estos enfoques permiten reconocer corrimiento arquitectónico. Sin embargo, diferentemente de nuestro enfoque no permiten reconocer situaciones de erosión.

Con respecto a CIA, se han presentado un conjunto de trabajos que se centran en un análisis a nivel de diseño e implementación. En [2] se presenta el prototipo

What-If Impact Analysis Tool que se implementó para analizar a priori el impacto de las dependencias del código. Aunque se ha desarrollado el prototipo, no se ha integrado en ningún entorno de desarrollo y tampoco ha sido testeado en proyectos reales. En [12] se presenta la técnica PathImpact, la cual construye una representación del comportamiento del sistema en tiempo de ejecución y la utiliza para estimar el impacto de cambio. Lee y Offut en [14] analizan las características del software orientado a objetos para definir cómo la composición, la herencia y el polimorfismo pueden influenciar en el CIA. Para ello categorizan los diferentes tipos de cambios que podrían ser aplicados en un sistema orientado a objetos y asignan a cada tipo un atributo que indica la influencia del cambio en otros objetos del sistema. Las aproximaciones presentadas en estos enfoques tienen como objetivo evaluar a priori el impacto de un cambio, es decir, hacen un análisis predictivo sobre un plan antes de tratar con las consecuencias.

6 Conclusiones

En este trabajo se presentó ArchGuru, un enfoque que integra el análisis de impacto de cambios (CIA) al análisis estructual tradicional de conformidad entre una arquitectura y su implementación. El enfoque permite detectar en la implementación: i) relaciones de divergencia y ausencia entre módulos, y ii) síntomas de erosión en el código que comprometen las suposiciones de un CIA basado en escenarios respecto al nivel de complejidad de las clases y su grado de acoplamiento. Si bien nuestro enfoque emplea algoritmos conocidos para conformidad estructural y CIA, la contribución de este trabajo radica en la integración de CIA desde una perspectiva de conformidad guiada por el atributo de modicabilidad. Para determinar la factibilidad del enfoque se utilizaron distintas versiones de un sistema real (HWS). El enfoque detectó correctamente tanto violaciones estructurales como situaciones de erosión entre las distintas versiones del código fuente. Como limitación del enfoque puede mencionarse la dependencia que existe con los mapeos entre los componentes arquitectónicos y el código. Esto provoca que deban rehacerce parcialmente los mapeos en cada nueva versión del sistema.

Como trabajo futuros, se pretende validar la efectividad del enfoque con un mayor número de sistemas, y se analizará el uso de otras técnicas de CIA que permitan aumentar la precisión del mismo. Por otro lado, se planea extender el tipo de análisis de escenarios de modificabilidad a otros atributos de calidad como performance y seguridad.

References

1. Abi-Antoun, M., Aldrich, J., Garlan, D., Schmerl, B., Nahas, N., Tseng, T.: Improving system dependability by enforcing architectural intent. ACM SIGSOFT Software Engineering Notes 30(4), 1–7 (2005)
2. Ajila, S.: Software maintenance: an approach to impact analysis of objects change. Software: Practice and Experience 25(10), 1155–1181 (1995)
3. Bohner, S.A.: Software Change Impact Analysis, chap. Software Change Impact Analysis for Design Evolution, pp. 67–81 (1996)
4. Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., Little, R.: Documenting software architectures: views and beyond. Pearson Education (2002)

5. Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
6. Díaz-Pace, J., Soria, Á., Rodríguez, G., Campo, M.R.: Assisting conformance checks between architectural scenarios and implementation. Information and Software Technology 54(5), 448–466 (2012)
7. Dimech, C., Balasubramaniam, D.: Maintaining architectural conformance during software development: A practical approach. In: Drira, K. (ed.) Software Architecture, LNCS, vol. 7957, pp. 208–223. Springer Berlin Heidelberg (2013)
8. Garcia, J., Popescu, D., Edwards, G., Medvidovic, N.: Identifying architectural bad smells. In: Winter, A., Ferenc, R., Knodel, J. (eds.) CSMR. pp. 255–258. IEEE (2009)
9. Herold, S.: Checking architectural compliance in component-based systems. In: Proceedings of the 2010 ACM Symposium on Applied Computing. pp. 2244–2251. ACM (2010)
10. Johnson, M., Maximilien, E., Ho, C.W., Williams, L.: Incorporating performance testing in test-driven development. Software, IEEE 24(3), 67–73 (May 2007)
11. Kruchten, P.B.: The 4+ 1 view model of architecture. Software, IEEE 12(6), 42–50 (1995)
12. Law, J., Rothermel, G.: Whole program path-based dynamic impact analysis. In: Software Engineering, 2003. Proceedings. 25th International Conference on. pp. 308–318. IEEE (2003)
13. Li, B., Sun, X., Leung, H., Zhang, S.: A survey of code-based change impact analysis techniques. Softw. Test., Verif. Reliab. 23(8), 613–646 (2013)
14. Li, L., Offutt, A.J.: Algorithmic analysis of the impact of changes to object-oriented software. In: Software Maintenance 1996, Proceedings., International Conference on. pp. 171–184. IEEE (1996)
15. Medvidovic, N., Egyed, A., Gruenbacher, P.: Stemming architectural erosion by coupling architectural discovery and recovery. In: STRAW. vol. 3, pp. 61–68 (2003)
16. Medvidovic, N., Rosenblum, D.S., Taylor, R.N.: A language and environment for architecture-based software development and evolution. In: Software Engineering, 1999. Proceedings of the 1999 International Conference on. pp. 44–53. IEEE (1999)
17. Murphy, G.C., Notkin, D., Sullivan, K.J.: Software reflexion models: Bridging the gap between design and implementation. IEEE Trans. Softw. Eng. 27(4), 364–380 (Apr 2001)
18. Ozkaya, I., Díaz Pace, J.A., Gurfinkel, A., Chaki, S.: Using architecturally significant requirements for guiding system evolution. In: CSMR. pp. 127–136. IEEE (2010)
19. Passos, L., Terra, R., Valente, M.T., Diniz, R., Mendonça, N.C.: Static architecture-conformance checking: An illustrative overview. IEEE software 27(5), 82–89 (2010)
20. Postma, A.: A method for module architecture verification and its application on a large component-based system. Information and Software Technology 45(4), 171–194 (2003)
21. Van Eyck, J., Boucké, N., Helleboogh, A., Holvoet, T.: Using code analysis tools for architectural conformance checking. In: Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge. pp. 53–54. SHARK '11, ACM, New York, NY, USA (2011)