



UNIVERSIDAD DE CHILE  
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS  
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

**EXPLORACIÓN DE LA OPTIMALIDAD DE SEÑALES NO-VISUALES DE  
CÓDIGO FUENTE EN AMBIENTES DE DESARROLLO INTEGRADO PARA  
PROGRAMADORES CIEGOS**

TESIS PARA OPTAR AL GRADO DE  
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN

ROBERTO GABRIEL PUGA FRANCO

PROFESOR GUÍA:  
FRANCISCO GUTIÉRREZ FIGUEROA

MIEMBROS DE LA COMISIÓN:  
JUAN ÁLVAREZ RUBIO  
JULIO GUERRA HOLLSTEIN  
IVÁN SIPIRÁN MENDOZA

SANTIAGO DE CHILE

2022

# Resumen

El resaltado de sintaxis permite a los programadores videntes explorar código y obtener información contextual de manera eficiente. Sin embargo, cuando se trata de programadores ciegos, no existe un estándar definido con respecto a los mecanismos no-visuales óptimos para la transmisión de este tipo de información. Se identifica como problema la transmisión optimal de información contextual del código a programadores ciegos.

La presente tesis describe la solución al problema a través de un modelo conceptual de casos de uso de señales de código fuente, y guías de diseño para la accesibilidad de la información contextual del código. Las propuestas se sustentan simultáneamente en la literatura como también en estudios empíricos.

El modelo conceptual de casos de uso propuesto categoriza las expectativas del usuario con respecto a la señales de código fuente según el tipo de actividad realizada y la familiaridad con la base de código con la que interactúa. Se estudió la validez de este modelo a través de una prueba pareada entre sujetos. En esta prueba participaron programadores videntes, con distinto nivel de experiencia. Sus resultados permiten inferir que el impacto de las señales visuales de código fuente depende del caso de uso evaluado.

Para lograr transmitir información contextual del código de manera no-visual, se implementó una extensión para el ambiente de desarrollo integrado Visual Studio Code. Esta permite que, al seleccionar o escribir una palabra reservada del lenguaje de programación Python, se genere una señal no-visual determinada. Esta extensión logra, según su configuración, emitir tres tipos de señales no-visuales: audio neutro, texto hablado, y vibraciones.

A través de un caso de estudio único holístico exploratorio se midió el desempeño de programadores ciegos en la resolución de desafíos de programación. Cada desafío del escenario experimental correspondía a una combinación única de un caso de uso, que debía ser resuelto utilizando una configuración particular de la extensión desarrollada. Los resultados demostraron una clara preferencia por los mecanismos de menor latencia, y son prometedores con respecto al uso de retroalimentación háptica.

Finalmente, se presentan consideraciones de diseño de ambientes de desarrollo integrado accesibles a programadores ciegos. Estas sugieren a los *earcons* como mecanismo de retroalimentación optimal, por la accesibilidad, oportunidad, celeridad y discernibilidad que exhiben. Además, se concluye sobre el trabajo realizado y se presentan perspectivas de trabajo futuro.



# Agradecimientos

Agradezco a mi familia, que me ha guiado, acompañado y motivado en este arduo camino.

Agradezco a mis amigos, que me han prestado su apoyo en los momentos clave de este proceso.

Agradezco a las personas que conocí por motivo de este trabajo, con quienes comparto el ideal de un mundo diverso y accesible.

Y agradezco a mis profesores, en especial a mi guía.



# Tabla de Contenido

|  |           |
|--|-----------|
| <b>1. Introducción</b>   | <b>1</b>  |
| 1.1. Problema abordado . . . . .   | 2         |
| 1.2. Hipótesis . . . . .   | 4         |
| 1.3. Preguntas de investigación . . . . .                                    | 4         |
| 1.4. Objetivos . . . . .   | 5         |
| 1.4.1. Objetivo general . . . . .  | 5         |
| 1.4.2. Objetivos específicos . . . . .                                       | 5         |
| 1.5. Solución . . . . .  | 5         |
| 1.6. Metodología . . . . .   | 6         |
| 1.7. Estructura del documento . . . . .                                      | 7         |
| <b>2. Trabajo Relacionado</b>  | <b>8</b>  |
| 2.1. Comprensión de programas y NVCs . . . . .                               | 8         |
| 2.2. Señales visuales en VSC . . . . .                                       | 11        |
| 2.3. Síntesis . . . . .  | 14        |
| <b>3. Modelo Conceptual de Casos de Uso</b>                                  | <b>15</b> |
| 3.1. Modelo propuesto . . . . .  | 15        |
| 3.2. Validación experimental del Modelo conceptual de casos de uso . . . . . | 17        |
| 3.2.1. Hipótesis . . . . .   | 17        |
| 3.2.2. Diseño experimental . . . . .   | 17        |
| 3.2.3. Instrumentos . . . . .  | 19        |
| 3.2.4. Participantes . . . . .   | 20        |
| 3.2.5. Aplicación . . . . .  | 21        |
| 3.2.6. Análisis de respuestas . . . . .                                      | 22        |
| 3.2.7. Resultados e interpretación . . . . .                                 | 24        |
| 3.2.7.1. Generales . . . . .   | 25        |
| 3.2.7.2. Por caso de uso . . . . .   | 26        |
| 3.2.8. Conclusiones . . . . .  | 28        |
| <b>4. VisualCueTransform, extensión para Visual Studio Code</b>              | <b>29</b> |
| 4.1. Implementación . . . . .  | 29        |
| 4.1.1. Modos sonido inerte y texto hablado . . . . .                         | 30        |
| 4.1.2. Modo retroalimentación háptica . . . . .                              | 31        |
| 4.2. Estructura del código . . . . .   | 32        |
| 4.2.1. Extensión VisualCueTransform . . . . .                                | 32        |
| 4.2.1.1. Clases . . . . .  | 32        |
| 4.2.1.2. Funciones . . . . .   | 43        |

|   |           |
|---|-----------|
| 4.2.2. Servidor HTTP y BLE . . . . .                                | 45        |
| 4.3. Retrospectiva . . . . .  | 46        |
| <b>5. Estudio con Usuarios</b>                                      | <b>47</b> |
| 5.1. Preguntas de investigación . . . . .                           | 47        |
| 5.2. Unidad de análisis . . . . .                                   | 48        |
| 5.3. Diseño experimental . . . . .                                  | 48        |
| 5.3.1. Validez de constructo . . . . .                              | 51        |
| 5.3.2. Validez interna . . . . .                                    | 51        |
| 5.3.3. Validez externa . . . . .                                    | 52        |
| 5.3.4. Validez ecológica . . . . .                                  | 52        |
| 5.3.5. Confiabilidad . . . . .                                      | 52        |
| 5.3.6. Replicabilidad . . . . .                                     | 53        |
| 5.4. Instrumentos . . . . .   | 53        |
| 5.5. Participantes . . . . .  | 54        |
| 5.6. Aplicación . . . . .   | 55        |
| 5.7. Análisis de respuestas . . . . .                               | 56        |
| 5.7.1. Análisis cuantitativo . . . . .                              | 56        |
| 5.7.2. Análisis cualitativo . . . . .                               | 58        |
| 5.8. Resultados e interpretación . . . . .                          | 58        |
| 5.8.1. Casos de uso de lectura . . . . .                            | 58        |
| 5.8.2. Casos de uso de escritura . . . . .                          | 59        |
| 5.8.3. Optimalidad por tipo de retroalimentación . . . . .          | 61        |
| 5.8.4. Optimalidad por familiaridad con la base de código . . . . . | 62        |
| 5.9. Conclusiones . . . . .   | 63        |
| <b>6. Consideraciones de Diseño</b>                                 | <b>65</b> |
| 6.1. Señales no-visuales de código . . . . .                        | 65        |
| 6.2. Implementación de NVCs en IDEs . . . . .                       | 67        |
| 6.3. Implementación de NVCs en lectores de pantalla . . . . .       | 67        |
| <b>7. Conclusiones</b>  | <b>69</b> |
| <b>Bibliografía</b>   | <b>72</b> |
| <b>Anexos</b>   | <b>74</b> |
| <b>Anexo A. Prueba pareada</b>                                      | <b>74</b> |
| A.1. Instrumentos . . . . .   | 74        |
| A.1.1. Transcripción formulario . . . . .                           | 74        |
| A.1.2. Capturas formulario . . . . .                                | 77        |
| <b>Anexo B. Caso de estudio</b>                                     | <b>88</b> |
| B.1. Instrumentos . . . . .   | 88        |
| B.1.1. Cuestionario de autoreporte . . . . .                        | 89        |
| B.1.2. Transcripción protocolo experimental . . . . .               | 90        |
| B.2. Certificación . . . . .  | 96        |
| B.2.1. Certificación comité de ética FCFM . . . . .                 | 97        |

|  |    |
|--|----|
| B.3. Hardware . . . . .                              | 99 |
| B.3.1. Computador para interacción usuaria . . . . . | 99 |
| B.3.2. Servidor bluetooth . . . . .                  | 99 |
| B.3.3. Pulsera retroalimentación háptica . . . . .   | 99 |

# Índice de Tablas

|      |   |    |
|------|---|----|
| 3.1. | Esquema de configuración de señales visuales según partes del instrumento y grupos. . . . . | 19 |
| 3.2. | Distribución de participantes prueba pareada. . . . .                                       | 21 |
| 3.3. | Requerimientos considerados en cada formato para cada medición. . . . .                     | 24 |
| 3.4. | Mediana Medición A por clasificadores. . . . .  | 25 |
| 3.5. | Mediana Medición B por clasificadores. . . . .  | 25 |
| 3.6. | Indicadores estadísticos de Mediciones A y B por clasificadores. . . . .                    | 26 |
| 3.7. | Indicadores estadísticos de Mediciones por caso de uso. . . . .                             | 27 |
| 3.8. | Indicadores estadísticos de Mediciones por escritura o lectura. . . . .                     | 27 |
| 3.9. | Indicadores estadísticos de Mediciones por familiaridad con la base de código. . . . .      | 27 |
| 4.1. | Palabras reservadas en Python según alcance asociado por VSC. . . . .                       | 30 |
| 5.1. | Esquema de configuración de señales visuales según partes del diseño experimental. . . . .  | 49 |
| 5.2. | Especificaciones de problemas de lectura de código. . . . .                                 | 57 |
| 5.3. | Especificaciones de problemas de escritura de código. . . . .                               | 57 |
| 5.4. | Resultados de problemas de lectura de código. . . . .                                       | 58 |
| 5.5. | Resultados de problemas de escritura de código. . . . .                                     | 60 |
| 5.6. | Resultados de análisis satisfacción y SUS. . . . .  | 61 |
| 5.7. | Diferencias de tiempo según conocimiento de la base de código. . . . .                      | 62 |

# Índice de Ilustraciones

|       |  |    |
|-------|--|----|
| 1.1.  | Ejemplo de exploración del lenguaje utilizando VCs. . . . .                                  | 3  |
| 2.1.  | Diagrama de tipos de señales visuales en VSC y su funcionamiento. . . . .                    | 12 |
| 2.2.  | Código fuente en Visual Studio Code utilizando el tema “Dark+ (default dark)”. . . . .       | 13 |
| 2.3.  | Código fuente en Visual Studio Code utilizando el tema “Plain (Dark)”. . . . .               | 13 |
| 3.1.  | Modelo conceptual de casos de uso y roles de las señales del código. . . . .                 | 16 |
| 3.2.  | Esquema resumen sobre la implementación del diseño experimental. . . . .                     | 20 |
| 4.1.  | Modelo de implementación retroalimentación háptica. . . . .                                  | 32 |
| 5.1.  | Esquema resumen sobre la implementación del diseño experimental. Parte 0 a Parte 6. . . . .  | 54 |
| 5.2.  | Esquema resumen sobre la implementación del diseño experimental. Parte 7 a Parte 13. . . . . | 54 |
| A.1.  | Captura de pantalla sección 1 formato A. . . . .   | 77 |
| A.2.  | Captura de pantalla sección 2 formato A. . . . .   | 78 |
| A.3.  | Captura de pantalla sección 3 formato A. . . . .   | 79 |
| A.4.  | Captura de pantalla sección 4 formato A. . . . .   | 80 |
| A.5.  | Captura de pantalla sección 5 formato A. . . . .   | 81 |
| A.6.  | Captura de pantalla sección 6 formato A. . . . .   | 82 |
| A.7.  | Captura de pantalla sección 7 formato A. . . . .   | 83 |
| A.8.  | Captura de pantalla sección 8 formato A. . . . .   | 84 |
| A.9.  | Captura de pantalla sección 9 formato A. . . . .   | 85 |
| A.10. | Captura de pantalla sección 10 formato A, primera parte. . . . .                             | 86 |
| A.11. | Captura de pantalla sección 10 formato A, segunda parte. . . . .                             | 87 |
| B.1.  | Cuestionario de autoreporte para caso de estudio único . . . . .                             | 89 |
| B.2.  | Certificado emitido por el comité de ética FCFM. Página 1. . . . .                           | 97 |
| B.3.  | Certificado emitido por el comité de ética FCFM. Página 2. . . . .                           | 98 |

# Capítulo 1

## Introducción

La evolución tecnológica ha traído consigo nuevas maneras de garantizar la accesibilidad a personas con discapacidad visual. Por ejemplo, Software como NVDA<sup>1</sup> o JAWS<sup>2</sup>, aplicaciones para la lectura de pantalla, permiten a las personas ciegas utilizar un computador. Esto se logra a través de la transformación del texto en audio mediante una voz sintética o braille. Por otra parte, la aplicación móvil Lazarillo [21] permite acceder, a través de mensajes de voz, a instrucciones de desplazamiento, como las que provee Google Maps [15] a usuarios videntes. Ambos tipos de aplicaciones han permitido aumentar el grado de independencia de las personas ciegas.

A pesar del constante avance tecnológico-científico en materia de accesibilidad, los mercados educativo y laboral en Chile, para las personas con discapacidad visual, no se han desarrollado al mismo ritmo. De acuerdo al Servicio Nacional de la Discapacidad, aún existen restricciones para el ingreso de personas ciegas a la educación terciaria, lo que limita sus oportunidades laborales [27].

Un factor que puede resultar determinante en el ingreso al campo laboral relacionado con la computación, es la accesibilidad de las aplicaciones que facilitan la implementación de programas. Sin embargo, no todas las herramientas necesarias para los flujos de trabajo cuentan con configuraciones o extensiones que las hagan accesibles [25], lo que dificulta al empleador con respecto a las facilidades requeridas [20].

Una de las herramientas más importantes en desarrollo de software, respecto a productividad, es el ambiente de desarrollo integrado (en adelante IDE por sus siglas en inglés) que se emplee [30]. La mayoría de las interfaces de usuario de estas aplicaciones utilizan señales visuales (en adelante, VC por sus siglas en inglés), para transmitir información contextual del código. Se utilizan, por ejemplo, diferentes colores para la declaración de una variable o de una función, distintos estilos de escritura para palabras comunes o palabras reservadas, distinta saturación para diferenciar entre código y comentarios, etc. Sin embargo, este tipo de señales no es accesible para programadores ciegos o con discapacidad visual, ya que, al ser puramente visuales, no las perciben.

<sup>1</sup> Non-Visual Desktop Access

<sup>2</sup> Job Access With Speech

## 1.1. Problema abordado

Para transmitir información contextual del código a programadores ciegos<sup>3</sup>, son de especial atención tres elementos: qué información se transmite, cómo se transmite, y cuándo se debe desencadenar la transmisión. A continuación, se discute sobre estos elementos para definir el problema abordado.

Con respecto a qué transmitir, se propone transformar las VCs, de un IDE y lenguaje de programación particular, en señales no-visuales del código (en adelante NVCs por sus siglas en inglés) como punto de partida. Esto equipara la información contextual a la que pueden acceder tanto programadores ciegos como videntes.

Con respecto a cómo se transmite la información, se pueden considerar factibles diversos tipos de estímulos para funcionar como NVCs. Algunos de los tipos a considerar son: sonidos, texturas, y vibraciones, ya que las personas ciegas los utilizan en su día a día. Por ejemplo, el sonido de un semáforo al cruzar la calle, la textura de un texto en braille, o las vibraciones al escribir en un celular táctil.

Con respecto a cuándo se debe desencadenar la transmisión, investigaciones previas demuestran que esto depende tanto del tipo de señal, como del lenguaje de programación utilizados [2, 5, 7]. Todas las VCs en un IDE pueden estar activas en simultáneo, y es el programador quién fija su atención en la que le sea relevante. Si se reemplazaran directamente todas por sonidos, reproducirlos en simultáneo puede resultar contraproducente. Esto ya que el programador deberá enfocar su audición en una de ellas, lo que se dificulta conforme es mayor la cantidad de canales de audio [16, 26]. Por esto, se propone capturar eventos particulares que permitan desencadenar las NVCs en los momentos apropiados.

Cuando se ha estudiado el impacto del uso de VCs en IDEs, se ha hecho desde el prisma de su utilidad en la exploración de código [30]. Esto omite el hecho de que los programadores reciben y utilizan la información contextual que las VCs transmiten en otros tipos de instancias [6]. Por ejemplo, cuando un programador se enfrenta a un lenguaje de programación nuevo, puede usar las VCs para identificar la forma correcta de escribir una palabra reservada del lenguaje. En un IDE con VCs encendidas, podría probar las distintas combinaciones de mayúsculas y minúsculas hasta que el color en que se resalta la palabra “True” denote que lo ha escrito de la forma correcta. En esencia, el programador no está explorando la base de código, sino el lenguaje de programación mismo. La figura 1.1 da cuenta de este ejemplo. En adelante, se utilizará el concepto de “casos de uso” para referirse a las distintas instancias de desencadenamiento de información contextual del código.

<sup>3</sup> En este texto se utiliza el género masculino como forma no marcada para referirse a personas de distintos géneros. Esta decisión se basa únicamente en la necesidad de simplificar las estructuras utilizadas con el fin de construir un discurso comprensible. A pesar de esta opción lingüística, se reconoce la diversidad de géneros y se valora la importancia del lenguaje inclusivo.

```

57 def python_return_true():
58     return TRUE
59
60 def python_return_true():
61     return true
62
63 def python_return_true():
64     return True

```

Figura 1.1: Ejemplo de exploración del lenguaje utilizando VCs.

En el caso particular de los programadores ciegos, es necesario notar que los principales lectores de pantalla, por defecto no soportan el acceso a la información contextual que proveen las VCs en los IDEs [3]. Es por esto que, si el IDE que se utiliza no cuenta con una forma de hacer accesible la información, un programador ciego se ve en desventaja con respecto al dominio de información contextual, en comparación con un programador vidente [17].

Respecto a los IDEs, no todos cuentan con maneras de hacer accesibles las VCs, y los que lo hacen, utilizan, en su mayoría, señales basadas en audio. Esto puede producir un mayor desgaste cognitivo en un programador ciego, considerando que es un estímulo simultáneo al audio utilizado por los lectores de pantalla [14].

Algunos IDEs utilizan mecanismos basados en sonido como NVCs, pero la elección de este tipo de señales no es necesariamente la óptima. Existen estudios que demuestran que distintos tipos de NVCs pueden resultar mejores en distintos tipos de tareas de exploración. Por ejemplo, se ha medido empíricamente que el uso de *spearcons*<sup>4</sup> en menús avanzados mejora la usabilidad en la navegación [28]. Por esto, al construir un producto accesible, es necesario conocer cuáles son los parámetros que hacen óptima algún tipo de señal por sobre otra.

Considerando que las NVCs forman parte de sistemas de interacción humano-computador, se dirá entonces, en el contexto de esta tesis, que una NVC es *optimal* cuando su accesibilidad y usabilidad sea la mayor dentro del conjunto de NVCs disponibles para la transmisión de información. Para medir la usabilidad de una NVC, se utilizarán los criterios establecidos en la norma ISO 9241-11 [19], evaluando positivamente la mayor eficacia, eficiencia, y satisfacción del usuario en el contexto de uso. Este tipo de medición respecto de NVCs se ha sugerido previamente, pero no estudiado en lenguajes de programación basados en texto [2]. Además, se diferenciarán las evaluaciones de las NVCs según instancias de uso adicionales a la exploración de código. En adelante, las instancias en que se desencadena la transmisión de información contextual del código a través de señales serán llamadas *casos de uso*.

En la presente tesis se evalúa el contexto particular del IDE Visual Studio Code (VSC en adelante) en su versión 1.57, al desarrollar en el lenguaje de programación Python en su versión 3.8.9. Visual Studio Code es uno de los IDEs más populares a la fecha [10], se encuentra disponible para múltiples sistemas operativos, y ofrece una API<sup>5</sup> que permite extender sus capacidades. Por otro lado, según el índice PYPL[9] el lenguaje de programación

<sup>4</sup> Íconos auditivos basados en texto hablado acelerado

<sup>5</sup> Interfaz de programación de aplicaciones. API por su siglas en inglés.



Python es el más popular a nivel mundial. Además, se define como población objetivo a las personas ciegas, mayores de 18 años, que reporten tener experiencia programando.

El problema abordado es entonces **determinar cuáles son las NVCs óptimas para transmitir información contextual del código fuente a programadores ciegos, según el caso de uso**. La relevancia de este problema surge de las mejoras que se pueden realizar en los IDEs o los lectores de pantalla, para que los usuarios ciegos o con discapacidad visual puedan acceder a la información contextual del código. Esto facilitaría las tareas relacionadas al desarrollo o al aprendizaje de la computación. Se trata de un problema computacional, ya que comprende la transmisión de información en sistemas interactivos, así como los métodos e instrumentos utilizados en procesos de ingeniería de software.

Siguiendo el Sistema de Clasificación Computacional de la ACM[1], la resolución de este problema problema intersecta dos áreas del estudio de la computación: *Human-Centered Computing*, ya que se aporta conocimiento con respecto a los mecanismos optimales para la accesibilidad en IDEs a usuarios ciegos; y *Software and its Engineering*, ya que se estudia el impacto de las NVC en la ejecución de tareas relacionadas con el desarrollo de software, y se aporta un modelo que amplía las nociones sobre las que se evalúan las señales del código provistas por los IDE.

## 1.2. Hipótesis

A partir de lo expuesto, el presente trabajo de tesis se sustenta en la definición de las siguientes hipótesis:

- H1. La optimalidad de una señal de código depende del caso de uso. Entiéndase por optimalidad a la accesibilidad y mayor usabilidad de la señal para la transmisión de información contextual del código. Entiéndase por caso de uso a las instancias de transmisión de información contextual del código.
- H2. La información contextual transmitida por una señal visual del código puede ser transmitida por una señal no-visual del código óptima, desencadenada según el caso de uso.

## 1.3. Preguntas de investigación

A continuación, se presentan las preguntas de investigación que permiten abordar las hipótesis:

- P1. Considerando el alcance de la investigación ¿Para qué utiliza las señales visuales del código un programador vidente?  
La respuesta a esta pregunta permite definir y evaluar los casos de uso de señales visuales del código. Esto, a su vez, posibilita estudiar la validez de la hipótesis H1.
- P2. ¿Cómo se puede transformar una señal visual del código en una señal no-visual del código que transmita la misma información?

La respuesta a esta pregunta permite definir un proceso para la transformación de señales visuales del código en señales no-visuales. Esto, a su vez, permite la evaluación de señales no-visuales del código respecto de su optimalidad, por lo que está relacionada con la hipótesis H2.

P3. ¿Cuál es la señal no-visual del código óptima para transmitir la información contenida en una señal visual del código, en un caso de uso particular?

Finalmente, esta pregunta permite establecer una comparación entre distintas señales no-visuales del código, respecto de su optimalidad, y decidir, para un caso de uso en particular, cuál resulta apropiada. Esto se relaciona directamente con la hipótesis H2, y permite estudiar la validez de lo sostenido en H1 para el caso no-visual.

## 1.4. Objetivos

Para dar respuesta a las preguntas de investigación, se definen los siguientes objetivos:

### 1.4.1. Objetivo general

Determinar NVCs que resulten óptimas para la transmisión de información contextual del código a programadores ciegos. Esto en el contexto del IDE Visual Studio Code, utilizando el lenguaje de programación Python.

### 1.4.2. Objetivos específicos

- O1. Identificar, describir y clasificar los tipos de VC, la información que entregan y las funciones que cumplen.
- O2. Identificar, describir y clasificar los distintos casos de uso en que los programadores utilizan las VCs.
- O3. Definir e implementar un mecanismo que permita transformar una VC particular en una NVC según la detección de un caso de uso.
- O4. Explorar la optimalidad de los distintos tipos de NVCs como mecanismos de transmisión de información sobre el código según el caso de uso.

## 1.5. Solución

La solución consiste en un modelo conceptual sobre el uso de señales del código, y consideraciones de diseño para el uso de señales no-visuales del código en ambientes de desarrollo integrado. Para su materialización, se requiere de las siguientes componentes:

1. Un modelo conceptual de casos de uso, que complementa las nociones actuales sobre cómo evaluar señales del código, y que define los eventos relevantes para la desencade-

nación de señales no-visuales del código. Esta parte de la solución está relacionada con los objetivos específicos O1 y O2.

2. Un mecanismo que permita la detección de los casos de uso y captura de eventos desencadenantes de las señales del código, según el modelo definido. Este mecanismo está relacionado con el objetivo específico O3.
3. Un sistema generador de señales no-visuales, que sea desencadenado según el mecanismo de detección implementado, y que permita utilizar distintos tipos de señales no-visuales. Este mecanismo está relacionado con el objetivo específico O3.
4. Estudios empíricos con usuarios sobre el modelo propuesto, y los distintos tipos de señales no-visuales. A través de su aplicación y análisis de resultados, se caracteriza y cuantifica el impacto de las distintas variables, validando el modelo propuesto, y obteniendo una base material para presentación de consideraciones de diseño. Estos estudios se relacionan con el objetivo específico O4.

Con lo anterior, se expanden los sistemas de evaluación de señales del código, a través de un nuevo modelo conceptual que representa el sistema de interacciones, y se determina las NVCs óptimas, según los casos de uso de dicho modelo, a través del análisis de estudios empíricos.

## 1.6. Metodología

A continuación, se presentan las actividades realizadas para el desarrollo de la solución y cumplimiento de los objetivos específicos:

1. Caracterización de las VCs a través del uso exploratorio del ambiente de desarrollo integrado VSC, el estudio de su documentación y la revisión del trabajo relacionado. Esto dice directa relación con el objetivo O1.
2. Elaboración de un modelo de los casos de uso, identificados a través del uso exploratorio del ambiente de desarrollo, el estudio de su documentación, y la revisión del trabajo relacionado. Esta tarea se relaciona con el objetivo O2.
3. Elaboración de instrumentos para la medición de la optimalidad de las VCs, a ser utilizados en un estudio empírico entre sujetos, análogos a los utilizados en el estado del arte. Esto dice relación con el objetivo O2.
4. Validación del modelo conceptual de casos de uso elaborado, utilizando los instrumentos elaborados para su evaluación a través de un estudio empírico con usuarios. Esta tarea se relaciona con el objetivo O2.
5. Caracterización y selección, según su utilidad potencial, de NVCs que puedan ser producidas desde o a través de un ordenador, según el estudio del trabajo relacionado y la documentación técnica del hardware disponible. Esto dice relación con el objetivo O3.

6. Diseño e implementación de una extensión para el ambiente de desarrollo integrado VSC y el lenguaje de programación Python, en sus versiones más recientes, que permite transformar una VC en una de las NVCs seleccionadas al identificarse un caso de uso particular. Esto dice relación con el objetivo O3.
7. Elaboración de instrumentos para la medición de la optimalidad de las NVCs generadas por la extensión implementada, a ser utilizados en un estudio empírico con programadores ciegos. Esto dice relación con el objetivo O4.
8. Diseño de un caso de estudio único que permite explorar las diferencias en optimalidad del sonido inerte, el texto hablado y la retroalimentación háptica como NVCs asociadas a palabras clave del código fuente, utilizando los instrumentos elaborados. Esto dice relación con el objetivo O4.
9. Análisis de los datos recolectados del caso de estudio único exploratorio y discusión con respecto a sus resultados. Esto dice relación con el objetivo O4.
10. Elaboración de pautas de diseño para la transmisión de información sobre el código, de origen visual, a programadores ciegos. Esto dice relación con el objetivo O4.

## 1.7. Estructura del documento

El resto de este documento se estructura como sigue. En el capítulo 2 se presenta y discute el trabajo relacionado al problema de la exploración de código, el sistema generador de VCs que utiliza el ambiente VSC, y el uso de NVCs para la transmisión de información. En el capítulo 3 se define y valida experimentalmente un modelo de casos de uso, que amplía el marco teórico sobre el cual se estudia el impacto de las señales del código. En el capítulo 4 se describe el proceso de diseño e implementación de una extensión para VSC capaz de transformar, según el caso de uso, algunas de las VCs de código fuente en NVCs. En el capítulo 5 se presenta un caso de estudio único exploratorio, con programadores ciegos, en que se evalúa la optimalidad del sonido inerte, el texto hablado y la retroalimentación háptica como NVCs asociadas a palabras clave del código fuente, en los distintos casos de uso. En el capítulo 6 se discuten los resultados obtenidos y se propone un conjunto de abstracciones y pautas de diseño para la transmisión de información de origen visual sobre el código a programadores ciegos. Finalmente, en el capítulo 7 se presentan las conclusiones y se describen perspectivas de trabajo futuro.

# Capítulo 2

## Trabajo Relacionado

En el presente capítulo, se analiza y discute la literatura relacionada. Según su origen, se divide en dos subcapítulos. El primer subcapítulo dice relación con la literatura científica sobre los procesos, mecanismos y estrategias a través de los cuales un programador puede conocer o intuir el comportamiento de una pieza de código. En este capítulo también se trata el uso de mecanismos no-visuales para la transmisión de información. El segundo sub-capítulo corresponde a la documentación provista por VSC sobre el funcionamiento del resaltado de sintaxis y las restricciones que esto implica.

Estos son los cuerpos de literatura de principal relevancia para la presente tesis. A través de su estudio, se logra construir un marco teórico acerca de la comprensión de programas utilizando mecanismos no-visuales de transmisión de información.

### 2.1. Comprensión de programas y NVCs

En un estudio exploratorio, Albusays y Ludi [3] logran plasmar la incapacidad de los lectores de pantalla para transmitir, a programadores no videntes, la información contenida en algunas de los IDEs. Esto lo consiguen a través de encuestas enfocadas en explorar los desafíos que enfrenta una muestra de 69 programadores ciegos al desarrollar, y las herramientas y estrategias que suelen utilizar para superarlos. En los resultados de la encuesta, se refleja en la síntesis de las preguntas abiertas que los lectores de pantalla no son herramientas adaptadas al procesamiento de información compleja como la de los IDEs ya que presentan problemas de estabilidad. Algunos programadores ciegos optan por utilizar editores de texto, dejando de lado las capacidades adicionales de los IDEs. Son estos resultados los que motivan a plantear una implementación que, utilizando la capacidad de análisis gramatical de los IDEs, logre transmitir la información contextual del código a programadores no-videntes.

A través de la observación de 28 programadores ciegos y entrevistas semiestructuradas sobre los lenguajes de programación, herramientas de desarrollo y tecnologías de asistencia en relación a la exploración de código, la siguiente investigación exploratoria de Albusays, Ludi y Huenerfauth [4] resulta clave en el entendimiento del rol de los IDEs en los procesos de la ingeniería de software. En esta, se hace explícito que existe información contextual compleja que las VCs transmiten a los usuarios de IDEs, pero que no son accesibles para los

programadores ciegos. Se identifica, además, la oportunidad de mejorar la accesibilidad de las herramientas de navegación. Se debe notar que esta investigación aborda un caso de uso particular: la exploración de una base de código compleja, con foco en la navegación. Entre los resultados destaca que, de los 28 participantes, 24 tuvieron dificultades para depurar el código; 23 reportan problemas para encontrar información específica, sin recorrer la base de código línea a línea o carácter a carácter; 14 reportan dificultad encontrando errores en el código; y 14 reportan dificultades para reconocer los niveles de alcance (scope). En base a las dificultades reportadas en este estudio, se construyen los escenarios de prueba de las evaluaciones realizadas en la presente tesis. Esto permite medir el impacto de la transformación de las VCs en NVCs en escenarios que resultan críticos para los programadores ciegos.

Los lectores de pantalla leen textualmente al programador el contenido de un archivo; sin embargo, existen formas distintas de utilizar el audio: íconos auditivos, que son representaciones de sonidos reales de objetos o eventos, por ejemplo, el sonido de una campana o el ladrido de un perro; earcons, que son breves patrones musicales, por ejemplo las utilizadas en notificaciones de algunas redes sociales; o spearcons, que son earcons basados en discurso hablado acelerado. En 2012, Walker, Lindsay, Nance et al. [28] desarrollaron el uso de *spearcons* para asistir en la navegación contextual de menús auditivos avanzados. En ese contexto, este tipo de señal logró superar la eficiencia, precisión y facilidad de aprendizaje de los otros tipos de señales auditivas. Sin embargo, cuando lo que se busca lograr es representar la estructura de un lenguaje orientado a objetos, Hutchinson y Metatla [18] demuestran en su investigación que los sonidos que no están basados en el diálogo son preferidos por programadores, a pesar de que tienen una mayor curva de aprendizaje. Este último estudio considera la participación tanto de programadores videntes como invidentes, y no descarta los otros tipos de señales visuales como factibles, ya que resultan levemente mejor evaluadas en cuanto a la precisión lograda al utilizarlas.

Ludi, Simpson y Merchant [22] investigaron el uso de señales auditivas para la exploración en lenguajes por bloques. Como conclusión de su estudio, reconocen que distintos tipos de señales auditivas pueden cumplir, con distintas eficacias y eficiencias, el propósito de la transmisión de la información. Es a partir de esto que se puede preguntar si otros tipos de NVCs, como podrían ser la hápticas, de temperatura, electromagnéticas, de presión, o de relieve, entre otras, pueden ser también útiles para la transmisión de información, y cómo contrastan entre ellas. Para el análisis de resultados de la investigación de Ludi, Simpson y Merchant, se utiliza el tiempo como medida de eficiencia, y permite comparar las distintas señales basadas en audio que utilizaron. También se mide la eficacia de las señales en función de la similitud con un resultado esperado. Por otro lado, Albusay reconoce en su retrospectiva de la literatura [2], la ausencia de mediciones sobre usabilidad en NVCs. Por todo lo anterior, la presente tesis utiliza el concepto de *optimalidad* de una señal como parámetro comparador, quedando definido por la accesibilidad y usabilidad de la señal. Esto se refiere a cuán eficaz, eficiente y satisfactoriamente un usuario logra resolver una tarea de programación particular. Por otro lado, dichas tareas se diferenciarán según sean de lectura o escritura de código, lo que es un primer avance hacia un modelo conceptual de *casos de uso*.

Enfocado en la exploración del código, se han implementado extensiones para IDEs y lenguajes particulares, como StructJumper [7] para el lenguaje Java en el IDE Eclipse, que

intenta transmitir información acerca de la estructura del código. Cabe notar que los esfuerzos de Baker, Milne y Ladner, autores del artículo en que se presenta StrcutJumper, se centran en un IDE y lenguaje de programación particulares. Esto se debe a que, así como el código se puede estructurar de distinta manera dependiendo del lenguaje de programación, las VCs que provea el IDE, y el cómo se provean, también varía. Por ejemplo, un IDE puede usar la negrita para diferenciar si una palabra es o no una palabra reservada del lenguaje de programación, mientras que otro IDE puede usar la negrita para diferenciar entre funciones nativas del lenguaje y funciones no nativas. De aquí se desprende la necesidad de restringir la investigación a un IDE y lenguaje particular, y así poder conocer de antemano las VCs que se utilizan para esa combinación particular de IDE y lenguaje.

Dadas las diferencias en las herramientas y estrategias disponibles para programadores videntes y no videntes, resulta clave poder compararlas, lo que no siempre se puede realizar de manera directa. Es por esto que el estudio de Armaly, Rodeghero y McMillan [6] entrega, quizás, el mejor antecedente para entender cómo ambos grupos enfrentan el desafío de la comprensión de programas. En este estudio se utiliza el rastreo de ratón y encuestas para intentar aproximarse a lo que sería el rastreo de ojos en programadores videntes. Con este estudio, modelan el comportamiento en la exploración de código de los programadores ciegos, describiéndolo como un proceso descendente (*top-down*). Los autores especifican que los elementos de control de flujo del código suelen ser menos leídos que las firmas de los métodos. Los autores profundizaron en este último hallazgo con un nuevo estudio en 2018 [5]. Implementaron AudioHighlight, una herramienta que asocia etiquetas HTML a palabras reservadas de C, C++, o Java. Los programadores podían interactuar con estas etiquetas para desencadenar señales auditivas. El uso de AudioHighlight mejoró la calidad y velocidad en la comprensión de programas. Estos resultados refuerzan la relevancias de las palabras reservadas de los lenguajes como focos de atención. Dado que estas palabras presentan especial dificultad para los programadores ciegos, e importancia en la estructura del código, la presente tesis se enfoca en las señales del código asociadas a ellas.

En 2019, Falase, Siu y Follmer [14] diseñan un dispositivo que representa de manera física y táctil la estructura del código, en particular su indentación. Los autores registraron la experiencia de programadores ciegos que utilizaron el dispositivo en breves demostraciones prácticas, y estudiaron en profundidad, con un programador ciego, la incorporación del dispositivo en su flujo de trabajo. Los autores describen que los usuarios del dispositivo ya contaban con técnicas y procesos para la navegación de código, por lo que no les resultó satisfactorio incorporar una herramienta que requería modificar el uso de la mano. Sin embargo, los programadores ciegos que utilizaron el dispositivo reconocen que la herramienta aminoraba la carga auditiva a la que están sujetas, dada la utilización de lectores de pantalla. Estos hallazgos son relevantes ya que permiten presuponer una evaluación positiva de las NVCs que disminuyan la carga auditiva y, por otro lado, sugiere que, de ser necesario hardware complementario, este no debería concentrar el uso de las manos.

Para generar un estímulo no-visual capaz de llamar la atención de programadores no-videntes, se debe considerar cuáles son los focos de atención de la persona [24]. Es por esto que las conclusiones del estudio de Monares, Ochoa, Pino, Lopez y Noguera [23] proponen mecanismos potencialmente útiles para lograrlo en personas no-videntes. En este estudio se utilizaron mecanismos no convencionales para la transmisión de información contextual a

bomberos, en particular, se utilizaron señales basadas en olores y vibraciones. Este último mecanismo se implementó a través de un teléfono inteligente adosado al brazo, lo que podría no resultar práctico en el caso de programadores ciegos, por el tamaño y contexto, según lo discutido en poster de Falase, Siu y Follmer [14]. Sin embargo, existen dispositivos de menor tamaño capaces de generar señales a través de vibraciones, como lo son las pulseras inteligentes que usan este tipo de señal al recibir notificaciones o para complementar las alarmas. Cabe destacar que en los resultados del estudio, las vibraciones son detectadas en un 100 % de los casos, y que los participantes lograron diferenciar cinco patrones distintos de vibración sin mayor dificultad. Es en base a este estudio que la presente tesis evalúa el uso de señales hápticas como NVC.

Todos los tipos de señales no visuales que fueron utilizadas en los estudios discutidos en los párrafos anteriores son factibles como salida de la transformación de una VC. Algunas de estas han sido utilizadas al evaluar el caso de la exploración de código. Sin embargo otras, en particular las hápticas, aún no han sido evaluadas como NVCs. En base a los resultados de los estudios discutidos, estas señales podrían dar buenos resultados al someterse a pruebas de eficacia, eficiencia y satisfacción en los casos de escritura y lectura, ya que no sobrecargan al programador con estímulos auditivos que puedan opacar o confundirse con la salida de los lectores de pantalla. Es por lo anterior que en la presente tesis se diseñan y evalúan NVCs basadas en vibración, y se comparan contra señales basadas en audio.

## 2.2. Señales visuales en VSC

Las interfaces de usuario del IDE Visual Studio Code pueden ser configuradas utilizando temas de color (*Color Themes*). Estos permiten modificar tanto color como estilo de las componentes interactivas de la aplicación. Los temas de color afectan también el resaltado de sintaxis en el panel de edición de código. Sin embargo, el proceso que asigna un color y estilo a cada palabra requiere de más metainformación sobre lo que está escrito en el archivo. VSC realiza dicho proceso separándolo en dos partes: tokenización y tematización [12].

La tokenización corresponde al proceso a través del cual se separa el texto en una lista de tokens, y se le asigna un tipo a cada uno de ellos. El tipo suele estar directamente ligado al scope en el que se encuentra el token. Esto se hace ya que los lenguajes de programación tienen distintas formas de demarcar qué es lo que separa a sus elementos, y muchas veces depende del contexto. Luego, el IDE tiene que ser lo suficientemente flexible como para permitir que los criterios de tokenización sean definidos por las partes interesadas en que un lenguaje en particular sea resaltado de manera correcta.

El motor de tokenización de VSC funciona utilizando gramáticas TextMate, que corresponden a colecciones semiestructuradas de expresiones regulares Oniguruma<sup>6</sup>. Para el caso de tokenización semántica, esto es, cuando el proceso de tokenización depende del significado del token y no solo de su sintaxis, el ambiente provee un punto de contribución para utilizar

<sup>6</sup> Oniguruma es una biblioteca de expresiones regulares que admite múltiples codificaciones de caracteres. Cuenta con características de implementaciones de expresiones regulares de diferentes lenguajes de programación.



servidores de lenguaje.

Por otro lado, la tematización corresponde al proceso de mapeo de colores y estilos a los distintos tokens definidos en el proceso de tokenización. Este proceso utiliza las configuraciones provenientes del tema de color, pero también es capaz de soportar reglas adicionales. Estas configuraciones y reglas son selectores que asignan un resaltado a un token en función de el o los scopes asignados en el proceso de tokenización.

La figura 2.1 representa los elementos fundamentales de la implementación de VCs en VSC. En la figura se profundiza en los procesos que proveen el resaltado de sintaxis, ya sea puramente sintáctico o también semántico.

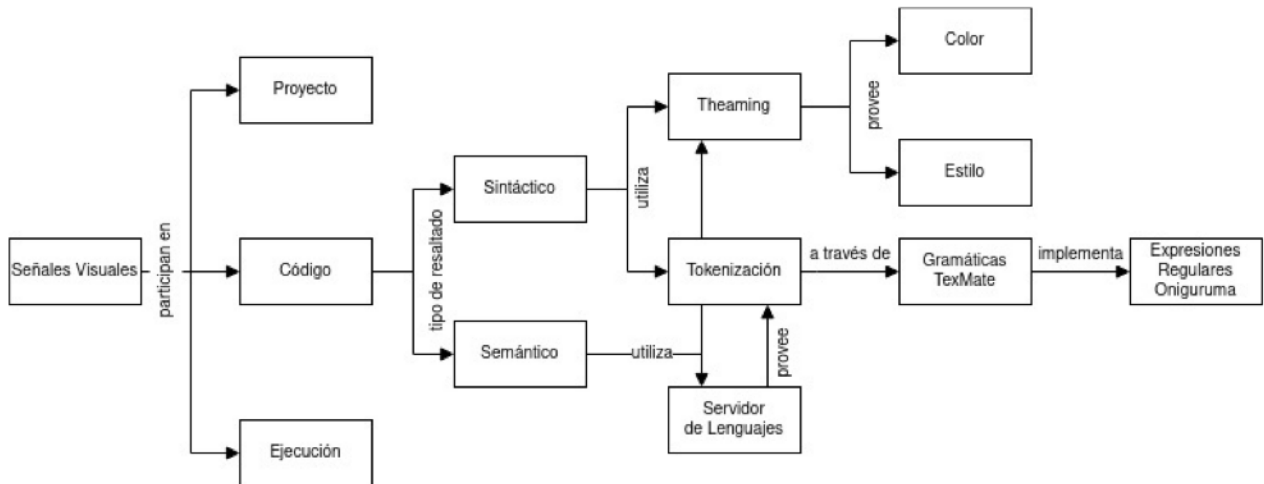


Figura 2.1: Diagrama de tipos de señales visuales en VSC y su funcionamiento.

VSC provee algunos temas por defecto, pero además permite a los usuarios crear y publicar extensiones en su mercado de software. Las extensiones son programas en JavaScript que permiten interactuar con el ambiente de desarrollo integrado a través de su API.

El IDE permite acceder a la información que utilizan los motores de tokenización y tematización utilizando la herramienta “Inspector de Scope”. Sin embargo, esta información sólo está disponible a través de la interfaz de usuario y no a través de la API, por lo que no es directo el cómo crear extensiones que utilicen el scope de los tokens como entrada.

Para efecto del estudio empírico de la presente tesis, se considerará como caso base de VCs encendidas al uso del tema de color “Dark+ (default dark)”, ya que es el tema por defecto, y contiene configuraciones para el resaltado de sintaxis del lenguaje Python. Para el caso base de VCs apagadas, se utilizará el tema “Plain (Dark)”, disponible instalando la extensión “Plain Theme 0.1.0” desde el mercado de extensiones de Visual Studio Code, ya que configura el editor de código para no utilizar resaltado de sintaxis. Las figuras 2.2 y 2.3 muestran cómo se aplican los temas “Dark+ (default dark)” y “Plain (Dark)” respectivamente, en una pieza de código tomada de un repositorio público<sup>7</sup>.

<sup>7</sup> Código tomado de <https://www.geeksforgeeks.org/python-program-for-bubble-sort/>

```
Users > robertopugafranco > Documents > sampel.py
1 # Python program for implementation of Bubble Sort
2
3 def bubbleSort(arr):
4     n = len(arr)
5
6     # Traverse through all array elements
7     for i in range(n-1):
8         # range(n) also work but outer loop will repeat one time more than needed.
9
10        # Last i elements are already in place
11        for j in range(0, n-i-1):
12
13            # traverse the array from 0 to n-i-1
14            # Swap if the element found is greater
15            # than the next element
16            if arr[j] > arr[j + 1] :
17                arr[j], arr[j + 1] = arr[j + 1], arr[j]
18
19    # Driver code to test above
20    arr = [64, 34, 25, 12, 22, 11, 90]
21
22    bubbleSort(arr)
23
24    print ("Sorted array is:")
25    for i in range(len(arr)):
26        print ("% d" % arr[i]),
```

Figura 2.2: Código fuente en Visual Studio Code utilizando el tema “Dark+ (default dark)”.

```
Users > robertopugafranco > Documents > sampel.py
1 # Python program for implementation of Bubble Sort
2
3 def bubbleSort(arr):
4     n = len(arr)
5
6     # Traverse through all array elements
7     for i in range(n-1):
8         # range(n) also work but outer loop will repeat one time more than needed.
9
10        # Last i elements are already in place
11        for j in range(0, n-i-1):
12
13            # traverse the array from 0 to n-i-1
14            # Swap if the element found is greater
15            # than the next element
16            if arr[j] > arr[j + 1] :
17                arr[j], arr[j + 1] = arr[j + 1], arr[j]
18
19    # Driver code to test above
20    arr = [64, 34, 25, 12, 22, 11, 90]
21
22    bubbleSort(arr)
23
24    print ("Sorted array is:")
25    for i in range(len(arr)):
26        print ("% d" % arr[i]),
27
```

Figura 2.3: Código fuente en Visual Studio Code utilizando el tema “Plain (Dark)”.

## 2.3. Síntesis

El estado del arte reconoce una falencia en la transmisión de información contextual del código a programadores ciegos, cuando se les compara con programadores videntes. En este capítulo se evidencia que, en gran medida, esto se debe al rol de las VCs en los IDEs, señales a las cuales las personas ciegas, por defecto, no pueden acceder.

Algunos estudios han explorado el uso de NVCs basadas en audio, como texto hablado, earcons y spearcons, pero solo lo han hecho en lenguajes de programación por bloques. Otros autores han explorado métodos no convencionales de transmisión de información, como olores, y vibraciones. Estos métodos no han sido comparados entre ellos en un mismo conjunto de pruebas. A pesar de esto, en otros contextos, las señales hápticas han sido validadas como mecanismos apropiados, o incluso superiores a métodos auditivos para la transmisión de información. Además, el estado del arte se enfoca en la exploración de código, por lo que destaca el estudio de Ludi, Simpson y Merchant [22] en que se evalúa diferenciadamente la escritura y la lectura, anticipando que se pueden medir diferencias de usabilidad según los casos de uso.

Con el fin de identificar mecanismos óptimos para la transmisión de información contextual del código a programadores ciegos, se debe explorar cómo se comparan las señales basadas en audio con respecto a métodos alternativos, como la retroalimentación háptica. Para realizar esta comparación en lenguajes de programación genéricos, se requiere de un modelo conceptual de casos de uso: una aproximación teórica a las instancias en que se debe desencadenar la transmisión de información contextual del código. De esta manera, basta con capturar el evento desencadenante para generar la señal no-visual deseada, quedando como parámetros de comparación la eficiencia, eficacia y satisfacción del usuario al utilizar cada tipo de señal particular.

La solución comprende un modelo basado en las contribuciones más recientes y guías de diseño fundadas en un caso de estudio exploratorio que lo utiliza. El modelo permite identificar las instancias en que se debe desencadenar una NVC particular, mientras que el caso de estudio único expande el conocimiento sobre cómo distintas NVCs se comparan en los distintos casos de uso. Las guías de diseño recogen, según la experiencia empírica y basada en la usabilidad, cuales son las NVCs óptimas en cada caso de uso. Además, presentan de manera estructurada la información cualitativa con respecto al impacto de las señales del código en los procesos de desarrollo desde la perspectiva de programadores ciegos.

# Capítulo 3

## Modelo Conceptual de Casos de Uso

En el presente capítulo se propone un modelo conceptual de casos de uso. Esto se hace en base a la discusión del trabajo relacionado presentado en el capítulo anterior, proponiendo criterios diferenciadores. Posteriormente, se explora la validez del modelo a través de una prueba pareada entre sujetos.

### 3.1. Modelo propuesto

En el capítulo anterior se estudia el impacto de NVCs en los procesos de desarrollo, enfocados en el caso de la exploración de una base de código desconocida. Sin embargo, no resulta claro el por qué no se ha profundizado, de la misma manera, otros casos de uso en que las señales del código pueden resultar relevantes.

Cuando nos enfrentamos a una pieza de código desconocida en un lenguaje de programación familiar, las señales visuales del código permiten al programador encontrar fácilmente palabras reservadas para entender la estructura general de la pieza. Por ejemplo, identificar cuáles son sus dependencias, cuál es el hilo de ejecución principal, cuáles son los métodos definidos por el programador, o cuáles son los tipos primitivos que se utilizan.

Cuando el lenguaje de programación no es familiar, entonces las VCs permitirían identificar categorías entre los elementos del lenguaje, por ejemplo, por su coloración. Esto podría hacer a un programador intuir que pueden tener comportamientos similares en la ejecución, o bien pertenecer a un mismo nivel de alcance.

Cuando un programador explora una pieza de código que fue desarrollada por él mismo, por ejemplo, al buscar un error luego de un problema en la compilación o ejecución, las VCs permiten identificar rápidamente problemas de escritura en palabras reservadas del lenguaje. Un ejemplo puede ser al escribir `retrun` en vez de `return` o `true` en vez de `True`. Además de esto, y similar al caso anterior, las VCs también nos permiten identificar conformidad con la sintaxis del lenguaje cuando escribimos en un lenguaje de programación conocido. Si el programador conoce de antemano que la palabra “return” es una palabra reservada del lenguaje, y que esta se resalta de una manera particular, entonces, cuando escriba esa palabra, esperará que se resalte de la forma que conoce.

Cuando escribimos en una base de código conocida, y las funciones definidas por el usuario se resaltan de forma distintas de los métodos primitivos y otros elementos del lenguaje, entonces se puede reconocer rápidamente si se aplicó la función que se esperaba invocar, o si se cometió un error de escritura. En cambio, cuando se escribe sobre una base de código desconocida, las VCs permiten identificar si se introdujo algún problema de sintaxis que antes no estaba presente en la pieza de código. Por ejemplo, cuando se deja un símbolo de doble comillas sobrante, todo lo que esté a la derecha de dicho símbolo se resaltará como String en ciertos lenguajes de programación. Otro caso puede ser el cambio en el resaltando que introducen los cambios de alcance en lenguajes embebidos. Por ejemplo, en React, esto permite predecir si un conjunto de símbolos se interpretarían como JSX o no.

En base a los casos descritos, se puede ver que las VCs no siempre entregan el mismo tipo de información al programador, y la relevancia que ellas tienen dependen del caso en el que resulten relevantes. Por esto, se define un modelo conceptual de casos de uso de señales de código, con el objetivo de evaluar si existen diferencias en la eficiencia y eficacia de estas señales para transmitir información, y la relevancia de su rol percibida por el programador. La figura 3.1 a continuación, presenta la organización conceptual de este modelo, y los distintos roles que pueden cumplir las señales del código según cada tipo de caso de uso.

Este modelo separa los casos de uso según dos criterios. Primero, según la acción principal que realiza el programador: los casos de lectura y los casos de escritura. Segundo, estos casos se subdividen según el conocimiento previo que tenga el programador sobre la base de códigos que está trabajando: bases de código conocidas y desconocidas. Por último, en un tercer nivel, se presentan los roles que pueden cumplir las señales del código dado el caso de uso.

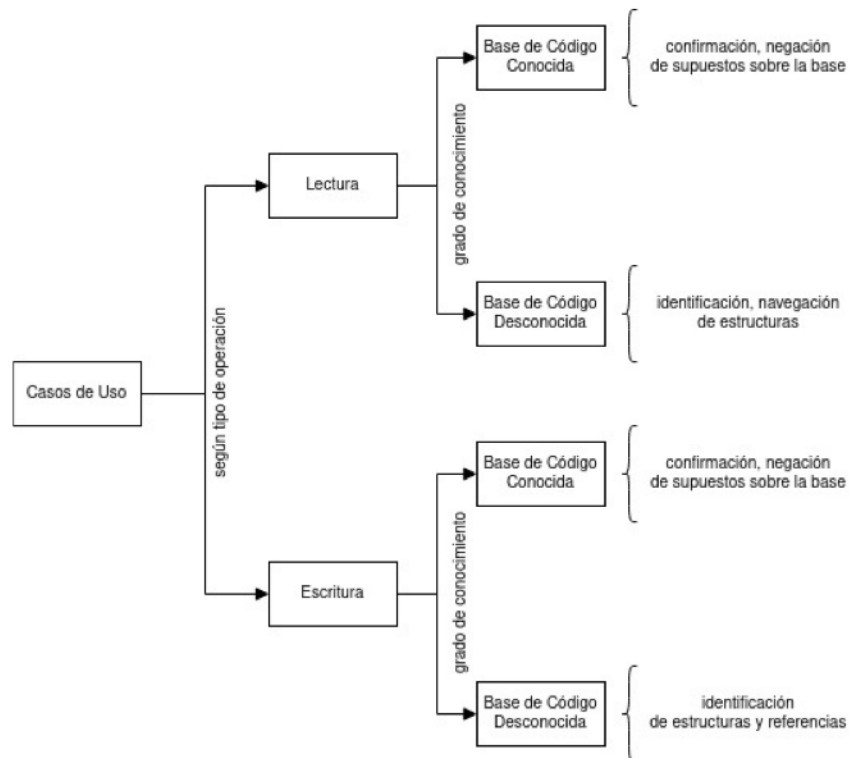


Figura 3.1: Modelo conceptual de casos de uso y roles de las señales del código.

El modelo asume que las acciones que un programador puede tomar sobre una base de código corresponden a la lectura y escritura (o sobreescritura) de ella. Además asume como una base de código desconocida a aquella que no ha sido explorada por el programador que debe realizar una acción sobre ella. Análogamente, se asume una base de código conocida a aquella que ya fue explorada previamente, o fue escrita por el programador, antes de realizar una acción sobre ella. Los estudios descritos en los siguientes capítulos utilizan este modelo para la definición, aplicación y evaluación de los experimentos realizados.

## **3.2. Validación experimental del Modelo conceptual de casos de uso**

Para determinar la validez del modelo conceptual de casos de usos, se diseña un estudio empírico entre sujetos. En este estudio, los participantes deben resolver un conjunto de desafíos de escritura y lectura de código Python, en el ambiente de desarrollo integrado VSC. Una mitad de los desafíos se deben resolver con las NVCs apagadas, esto es, utilizando el tema de color “Plain (Dark)”, y la otra mitad se debe resolver con las NVCs encendidas, esto es, utilizando el tema de color por defecto “Dark+ (default dark)”.

### **3.2.1. Hipótesis**

Para este experimento, se definen las siguientes hipótesis, cuya demostración, en su conjunto, validan el modelo conceptual de caso de usos propuesto.

- H1. El nivel de esfuerzo necesario para completar tareas de escritura y lectura de código es menor al realizarlas con señales visuales del código encendidas, que al realizarlas con dichas señales apagadas.
- H2. El nivel de esfuerzo necesario es distinto según el tipo de tarea y el conocimiento de la base de código. Esto considera tipos de tareas de escritura y lectura, y bases de código conocidas y desconocidas.

### **3.2.2. Diseño experimental**

Se diseñó una prueba pareada, específicamente para este estudio, que permitiera contrastar el nivel de esfuerzo necesario al leer y escribir código, según las VCs estén encendidas o apagadas. Para lograr esto, se creó un conjunto de desafíos de programación. En los desafíos de lectura, los sujetos de prueba debían encontrar cadenas de caracteres específicos, e identificar si, según su alcance en el código, corresponden a palabras reservadas del lenguaje. En los desafíos de escritura, se debía reemplazar una cadena de caracteres específica por una palabra reservada del lenguaje, o corregir un error de sintaxis.

El conjunto de sujetos que participaron en estas pruebas se dividió en dos grupos. Uno de estos grupos realizó los desafíos con VCs encendidas en la primera mitad de problemas,

y apagadas en la segunda. El otro grupo realizó el desafío con VCs apagadas en la primera mitad, y encendidas en la segunda.

Esta prueba consta de ocho partes. Las partes 1, 2 y 3 son tipos de tareas análogas a las partes 4, 5 y 6, pero se diferencian, al momento de resolverlas, según la configuración de VCs que se debe utilizar. A continuación se describen las partes de la prueba.

- Parte 0: Se recopila información sobre el participante con respecto a: condiciones visuales, nivel de experiencia programando, familiaridad con el lenguaje de programación Python, familiaridad con la aplicación Visual Studio Code, familiaridad con otros ambientes de desarrollo integrado.
- Parte 1: Sobre la lectura de una base de código desconocida. En esta sección se presenta un programa desconocido sobre el cual se deben encontrar palabras o conceptos clave del lenguaje Python. Esta sección contiene tres requerimientos y mide el nivel de esfuerzo necesario para resolverlos.
- Parte 2: Sobre la lectura de una base de código conocida. En esta sección se presenta el mismo programa de la Parte 1, pero con algunas modificaciones. Nuevamente se deben encontrar palabras o conceptos clave del lenguaje Python. Esta sección contiene tres requerimientos y mide el nivel de esfuerzo necesario para resolverlos.
- Parte 3: Sobre la escritura de código. En esta sección se presenta un programa que debe ser modificado para satisfacer un requerimiento. Esta sección contiene un requerimiento y mide el nivel de esfuerzo necesario para resolverlo.
- Parte 4: Sobre la lectura de una base de código desconocida. En esta sección se presenta un programa desconocido sobre el cual se deben encontrar palabras o conceptos clave del lenguaje Python. Esta sección contiene tres requerimientos y mide el nivel de esfuerzo necesario para resolverlos.
- Parte 5: Sobre la lectura de una base de código conocida. En esta sección se presenta el mismo programa de la Parte 4, pero con algunas modificaciones. Nuevamente se deben encontrar palabras o conceptos clave del lenguaje Python. Esta sección contiene tres requerimientos y mide el nivel de esfuerzo necesario para resolverlos.
- Parte 6: Sobre la escritura de código. En esta sección se presenta un programa que debe ser modificado para satisfacer un requerimiento. Esta sección contiene un requerimiento y mide el nivel de esfuerzo necesario para resolverlo.
- Parte 7: Sobre el experimento. En esta sección se presentan 10 afirmaciones con respecto al efecto de las señales visuales del código. Se mide qué tan de acuerdo o en desacuerdo se está con cada una de las afirmaciones.

La tabla 3.1 a continuación, muestra la configuración de VCs según la parte del instrumento y el grupo de aplicación.

Tabla 3.1: Esquema de configuración de señales visuales según partes del instrumento y grupos.

| Partes  | Grupo A    | Grupo B    |
|---------|------------|------------|
| 1, 2, 3 | Apagadas   | Encendidas |
| 4, 5, 6 | Encendidas | Apagadas   |
| 0, 7    | No Aplica  |            |

### 3.2.3. Instrumentos

En razón de la pandemia por Coronavirus SARS-CoV-2, se diseñaron instrumentos capaces de ser aplicados de manera remota. Se crearon dos formatos de formularios web, utilizando la plataforma Google Forms: el primer formato, formato A, instruye al participante a utilizar primero VCs desactivadas, y luego activadas; el segundo formato, formato B, instruye al participante a utilizar primero VCs activadas y luego desactivadas. Las preguntas de ambos formatos son idénticas, y hacen referencia a un mismo conjunto de archivos de código Python. Estos archivos fueron desarrollados a medida para el experimento. Las preguntas fueron elaboradas a medida, y se validaron en una aplicación piloto con 5 estudiantes universitarios.

Cada formato de formulario contiene diez secciones. Las secciones 1 y 2 corresponden a una breve introducción a la prueba y Visual Studio Code. En estas, se solicita la dirección de correo electrónico, garantizando solo una respuesta por participante, se presenta un cuestionario de autorreporte, y se enseña cómo iniciar un espacio de trabajo con los archivos necesarios para la prueba. La sección 3 es análoga a la parte 0 del diseño experimental, en que se recogen antecedentes sobre los participantes. Las secciones 4, 5 y 6 son análogas a las partes 1, 2 y 3 del diseño experimental. El formato A solicita VCs desactivadas para resolver estas secciones, mientras que el formato B solicita VCs activadas. Las secciones 7, 8 y 9 son análogas a las partes 4, 5 y 6 del diseño experimental. El formato A solicita VCs activadas para resolver estas secciones, mientras que el formato B solicita VCs desactivadas. Finalmente, la sección 10 implementa la parte 7 del diseño experimental, solicitando al participante retroalimentación con respecto a la totalidad de la prueba. La figura 3.2 esquematiza la implementación del diseño experimental a través de las secciones de los formularios, la cantidad de requerimientos en cada sección, el tipo de actividad realizada, los archivos necesarios de la base de código, y las configuraciones de VCs utilizadas por cada grupo. Las flechas indican que el archivo necesario en la sección siguiente, es similar, pero con leves modificaciones.



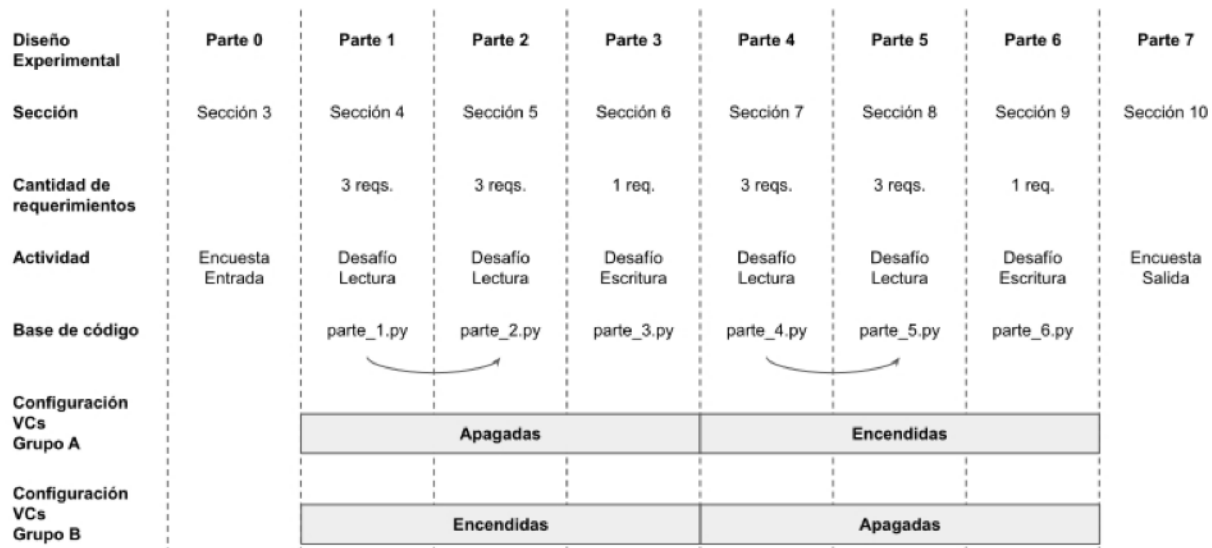


Figura 3.2: Esquema resumen sobre la implementación del diseño experimental.

El anexo A.1.1 contiene una transcripción en texto del formulario, en su formato A. Este formulario fue utilizado con el grupo A. En el anexo A.1.2 se adjuntan capturas de pantalla de formulario, en su formato A. El formato B es análogo al presentado en anexos, pero se intercambian las instrucciones con respecto al tema de color a utilizar, esto es, partes 2 y 7.

### 3.2.4. Participantes

Esta prueba se aplicó a 70 estudiantes de Ingeniería Civil de la Universidad de Chile, separados en tres grupos. Un primer grupo de 11 personas que se encontraban cursando “Diseño de Sistemas Interactivos” del Departamento de Ciencias de la Computación. Para estar cursando dicha asignatura es necesario haber aprobado cursos introductorios de programación. Este grupo se denominó “HCI” y se diferencia de los siguientes porque se asumió un mayor nivel de experiencia programando. Un segundo grupo de 33 personas del curso “Herramientas Computacionales para Ingeniería y Ciencias”. Este curso no tiene requisitos, pues es del primer semestre del plan común de ingeniería. Un tercer grupo de 25 personas, también del curso “Herramientas Computacionales para Ingeniería y Ciencias”. Todos los grupos están compuestos por hombres y mujeres, mayores de 18 años.

Se eligió este grupo de prueba, ya que, dadas las condiciones para la aplicación, este grupo cumplía con los requisitos de ser personas mayores de edad con conocimientos básicos de programación, capaces de participar en una prueba remota en un horario y fecha particular. Además, estos cursos son dictados por el profesor guía de la presente tesis, por lo que eran de fácil contacto y coordinación.

Antes de la aplicación, los participantes fueron informados de los objetivos del experimento, así como también que sus resultados serían publicados en la presente tesis. Todos los sujetos participaron del experimento de manera libre y voluntaria. Se obtuvo consentimiento

de los participantes para el uso anonimizado de los datos recolectados. Para incentivar la participación del estudio, se sorteó entre los participantes dos tarjetas de regalo, con monto de \$10.000 cada una, por un trabajo de entre 20 a 40 minutos.

En base al cuestionario de autorreporte de los formularios web y el grupo de aplicación, se diferencié a los participantes según los siguientes criterios:

- HCI: clasifica al participante según si pertenece a la primera sesión de aplicación del experimento.
- Impairment: clasifica al participante según si declara tener algún tipo de condición visual (miopía, hipermetropía, astigmatismo, daltonismo, acromatopsia, ceguera parcial o total).
- Prog: clasifica al participante según si declara tener un nivel de experiencia 2 o más programando, en escala de Likert de 1 a 5, en que 1: “Nunca he programado” y 5: “Programo regularmente”.

A continuación, la tabla 3.2 presenta la distribución de participantes por grupo y criterios de diferenciación. En dicha tabla: N corresponde a la cantidad de participantes del grupo, HCI a la cantidad de participantes integrantes del curso “Diseño de Sistemas Interactivos”, Prog a la cantidad de participantes que declaran tener experiencia programando, e Imp a la cantidad de participantes que declara tener alguna condición visual no discapacitante.

Tabla 3.2: Distribución de participantes prueba pareada.

| Aplicación   | Grupo A |     |      |     | Grupo B |     |      |     |
|--------------|---------|-----|------|-----|---------|-----|------|-----|
|              | N       | HCI | Prog | Imp | N       | HCI | Prog | Imp |
| <b>1</b>     | 5       | 5   | 5    | 4   | 6       | 6   | 6    | 2   |
| <b>2</b>     | 17      | 0   | 5    | 9   | 16      | 0   | 7    | 5   |
| <b>3</b>     | 13      | 0   | 6    | 4   | 13      | 0   | 5    | 7   |
| <b>Total</b> | 35      | 5   | 17   | 17  | 35      | 6   | 18   | 14  |

### 3.2.5. Aplicación

Como requisito previo para la aplicación de la prueba, se solicitó por escrito a los participantes que instalaran la versión 1.57 de Visual Studio Code, la más reciente, desde su centro de descargas<sup>8</sup>. Además, dado el contexto de educación remota en el que estaban involucrados, se solicitó que, en el horario de una de sus clases con el profesor Francisco Gutierrez, se conectaran a una sala en Zoom para la aplicación de la prueba, de manera remota.

La prueba se aplicó en tres instancias, una por cada grupo. En ellas, al iniciar la aplicación, se distribuyó a los participantes aleatoriamente según su rol único nacional (R.U.N.), en grupos A y B. Luego, se envía el enlace al formulario formato A a los miembros del grupo

<sup>8</sup> <https://visualstudio.microsoft.com/es/downloads/>

A, y el enlace al formulario formato B a los miembros del grupo B. Se indicó a los participantes que debían seguir las instrucciones allí señaladas y completar los campos de respuesta del formulario, y que, al finalizar, enviaran sus respuestas. No se proporcionó información adicional y solo se respondieron preguntas de manera pública en la sala de Zoom con respecto a conceptos que no se entendieran. En cada instancia se dispuso de 45 minutos para la realización de la actividad.

La primera instancia de aplicación fue a los estudiantes especializados en computación, el grupo “HCI”. La segunda y tercera instancias fueron a los estudiantes del plan común de ingeniería. Todas las instancias se realizaron el mismo día, y el encargado de la aplicación fue el autor.

### **3.2.6. Análisis de respuestas**

Para el análisis de respuestas se consideró como “Experiencia A” a las partes del experimento realizadas con las VCs apagadas, es decir, las partes 1, 2, y 3 del formato A, y las partes 4, 5 y 6 del formato B. Análogamente, se consideró como “Experiencia B” las partes del experimento realizadas con las VCs encendidas, es decir, las partes 1, 2, y 3 del formato B, y las partes 4, 5 y 6 del formato A.

Se organizan los datos de esfuerzo en mediciones, tal que:

- Medición A.1.1: Corresponde al esfuerzo necesario para completar el primer requerimiento de la parte 1 en el formato A o del primer requerimiento de la parte 4 del formato B.
- Medición A.1.2: corresponde al esfuerzo necesario para completar el segundo requerimiento de la parte 1 en el formato A o del segundo requerimiento de la parte 4 del formato B.
- Medición A.1.3: corresponde al esfuerzo necesario para completar el tercer requerimiento de la parte 1 en el formato A o del tercer requerimiento de la parte 4 del formato B.
- Medición A.1: corresponde a la suma de las mediciones A.1.1, A.1.2, A.1.3, las cuales corresponden a los requerimientos de lectura de base de código desconocida con VCs inactivas.
- Medición A.2.1: corresponde al esfuerzo necesario para completar el primer requerimiento de la parte 2 en el formato A o del primer requerimiento de la parte 5 del formato B.
- Medición A.2.2: corresponde al esfuerzo necesario para completar el segundo requerimiento de la parte 2 en el formato A o del segundo requerimiento de la parte 5 del formato B.
- Medición A.2.3: corresponde al esfuerzo necesario para completar el tercer requerimiento de la parte 2 en el formato A o del tercer requerimiento de la parte 5 del formato B.

- Medición A.2: corresponde a la suma de las mediciones A.2.1, A.2.2, A.2.3, las cuales corresponden a los requerimientos de lectura de base de código conocida con VCs inactivas.
- Medición A.L: corresponde al promedio de los promedios de las mediciones A1.1, A1.2 y A.1.3 y las mediciones A.2.1, A.2.2 y A.2.3. Estos corresponden a los requerimientos de lectura con VCs inactivas.
- Medición A.3: corresponde al esfuerzo necesario para completar el requerimiento de la parte 3 en el formato A o del requerimiento de la parte 6 del formato B.
- Medición A: corresponde a la suma de las mediciones A.1, A.2, y A.3, las cuales corresponden a lectura y escritura con VCs inactivas. Refleja la totalidad del esfuerzo necesario para la “Experiencia A”.
- Medición B.1.1: corresponde al esfuerzo necesario para completar el primer requerimiento de la parte 1 en el formato B o del primer requerimiento de la parte 4 del formato A.
- Medición B.1.2: corresponde al esfuerzo necesario para completar el segundo requerimiento de la parte 1 en el formato B o del segundo requerimiento de la parte 4 del formato A.
- Medición B.1.3: corresponde al esfuerzo necesario para completar el tercer requerimiento de la parte 1 en el formato B o del tercer requerimiento de la parte 4 del formato A.
- Medición B.1: corresponde a la suma de las mediciones B.1.1, B.1.2, B.1.3, las cuales corresponden a los requerimientos de lectura de base de código desconocida con VCs activas.
- Medición B.2.1: corresponde al esfuerzo necesario para completar el primer requerimiento de la parte 2 en el formato B o del primer requerimiento de la parte 5 del formato A.
- Medición B.2.2: corresponde al esfuerzo necesario para completar el segundo requerimiento de la parte 2 en el formato B o del segundo requerimiento de la parte 5 del formato A.
- Medición B.2.3: corresponde al esfuerzo necesario para completar el tercer requerimiento de la parte 2 en el formato B o del tercer requerimiento de la parte 5 del formato A.
- Medición B.2: corresponde a la suma de las mediciones B.2.1, B.2.2, B.2.3, las cuales corresponden a los requerimientos de lectura de base de código conocida con VCs activas.
- Medición B.L: corresponde al promedio de los promedios de las mediciones B1.1, B1.2 y B.1.3 y las mediciones B.2.1, B.2.2 y B.2.3. Estos corresponden a los requerimientos de escritura con VCs activas.
- Medición B.3: corresponde al esfuerzo necesario para completar el requerimiento de la parte 3 en el formato B o del requerimiento la parte 6 del formato A.
- Medición B: corresponde a la suma de las mediciones B.1, B.2, y B.3, las cuales corresponden a lectura y escritura con VCs activas. Refleja la totalidad del esfuerzo necesario para la “Experiencia B”.

Estas mediciones permiten el análisis estadístico bivariado, según los casos de uso, del esfuerzo percibido utilizando VCs activas o inactivas. La tabla 3.3 a continuación presenta los requerimientos de cada formato y las mediciones con las que se asocian.

Tabla 3.3: Requerimientos considerados en cada formato para cada medición.

| Medición |     | Formato A                   | Formato B                   |                         |                         |
|----------|-----|-----------------------------|-----------------------------|-------------------------|-------------------------|
| A        | A.1 | A.1.1                       | requerimiento 1 parte 1     | requerimiento 1 parte 4 |                         |
|          |     | A.1.2                       | requerimiento 2 parte 1     | requerimiento 2 parte 4 |                         |
|          |     | A.1.3                       | requerimiento 3 parte 1     | requerimiento 3 parte 4 |                         |
|          | A.2 | A.2.1                       | requerimiento 1 parte 2     | requerimiento 1 parte 5 |                         |
|          |     | A.2.2                       | requerimiento 2 parte 2     | requerimiento 2 parte 5 |                         |
|          |     | A.2.3                       | requerimiento 3 parte 2     | requerimiento 3 parte 5 |                         |
|          | A.3 | requerimiento único parte 3 | requerimiento único parte 6 |                         |                         |
|          | B   | B.1                         | B.1.1                       | requerimiento 1 parte 4 | requerimiento 1 parte 1 |
|          |     |                             | B.1.2                       | requerimiento 2 parte 4 | requerimiento 2 parte 1 |
| B.1.3    |     |                             | requerimiento 3 parte 4     | requerimiento 3 parte 1 |                         |
| B.2      |     | B.2.1                       | requerimiento 1 parte 5     | requerimiento 1 parte 2 |                         |
|          |     | B.2.2                       | requerimiento 2 parte 5     | requerimiento 2 parte 2 |                         |
|          |     | B.2.3                       | requerimiento 3 parte 5     | requerimiento 3 parte 2 |                         |
| B.3      |     | requerimiento único parte 6 | requerimiento único parte 3 |                         |                         |

No se tomó en consideración si las respuestas a los desafíos planteados eran correctas o incorrectas, ni el tiempo requerido para resolverlos. Esto se debe a que el sistema de aplicación del instrumento, Google Forms, no permite registros parciales de la actividad del usuario. Es por esto que no se tienen registros de cuánto tiempo tomó el encontrar cada respuesta, o si la respuesta enviada fue la primera encontrada. Además, como cada formato incluye ambas experiencias, solo se tiene registro de los tiempos totales del formato, esto es, “Experiencia A” y “Experiencia B” concatenados.

### 3.2.7. Resultados e interpretación

En el experimento se controlaron dos variables principales: la configuración de VCs utilizada, y el tipo de caso de uso según el modelo conceptual propuesto. A continuación se presentan e interpretan los resultados del experimento. Se dividen en generales, esto es, solo diferenciando por configuración de VCs; y por caso de uso, en que se diferencia tanto por configuración de VCs como por tipo de caso de uso.

### 3.2.7.1. Generales

Se consideran como resultados generales del experimentos a las medidas estadísticas obtenidas de los indicadores de esfuerzo, sin diferenciar por caso de uso. Por esto, las mediciones más relevantes para este análisis serán la Medición A y Medición B.

Para realizar pruebas estadísticas sobre las Mediciones A y B, primero se intentó corroborar la normalidad de los datos. Se aplicó una prueba Shapiro-Wilk, pero no se rechazó la hipótesis nula. Por esto, se procedió a comparar las Mediciones aplicando una prueba de los rangos con signos de Wilcoxon. Esta prueba corresponde a la versión no paramétrica de la prueba t de Student tradicional.

Los resultados de la prueba de los rangos con signos de Wilcoxon sobre la Mediciones A y B son los siguientes:  $V = 1942.5$ ,  $p\text{-value} = 1.543\text{e-}09$ ,  $95\%CI = [0.366, 0.6744]$ ,  $\delta = 0.5379$ . En base a esto, se deduce que las señales de código fuente tienen un efecto estadísticamente significativo en la disminución del esfuerzo percibido por parte de los programadores, lo que es consistente con la literatura [30].

Se aplicó una prueba de los rangos con signos de Wilcoxon a los subconjuntos de las Mediciones A y B, diferenciados por los clasificadores HCI, Prog e Imp. Las tabla 3.4 y 3.5 indican las medianas de cada subconjunto de cada Medición. La tabla 3.6 expone los resultados de la prueba, siendo “V” el valor del estadístico de la prueba de los rangos con signos de Wilcoxon, “p-value” la probabilidad de encontrar resultados como los obtenidos si la hipótesis nula fuera cierta, “95 %CI” el intervalo de confianza al 95 %, y “ $\delta$ ” el valor del delta de Cliff.

Tabla 3.4: Mediana Medición A por clasificadores.

|      | Medición A        | Me |
|------|-------------------|----|
| HCI  | pertenecientes    | 15 |
|      | no pertenecientes | 17 |
| Prog | pertenecientes    | 16 |
|      | no pertenecientes | 17 |
| Imp  | pertenecientes    | 17 |
|      | no pertenecientes | 19 |

Tabla 3.5: Mediana Medición B por clasificadores.

|      | Medición B        | Me |
|------|-------------------|----|
| HCI  | pertenecientes    | 8  |
|      | no pertenecientes | 10 |
| Prog | pertenecientes    | 10 |
|      | no pertenecientes | 11 |
| Imp  | pertenecientes    | 11 |
|      | no pertenecientes | 10 |

Tabla 3.6: Indicadores estadísticos de Mediciones A y B por clasificadores.

|             |                          | $V$    | $p - value$ | 95 % $CI$      | $\delta$ |
|-------------|--------------------------|--------|-------------|----------------|----------|
| <b>HCI</b>  | <b>pertenecientes</b>    | 63.5   | 0.007       | 0.4223, 0.942  | 0.8017   |
|             | <b>no pertenecientes</b> | 1326.5 | 6.30E-08    | 0.2889, 0.6455 | 0.4872   |
| <b>Prog</b> | <b>pertenecientes</b>    | 219    | 0.0003      | 0.2389, 0.792  | 0.5784   |
|             | <b>no pertenecientes</b> | 481    | 5.03E-05    | 0.1782, 0.6434 | 0.4398   |
| <b>Imp</b>  | <b>pertenecientes</b>    | 0.4398 | 1.79E-05    | 0.3094, 0.7723 | 0.5869   |
|             | <b>no pertenecientes</b> | 574    | 2.23E-05    | 0.288, 0.69633 | 0.5214   |

Los resultados evidencian que la variación de la magnitud del esfuerzo percibido por el programador es mayor cuando este tiene experiencia programando. Esto se puede interpretar como que el programador, conforme gana experiencia desarrollando, es más dependiente en el uso de los recursos adicionales que provee el IDE. Por esto, al desactivar las VCs, las tareas propuestas se perciben como excesivamente más demandantes.

Con respecto al efecto de condiciones visuales no discapacitantes, se evidencia que las personas con condiciones visuales son menos dependientes del uso de las VCs para la resolución de tareas. Esto porque al desactivarlas, la variación del esfuerzo percibido es menor que en sus pares que no reportan condiciones visuales.

En base a los resultados obtenidos, se valida la hipótesis H1 de este experimento, ya que efectivamente, el nivel de esfuerzo necesario completar los desafíos se ve disminuído al utilizar VCs. Está disminución del esfuerzo se sostiene ante todos los tipos de actividades realizadas, y todos los clasificadores utilizados.

### 3.2.7.2. Por caso de uso

Se consideran como resultados por caso de uso del experimentos a las medidas estadísticas obtenidas de los indicadores de esfuerzo, diferenciando por caso de uso. Por esto, las mediciones relevantes para este análisis serán: Medición A1, Medición A2, Medición AL, Medición A3, Medición B1, Medición B2, Medición BL y Medición B3.

La tabla 3.7 presenta los indicadores estadísticos sobre las Mediciones relevantes para esta subsección de resultados. La columna “ $M$ ” indica la media aritmética. La columna “ $SD$ ” indica la desviación estándar. La columna “ $Me$ ” indica la mediana.

Tabla 3.7: Indicadores estadísticos de Mediciones por caso de uso.

| <b>Medición</b> | <i>M</i> | <i>SD</i> | <i>Me</i> |
|-----------------|----------|-----------|-----------|
| <b>A1</b>       | 7.1      | 2.88      | 7         |
| <b>A2</b>       | 6.9571   | 2.9754    | 6.5       |
| <b>AL</b>       | 2.3429   | 0.9316    | 2.3333    |
| <b>A3</b>       | 2.6143   | 1.1458    | 2         |
| <b>B1</b>       | 4.4714   | 2.1784    | 4         |
| <b>B2</b>       | 4.6143   | 2.0522    | 4         |
| <b>BL</b>       | 1.5243   | 0.6641    | 1.3333    |
| <b>B3</b>       | 2.2857   | 1.2411    | 2         |

Para comprobar que la variación en el esfuerzo percibido se debe al tipo de actividad del caso de uso, se realizó la prueba de los rangos con signos de Wilcoxon a los siguientes pares de mediciones: Medición AL con Medición A3, esto corresponde a comparar lectura y escritura con NVCs apagadas; y Medición BL con B3, esto corresponde a comparar lectura y escritura con NVCs encendidas. La tabla 3.8 expone los resultados de la prueba.

Tabla 3.8: Indicadores estadísticos de Mediciones por escritura o lectura.

|                    |                    | <i>V</i> | <i>p - value</i> | 95 % <i>CI</i>   | $\delta$ |
|--------------------|--------------------|----------|------------------|------------------|----------|
| <b>Medición AL</b> | <b>Medición A3</b> | 773.5    | 7.51E-02         | -0.2793, 0.1036  | -0.0912  |
| <b>Medición BL</b> | <b>Medición B3</b> | 141.5    | 6.18E-07         | -0.4972, -0.1199 | -0.3212  |

Los estadísticos de las tablas 3.7 y 3.8 muestran que cuando las NVCs están activadas, existe un efecto significativo de disminución del esfuerzo percibido, atribuible al tipo de actividad del caso de uso. En particular, el caso de uso de escritura requiere un esfuerzo significativamente menor que el promedio de los esfuerzos percibidos en los casos de lectura. Con esto, se valida parcialmente la hipótesis H2 del presente experimento.

Para comprobar que la variación en el esfuerzo percibido se debe al grado de familiaridad con las base de código del caso de uso, se realizó la prueba de los rangos con signos de Wilcoxon a los siguientes pares de mediciones: Medición A1 con Medición A2, esto corresponde a comparar lectura de base conocida y desconocida con NVCs apagadas; y Medición B1 con B2, esto corresponde a comparar lectura de base conocida y desconocida con NVCs encendidas. La tabla 3.9 expone los resultados de la prueba.

Tabla 3.9: Indicadores estadísticos de Mediciones por familiaridad con la base de código.

|                    |                    | <i>V</i> | <i>p - value</i> | 95 % <i>CI</i>  | $\delta$ |
|--------------------|--------------------|----------|------------------|-----------------|----------|
| <b>Medición A1</b> | <b>Medición A2</b> | 615      | 4.12E-01         | -0.1501, 0.2277 | 0.0402   |
| <b>Medición B1</b> | <b>Medición B2</b> | 268      | 1.96E-01         | -0.2652, 0.0999 | -0.0855  |



Los estadísticos de las tablas 3.7 y 3.9 no muestran una relación aparente entre el nivel de familiaridad con la base de código con respecto al esfuerzo percibido al realizar tareas de lectura. Las medias aritméticas de las Medidas A1 y A2 muestran una disminución del esfuerzo al conocer la base de código, mientras que las medias aritméticas de las Medidas B1 y B2 muestran lo contrario. Además, los valores de significancia, para el test de Wilcoxon aplicado a las Medidas A1 y A2 y Medidas B1 y B2, confirman el bajo grado de significancia. Con esto, se debe descartar parcialmente la hipótesis H2 del presente experimento. El grado de familiaridad con la base de código no tiene una influencia estadísticamente significativa sobre el esfuerzo necesario al realizar tareas de escritura y lectura de código.

### **3.2.8. Conclusiones**

Las VCs efectivamente reducen el esfuerzo percibido, tanto en actividades de lectura y escritura de código. La magnitud de dicha variación depende del nivel de experiencia programando de la persona. Las personas con mayor nivel de experiencia programando reportan un mayor cambio en el esfuerzo percibido en las actividades, cuando las VCs pasan de activas a inactivas o viceversa. Inversamente, las personas con menor nivel de experiencia programando o con condiciones visuales no discapacitantes, reportan un menor cambio en el esfuerzo percibido en las actividades, cuando las VCs pasan de activas a inactivas o viceversa. Por ende, el nivel de esfuerzo necesario es distinto dependiendo de si la tarea es de escritura o de lectura, y esto es especialmente notorio cuando es mayor el nivel de experiencia programando.

En el experimento diseñado, al considerar como indicador la suma de los esfuerzos reportados en cada pregunta, se preponderan los casos de lectura por sobre los de escritura. El caso de escritura podría ser medido de mejor forma por otro tipo de experimento, por ejemplo, con seguimiento de pupila durante el desarrollo, o midiendo tiempo para corregir errores de sintaxis en un archivo.

Con respecto al modelo conceptual de casos de uso propuesto, solo se puede asumir como válida la diferenciación según tipo de actividad. Se cuenta con evidencia estadísticamente significativa que demuestra que el tipo de actividad influye sobre la usabilidad de las VCs. Mientras tanto, con respecto a la diferenciación según nivel de familiaridad con la base de código, por la forma en que se plantea el experimento, no se puede descartar del todo su efecto sobre la usabilidad. Es posible que esto afecte otros de los parámetros de usabilidad, como la eficacia y eficiencia, que no fueron analizados en este experimento por las restricciones presentadas. Además, al plantearse múltiples desafíos de lectura sobre el mismo archivo de código, puede que el programador reconozca la pieza como familiar antes de lo pronosticado.

Estos antecedentes permiten deducir consideraciones fundamentales para el diseño de el caso de estudio con programadores ciegos. Se debe priorizar que la mediciones de optimalidad contemplen todos los parámetros de usabilidad y accesibilidad. Adicionalmente, se debe limitar la cantidad de desafíos que utilizan cada archivo de código.

# Capítulo 4

## VisualCueTransform, extensión para Visual Studio Code

VisualCueTransform es una extensión de VSC creada para la presente tesis. Esta permite a un programador ciego determinar si una palabra en el código se considera reservada para el lenguaje Python, según su alcance (*scope*). Para lograr esto, se capturan eventos desencadenantes y se genera una NVC determinada para transmitir la información al programador.

La extensión se crea con el fin de evaluar la optimalidad de distintos tipos de NVCs. Se implementó para el ambiente de desarrollo integrado Visual Studio Code ya que es uno de los más populares a la fecha, se encuentra disponible para múltiples sistemas operativos, y ofrece una API que permite extender sus capacidades. Es relevante el hecho de que esté disponible para Windows, ya que el software de lectura de pantalla NVDA utiliza las APIs de accesibilidad de este sistema operativo. Por otro lado, según el índice PYPL, el lenguaje de programación Python es el más popular a nivel mundial [9], por lo que se consideró este lenguaje para la identificación de palabras reservadas. Tanto VSC como Python son familiares para los programadores ciegos entrevistados durante la etapa de exploración del problema.

### 4.1. Implementación

Se implementó la extensión VisualCueTransform utilizando la API de extensiones de VSC [11]. Esta permite acceder e interactuar con los distintos componentes de VSC, siendo de particular relevancia el espacio de trabajo (*workspace*) que el usuario ha definido, y la ventana (*window*) en la que esté editando un archivo particular. La extensión fue desarrollada en el lenguaje de programación TypeScript, utilizando el administrador de paquetes npm, ya que VSC provee un ambiente de ejecución Node.js para sus extensiones.

La extensión cuenta con cuatro modos de retroalimentación: sonido inerte, texto hablado, retroalimentación háptica y desactivada. Cuando la extensión está instalada en VSC, esta se activará automáticamente cuando el usuario se encuentre editando un archivo Python (extensión de archivo .py). Los modos “sonido inerte”, “texto hablado” y “retroalimentación háptica” generan la NVC respectiva cuando el usuario selecciona o escribe una palabra en un alcance en que esta se considere reservada para el lenguaje Python. Cuando la extensión se

configura en modo “desactivada”, no se generan NVCs adicionales, ante ninguna interacción del usuario.

Cuando la extensión está activada, esto es, en un modo distinto a “desactivada”, se detectan tres tipos de eventos desencadenantes: clic sobre una palabra, selección de texto y fin de escritura. Para esto, se implementaron dos controladores: controlador de selección, y controlador de escritura. La captura de los eventos de clic sobre una palabra y selección de texto es realizada por el controlador de selección. El evento de clic sobre una palabra se modela como una selección cuyo inicio y fin tienen el mismo índice. La captura del evento de fin de escritura es realizada por el controlador de escritura.

Una vez se detecta el evento desencadenante, se procede a verificar que lo seleccionado o lo escrito sea efectivamente una palabra reservada del lenguaje. Para esto, primero se verifica que el alcance sea el correcto. Luego, se verifica coincidencia con alguna de las cuatro listas de palabras reservadas. Estas cuatro listas se crearon a partir de los cuatro tipos de alcances que VSC asigna a las palabras reservadas de Python, lo que hace que su resaltado de sintaxis por defecto les asigne cuatro colores distintos. Los cuatro tipos son: constantes, operadores lógicos, controladores de memoria y controladores de flujo. La tabla 4.1 explicita las palabras reservadas del lenguaje Python y su clasificación según el inspector de alcances de VSC. Si ambas condiciones se cumplen, se emite una NVC a través de un servicio de notificación. El tipo de NVC con el que se notifica dependerá de la configuración de la extensión.

Tabla 4.1: Palabras reservadas en Python según alcance asociado por VSC.

| Palabras reservadas       |  |
|---------------------------|--|
| <b>Constantes</b>         | False, None, True  |
| <b>Lógicas</b>            | and, is, not, or   |
| <b>Control de memoria</b> | class, def, lambda, nonlocal   |
| <b>Control de flujo</b>   | as, assert, async, await, break, continue, del, elif, else, except, finally, for, from, global, if, import, in, pass, raise, return, try, while, with, yield |

Ya que VSC no provee acceso a los alcances asociados a los *tokens* a través de su API, la capacidad de la extensión para detectar palabras claves es limitada. La simplificación de la detección de alcance detecta si hay un símbolo numeral (#) en la misma línea, o si la cantidad de comillas, anterior al texto seleccionado o escrito, es impar. Esto es suficiente para la correcta detección de palabras reservadas, en el conjunto de archivos de código que se utilizaron en el caso de estudio.

#### 4.1.1. Modos sonido inerte y texto hablado

La extensión cuenta con dos tipos de retroalimentación sonora: sonido inerte y texto hablado. La retroalimentación a través de sonido inerte consiste en la reproducción de un archivo de audio, en formato MP3, con el sonido de una campana durante 1 segundo. La retroalimentación a través de texto hablado corresponde a la reproducción de un archivo de

audio, en formato MP3, con locuciones de los conceptos “*constant*”, “*logical*”, “*storage*”, y “*control*”. Estos se reproducen según el tipo de palabra reservada con la que se interactúe.

El archivo de audio inerte fue descargado de un proveedor gratuito de efectos de sonido<sup>9</sup>, y posteriormente fue editado para ajustar su duración. Los archivos de audio de texto hablado fueron generados utilizando una plataforma web gratuita<sup>10</sup>. La elección de tipo de voz fue arbitraria. Se eligió el idioma inglés para las locuciones porque es utilizado internacionalmente, y porque las palabras clave de Python también están en inglés. Para la reproducción de los archivos de audio, la extensión utiliza el paquete “sound-play” en su versión 1.1.0.

### 4.1.2. Modo retroalimentación háptica

Para la implementación de la retroalimentación háptica se utilizó una pulsera inteligente MiBand 3, de la marca Xiaomi. Esta pulsera, en su uso normal, utiliza vibraciones para informar al usuario sobre una nueva notificación. Gracias al trabajo de memoria de Elías Zelada [31], se cuenta con una implementación previa que permite conectar con este modelo particular de pulsera con un computador, a través de Bluetooth Low Energy (BLE en adelante). Esto permite recibir información de la pulsera o enviar notificaciones customizadas que generen vibraciones.

Para poder conectar un computador con la pulsera a través de Bluetooth, los programas existentes utilizan la pila de módulos BlueZ, que implementa los estándares del protocolo Bluetooth en Linux. Sin embargo, para el caso de estudio, se requiere ejecutar el lector de pantalla NVDA. Considerando que NVDA depende de la API de accesibilidad de Windows, se implementó un servidor HTTP en Linux, capaz de recibir solicitudes para conectar y hacer vibrar la pulsera. Con esto, cuando el usuario interactúa con VSC en un computador, teniendo la extensión activada en modo háptico, si se identifica una palabra clave del lenguaje Python, se enviará un mensaje HTTP a un segundo computador. Este segundo computador ejecuta un servidor HTTP, que en respuesta escribirá en la característica de notificaciones de la pulsera a través de BLE, haciendo que esta vibre.

Los procesos necesarios para la interacción completa, esto es, la utilización de NVDA para acceder a VSC, y la escritura de características a través de Bluetooth para que la pulsera vibre, no se pueden integrar en un solo computador utilizando recursos existentes. Máquinas virtuales como las de VirtualBox o VMWare no soportan el acceso a Bluetooth por parte de la instancia huésped (*guest*), y los proveedores que lo soportan no pueden garantizar la estabilidad de la conexión. Esto último es clave para mantener el emparejamiento de los dispositivos involucrados. Análogamente, Windows Subsystem for Linux (WSL) no permite que el ambiente GNU/Linux que se ejecute acceda al Bluetooth. Por esto, para el funcionamiento de este modo de retroalimentación, se requiere un computador secundario que actúe como puente entre los sistemas operativos. La figura 4.1 esquematiza los procesos en ejecución y protocolos de comunicación necesarios para hacer vibrar la pulsera. Con esta

<sup>9</sup> “Bell, school handheld bell pick up from table” de [www.zapsplat.com](http://www.zapsplat.com) (Consultado por última vez el 26 de diciembre de 2021)

<sup>10</sup> Se utilizó la configuración “US English - Kendra” de <https://ttsmp3.com/> (Consultado por última vez el 26 de diciembre de 2021)

configuración, la retroalimentación háptica muestra una latencia promedio de 1,8 segundos, desde que el usuario interactúa con el edito de código, hasta que la pulsera vibra.

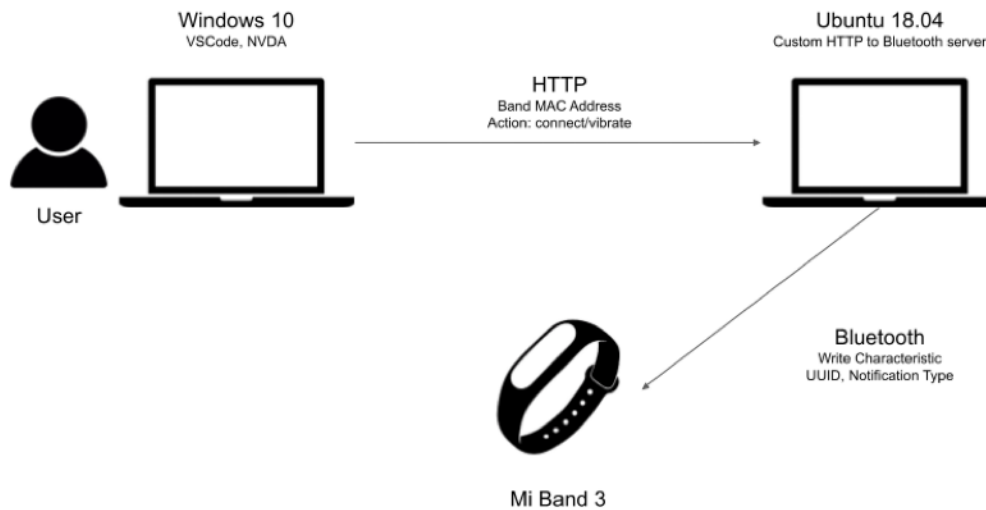


Figura 4.1: Modelo de implementación retroalimentación háptica.

## 4.2. Estructura del código

En esta sección se presentan las clases y funciones de los desarrollos principales, y el cómo interactúan entre ellas. Esta sección se divide en dos subsecciones. La primera trata sobre el archivo `extension.ts`, que implementa la extensión de VSC. La segunda trata sobre el archivo `main.py`, que implementa el servidor HTTP y BLE que establece la comunicación entre la extensión y la pulsera Bluetooth.

### 4.2.1. Extensión VisualCueTransform

Las extensiones de VSC son proyectos en TypeScript o JavaScript que luego son compilados y ejecutados en un ambiente Node.js [11]. La extensión VisualCueTransform en particular es un proyecto en TypeScript. Su archivo principal, `extension.ts`, define los comandos con los que se interactúa con la extensión, un sistema de captura y evaluación de eventos desencadenantes, y un sistema de retroalimentación usuaria. A continuación se presentan y describen las principales clases y funciones de este archivo.

#### 4.2.1.1. Clases

Para la captura y validación de eventos en el editor de código de VSC se implementaron las siguientes clases:

- **SelectionController**: permite capturar eventos que modifican la selección del editor de texto, suscribiéndose a su contexto. Implementa la función `dispose` que maneja la

descripción a contextos cuando la extensión es desactivada.

```
1 class SelectionController {
2
3     private __selection: Selection;
4     private __disposable: vscode.Disposable;
5
6     constructor(selection: Selection) {
7         this.__selection = selection;
8
9         // subscribe to selection change and editor activation events
10        let subscriptions: vscode.Disposable[] = [];
11        vscode.window.onDidChangeTextEditorSelection(this.__onEvent, this,
12        ↪ subscriptions);
13
14        // create a combined disposable from both event subscriptions
15        this.__disposable = vscode.Disposable.from(...subscriptions);
16    }
17
18    private __onEvent(event: vscode.TextEditorSelectionChangeEvent) {
19        this.__selection.updateSelection(event);
20    }
21
22    public dispose() {
23        this.__disposable.dispose();
24    }
25 }
```

- **Selection:** modela la selección de texto en el editor de código. Cuando la selección de texto cambia, se transforma la selección en el texto que contiene. Luego, la función `__getSelectionCues` verifica que la nueva selección tenga asociado un alcance apropiado para ser interpretada como palabra clave utilizando la función privada `__checkScope`. De ser así, se verifica que la selección sea una palabra clave utilizando un objeto de la clase `WordChecker`.

```
1 export class Selection {
2
3     private __string: string;
4
5     private __wordchecker : WordChecker;
6
7     constructor(notifier:Notify){
8         this.__string= "";
9         this.__wordchecker= new WordChecker(notifier);
10    }
11
12    public updateSelection(event: vscode.TextEditorSelectionChangeEvent) {
13        // ignore clicks
14        const currentSelection = event.selections && event.selections.length && event.
15        ↪ selections[0];
16        const currentSelectionIsJustAClick = currentSelection.anchor.line ===
```

```

↪ currentSelection.active.line && currentSelection.anchor.character ===
↪ currentSelection.active.character;
16 if (currentSelectionIsJustAClick) {
17     return false;
18 }
19
20 // Get the current text editor
21 let editor = vscode.window.activeTextEditor;
22 if (!editor) {
23     return;
24 }
25
26 let doc = editor.document;
27
28 // Only update cues if an python file
29 if (doc.languageId === "python") {
30     this._getSelectionCues(event);
31
32 } else {
33     return;
34 }
35 }
36 }
37
38 public _getSelectionCues(event: vscode.TextEditorSelectionChangeEvent){
39     // generate text from selections
40     const eol = event.textEditor.document.eol === vscode.EndOfLine.LF ? '\n' : '\r\n';
41     let text = event.selections.map(selection => event.textEditor.document.getText(
↪ selection)).join(eol);
42     text = text.replace(/^\\s+/, "");
43     text = text.replace(/\\s+$/, "");
44     let selection_range = new vscode.Range(event.selections[0].anchor, event.selections
↪ [0].end)
45     if (this._checkScope(selection_range, event.textEditor.document)){
46         this._wordchecker.check(text);
47     }
48 }
49
50 private _checkScope(range: vscode.Range, document:vscode.TextDocument){
51     let baseline = new vscode.Position(0,0)
52     let pivot = range.start;
53     let text_before_pivot_range = new vscode.Range(baseline,pivot);
54     let text_before_pivot = document.getText(text_before_pivot_range);
55     let quotation_count = (text_before_pivot.split('\\').length-1)
56     let pivot_line_position_start = new vscode.Position(pivot.line,0)
57     let line_before_pivot_range = new vscode.Range(pivot_line_position_start,pivot
↪ )
58     let text_line_before_pivot = document.getText(line_before_pivot_range);
59     let hashtag_count = (text_line_before_pivot.split('#').length-1)
60     console.log(hashtag_count+" "+quotation_count)
61     if (hashtag_count < 1 && !(quotation_count&1)){
62         return true

```



```

63     }
64   }
65 }

```

- **WritingController:** permite capturar eventos que modifican el archivo sobre el que se está trabajando, suscribiéndose a su contexto. Implementa la función `dispose` que maneja la desuscripción a contextos cuando la extensión es desactivada.

```

1  class WritingController {
2
3     private __writingBuffer: WritingBuffer;
4     private __disposable: vscode.Disposable;
5
6     constructor(writingBuffer: WritingBuffer) {
7         this.__writingBuffer = writingBuffer;
8
9         let subscriptions: vscode.Disposable[] = [];
10        vscode.workspace.onDidChangeTextDocument(this.__onEvent, this, subscriptions);
11
12        this.__disposable = vscode.Disposable.from(...subscriptions);
13    }
14
15    private __onEvent(event: vscode.TextDocumentChangeEvent) {
16        console.log("writingController recibió el evento")
17        this.__writingBuffer.updateBuffer(event);
18    }
19
20    public dispose() {
21        this.__disposable.dispose();
22    }
23 }

```

- **WritingBuffer:** modela la escritura de texto en el archivo que se está editando. Cuando este archivo cambia, se actualiza el *buffer* de escritura. Esto se hace utilizando la función `__getLastTypedWord`, que transforma la posición del cursor en el texto de la palabra que lo contiene. Luego, se verifica que la nueva selección tenga asociado un alcance apropiado para ser interpretada como palabra clave utilizando la función privada `__checkScope`. De ser así, se verifica la palabra en *buffer* sea una palabra clave utilizando un objeto de la clase `WordChecker`.

```

1  export class WritingBuffer {
2
3     private __string: string;
4
5     private __wordchecker : WordChecker;
6
7     constructor(notifier:Notify){
8         this.__string= "";
9         this.__wordchecker= new WordChecker(notifier);
10    }

```



```

11
12 public updateBuffer(event: vscode.TextDocumentChangeEvent) {
13     let editor = vscode.window.activeTextEditor;
14     if (!editor) {
15         return;
16     }
17
18     let doc = event.document;
19
20     if (doc.languageId === "python") {
21         this._getLastTypedWord(event);
22     } else {
23         return;
24     }
25 }
26
27 private _getLastTypedWord(event: vscode.TextDocumentChangeEvent){
28     let document = event.document
29     //const eol = document.eol === vscode.EndOfLine.LF ? '\n' : '\r\n';
30     let position = event.contentChanges[0].range.start
31     let range = document.getWordRangeAtPosition(position)
32     let word = document.getText(range)
33     if (this._checkScope(range,document)){
34         this._wordchecker.check(word);
35     }
36 }
37
38 private _checkScope(range: vscode.Range, document:vscode.TextDocument){
39     let baseline = new vscode.Position(0,0)
40     let pivot = range.start;
41     let text_before_pivot_range = new vscode.Range(baseline,pivot);
42     let text_before_pivot = document.getText(text_before_pivot_range);
43     let quotation_count = (text_before_pivot.split('"').length-1)
44     let pivot_line_position_start = new vscode.Position(pivot.line,0)
45     let line_before_pivot_range = new vscode.Range(pivot_line_position_start,pivot
↵ )
46     let text_line_before_pivot = document.getText(line_before_pivot_range);
47     let hashtag_count = (text_line_before_pivot.split('#').length-1)
48     console.log(hashtag_count+" "+quotation_count)
49     if (hashtag_count < 1 && !(quotation_count&1)){
50         return true
51     }
52 }
53 }

```

Para el reconocimiento de palabras reservadas del lenguaje Python, se implementaron las siguientes clases:

- **Keyword:** clase abstracta. Se construye con un objeto de clase **Notify**. Implementa la función **check**, que verifica si una palabra está en una lista de palabras reservadas y, de

ser así, solicita que el objeto de clase `Notify` notifique al usuario. Delega a las extensiones de esta clase la definición de la lista de palabras que reconocen como reservadas, así como las rutas de los archivos de audio que deben ser reproducidos en cada modo de la extensión.

```
1 export abstract class Keyword{
2   abstract __pythonkw:string[];
3   abstract __inertSoundFilePath:string;
4   abstract __speechSoundFilePath:string;
5   private __notifier:Notify;
6
7   constructor(notifier:Notify){
8     this.__notifier=notifier
9   }
10
11  private __getPythonKW(){
12    return this.__pythonkw
13  }
14
15  private __getInertSoundFilePath(){
16    return this.__inertSoundFilePath
17  }
18  }
19  private __getSpeechSoundFilePath(){
20    return this.__speechSoundFilePath
21  }
22
23  private __getNotifier(){
24    return this.__notifier
25  }
26
27  private __setPythonKW(new__pythonkw:string[]){
28    this.__pythonkw = new__pythonkw
29  }
30
31  private __setInertSoundFilePath(new__inertSoundFilePath:string){
32    this.__inertSoundFilePath = new__inertSoundFilePath
33  }
34
35  private __setSpeechSoundFilePath(new__speechSoundFilePath:string){
36    this.__speechSoundFilePath = new__speechSoundFilePath
37  }
38
39  private __setNotifier(new__notifier:Notify){
40    this.__notifier = new__notifier
41  }
42
43  public check(text:string) : boolean{
44    console.log("keyword checking")
45    if (this.__getPythonKW().some(x => x === text)){
46      console.log("found keyword")
47      this.__getNotifier().notify(this.__getInertSoundFilePath(),this.
```

```

↪ __getSpeechSoundFilePath());
48     return true;
49 }
50     return false;
51 }
52 }

```

- **Constant:** implementa la clase abstracta `Keyword`. Define una lista con las palabras reservadas del lenguaje Python a las cuales el inspector de alcances de VSC les asigna un alcance de constante. Define los archivos de audio a reproducir según el modo de configuración de la extensión.

```

1 export class Constant extends Keyword{
2     __pythonkw = ["False", "None", "True"];
3     __inertSoundFilePath = path.join(
4         __dirname,
5         "..",
6         "sounds",
7         "inert_bell.mp3"
8     );
9     __speechSoundFilePath = path.join(
10        __dirname,
11        "..",
12        "sounds",
13        "ttsMP3.com_constant.mp3"
14    );
15 }

```

- **Logical:** implementa la clase abstracta `Keyword`. Define una lista con las palabras reservadas del lenguaje Python a las cuales el inspector de alcances de VSC les asigna un alcance de operador lógico. Define los archivos de audio a reproducir según el modo de configuración de la extensión.

```

1 export class Logical extends Keyword{
2     __pythonkw = ["and", "is", "not", "or"];
3     __inertSoundFilePath = path.join(
4         __dirname,
5         "..",
6         "sounds",
7         "inert_bell.mp3"
8     );
9     __speechSoundFilePath = path.join(
10        __dirname,
11        "..",
12        "sounds",
13        "ttsMP3.com_logical.mp3"
14    );
15 }

```

- **Storage:** implementa la clase abstracta `Keyword`. Define una lista con las palabras

reservadas del lenguaje Python a las cuales el inspector de alcances de VSC les asigna un alcance de control de memoria. Define los archivos de audio a reproducir según el modo de configuración de la extensión.

```
1 export class Storage extends Keyword{
2   __pythonkw = ["class", "def", "lambda", "nonlocal"];
3   __inertSoundFilePath = path.join(
4     __dirname,
5     "..",
6     "sounds",
7     "inert_bell.mp3"
8   );
9   __speechSoundFilePath = path.join(
10    __dirname,
11    "..",
12    "sounds",
13    "ttsMP3.com_storage.mp3"
14  );
15 }
```

- **Controll**: implementa la clase abstracta **Keyword**. Define una lista con las palabras reservadas del lenguaje Python a las cuales el inspector de alcances de VSC les asigna un alcance de control de flujo. Define los archivos de audio a reproducir según el modo de configuración de la extensión.

```
1 export class Controll extends Keyword{
2   __pythonkw = ["as", "assert", "async", "await",
3     "break", "continue", "del", "elif", "else",
4     "except", "finally", "for", "from", "global",
5     "if", "import", "in", "pass", "raise",
6     "return", "try", "while", "with", "yield"];
7   __inertSoundFilePath = path.join(
8     __dirname,
9     "..",
10    "sounds",
11    "inert_bell.mp3"
12  );
13  __speechSoundFilePath = path.join(
14    __dirname,
15    "..",
16    "sounds",
17    "ttsMP3.com_logical.mp3"
18  );
19 }
```

- **WordChecker**: modela el proceso de verificación de pertenencia de palabras a distintos conjuntos de palabras reservadas. Para esto, utiliza la función pública **check** que disponen los objetos de las clases que implementan **Keyword**.

```
1 export class WordChecker{
2   private __constant : Constant;
```

```

3   private __logical : Logical;
4   private __storage : Storage;
5   private __controll : Controll;
6
7   constructor(notifier:Notify){
8       this.__constant = new Constant(notifier);
9       this.__logical = new Logical(notifier);
10      this.__storage = new Storage(notifier);
11      this.__controll = new Controll(notifier);
12  }
13
14  public check(text: string){
15      console.log('checking...')
16      var ready = this.__constant.check(text);
17      if(!ready){
18          ready = this.__logical.check(text);
19          if(!ready){
20              ready = this.__storage.check(text);
21              if(!ready){
22                  ready = this.__controll.check(text);
23              }
24          }
25      }
26  }
27  }

```

Para la generación de NVCs y comunicación dispositivos periféricos, se implementaron las siguientes clases:

- **Notify**: modela el proceso de generación de NVCs. Utiliza un archivo de configuración asociado a la extensión para identificar el modo inicial, y las direcciones del servidor HTTP y la pulsera Bluetooth. La función **update** permite cambiar el modo de configuración. La función **\_\_playAudio** permite reproducir un archivo MP3. La función pública **notify** permite generar una NVC según el modo de configuración. Para esto, se debe proveer las direcciones de los archivos MP3 a reproducir en caso que la extensión esté configurada en alguno de los modos de audio. Utiliza un objeto de la clase **BluetoothController** para manejar el modo de retroalimentación háptica.

```

1  export class Notify{
2      private __mode: string|undefined;
3      private __btController: BluetoothController;
4
5      constructor(){
6          const configuration = vscode.workspace.getConfiguration();
7          this.__mode = configuration.get('visualcuetransform.general.feedbackType');
8          this.__btController = new BluetoothController(String(configuration.get('
↳ visualcuetransform.HTTPServer.address')),Number(configuration.get('
↳ visualcuetransform.HTTPServer.port')),String(configuration.get('
↳ visualcuetransform.bluetooth.address'))));
9      }

```

```

10
11 private __getMode(){
12     return this.__mode
13 }
14
15 private __setMode(new__mode:string){
16     this.__mode = new__mode
17 }
18
19 private __getBTController(){
20     return this.__btController
21 }
22
23 private __setBTController(new__btController:BluetoothController){
24     this.__btController = new__btController
25 }
26
27 private __playAudio = (soundFilePath:string) => {
28     sound.play(soundFilePath)
29 };
30
31 public notify(inertSoundFilePath:string,speechSoundFilePath:string){
32     console.log(this.__getMode())
33     if (this.__getMode() == 'inert'){
34         this.__playAudio(inertSoundFilePath)
35     }
36     else if (this.__getMode() == 'speech'){
37         this.__playAudio(speechSoundFilePath)
38     }
39     else if (this.__getMode() == 'haptic'){
40         this.__getBTController().sendVibrate()
41     }
42 }
43
44 public update(){
45     const configuration = vscode.workspace.getConfiguration();
46     this.__mode = configuration.get('visualcuetransform.general.feedbackType');
47     console.log("configuration changed to "+this.__mode)
48 }
49 }

```

- BluetoothController:** modela el proceso de comunicación con el servidor HTTP que maneja la conexión con la pulsera Bluetooth. Al construirse un objeto, se obtienen las coordenadas del servidor HTTP y la pulsera desde el archivo de configuración de la extensión, y se solicita al servidor HTTP establecer la conexión con la pulsera, esto es, autenticación y emparejamiento. Se implementa la función privada `http_post_request` que, utilizando el paquete `follow-redirects`, envía las solicitudes de conexión y vibración a través de HTTP al servidor. La función pública `vibrate` utiliza la función privada `http_post_request`, instruyendo al servidor HTTP a escribir en la característica de mensaje personalizado que hace vibrar la pulsera.



```

1 class BluetoothController {
2
3     private __HTTPServerAddress: string;
4     private __BluetoothAddress: string|undefined;
5     private __HTTPServerPort: number|undefined;
6
7     constructor(httpServerAddress:string, httpServerPort:number|undefined,
8         ↪ bluetoothAddress:string) {
9         this.__HTTPServerAddress = httpServerAddress;
10        this.__HTTPServerPort = httpServerPort;
11        this.__BluetoothAddress = bluetoothAddress;
12        this.__init()
13    }
14
15    private __init(){
16        this.http_post_request("connect")
17    }
18
19    private __getHTTPServerAddress(){
20        return this.__HTTPServerAddress
21    }
22
23    private __getHTTPServerPort(){
24        console.log("posrt is"+this.__HTTPServerPort)
25        return this.__HTTPServerPort
26    }
27
28    private __getBluetoothAddress(){
29        return this.__BluetoothAddress
30    }
31
32    public sendVibrate(){
33        this.http_post_request("vibrate")
34    }
35
36    public update(){
37        const configuration = vscode.workspace.getConfiguration();
38        this.__HTTPServerAddress = String(configuration.get('visualcuetransform.
39        ↪ HTTPServer.address'));
40        this.__BluetoothAddress = String(configuration.get('visualcuetransform.bluetooth.
41        ↪ address'));
42        this.__init();
43    }
44
45    private http_post_request(todo:string){
46
47        var options = {
48            'method': 'GET',
49            'hostname': this.__getHTTPServerAddress(),
50            'port': 8001,
51            'path': '/',

```

```

50     'headers': {
51         'Bluetooth-Address': this.__getBluetoothAddress(),
52         'To-Do': todo
53     },
54     'maxRedirects': 20
55 };
56
57 var req = http.request(options, function (res) {
58     var chunks = [];
59
60     res.on("data", function (chunk) {
61         chunks.push(chunk);
62     });
63
64     res.on("end", function (chunk) {
65         var body = Buffer.concat(chunks);
66         console.log(body.toString());
67     });
68
69     res.on("error", function (error) {
70         console.error(error);
71     });
72 });
73
74 req.end();
75 }
76 }

```

#### 4.2.1.2. Funciones

La función principal del programa corresponde a **activate**. Esta función es llamada cuando se activa la extensión. Cabe considerar que la extensión cuenta con una configuración que la activa automáticamente cuando se abre un archivo de código Python en el editor de código. Esta función se encarga de crear los objetos necesarios para el uso de NVCs, suscribir los objetos que así lo requieran al contexto de ejecución, e implementar los comandos para cambiar el modo de retroalimentación de la extensión. Esto último lo hace actualizando el objeto **notifier** de la clase **Notify**.

```

1  export function activate(context: vscode.ExtensionContext) {
2
3      console.log('"Visual Cue Transform" is now active');
4      let notifier = new Notify()
5      let selection = new Selection(notifier);
6      let selectionController = new SelectionController(selection);
7      let writingBuffer = new WritingBuffer(notifier);
8      let writingController = new WritingController(writingBuffer);
9
10     context.subscriptions.push(selectionController);
11     context.subscriptions.push(writingController);

```



```

12
13 let disableFeedback = vscode.commands.registerCommand('visualcuetransform.set.
    ↪ feedbacktype.disabled', async () => {
14     const configuration = vscode.workspace.getConfiguration();
15     const feedbackType = configuration.get('visualcuetransform.general.feedbackType')
16     if (feedbackType !== 'disabled') {
17         await configuration.update('visualcuetransform.general.feedbackType', 'disabled')
18         vscode.window.showInformationMessage('The visual cue transformation feedback
    ↪ has been disabled');
19         notifier.update()
20     }
21 });
22
23 let inertFeedback = vscode.commands.registerCommand('visualcuetransform.set.
    ↪ feedbacktype.inert', async () => {
24     const configuration = vscode.workspace.getConfiguration();
25     const feedbackType = configuration.get('visualcuetransform.general.feedbackType')
26     if (feedbackType !== 'inert') {
27         await configuration.update('visualcuetransform.general.feedbackType', 'inert')
28         vscode.window.showInformationMessage('The visual cue transformation feedback
    ↪ type has been set to inert sound feedback');
29         notifier.update()
30     }
31 });
32
33 let speechFeedback = vscode.commands.registerCommand('visualcuetransform.set.
    ↪ feedbacktype.speech', async () => {
34     const configuration = vscode.workspace.getConfiguration();
35     const feedbackType = configuration.get('visualcuetransform.general.feedbackType')
36     if (feedbackType !== 'speech') {
37         await configuration.update('visualcuetransform.general.feedbackType', 'speech')
38         vscode.window.showInformationMessage('The visual cue transformation feedback
    ↪ type has been set to speech feedback');
39         notifier.update()
40     }
41 });
42
43 let hapticFeedback = vscode.commands.registerCommand('visualcuetransform.set.
    ↪ feedbacktype.haptic', async () => {
44     const configuration = vscode.workspace.getConfiguration();
45     const feedbackType = configuration.get('visualcuetransform.general.feedbackType')
46     if (feedbackType !== 'haptic') {
47         await configuration.update('visualcuetransform.general.feedbackType', 'haptic')
48         vscode.window.showInformationMessage('The visual cue transformation feedback
    ↪ type has been set to haptic feedback');
49         notifier.update()
50     }
51 });
52
53 context.subscriptions.push(disableFeedback);
54 context.subscriptions.push(inertFeedback);
55 context.subscriptions.push(speechFeedback);

```

```
56 context.subscriptions.push(hapticFeedback);
57 }
```

## 4.2.2. Servidor HTTP y BLE

La comunicación con la pulsera MiBand 3 se puede establecer fácilmente utilizando la biblioteca MiBand3 de Yogesh Ojha publicada en GitHub<sup>11</sup>. Para utilizarla se siguió las instrucciones provistas en el archivo `README.md` del repositorio.

Se tradujo la biblioteca a Python 3, ya que algunos de los paquetes necesarios para utilizarla con Python 2 se encuentran obsoletos. De esta librería, se modificó el archivo `main.py` para incluir un servidor HTTP. Este servidor escucha solicitudes HTTP y, según la acción solicitada, establece conexión con la pulsera, o envía un mensaje personalizado para hacerla vibrar. Cabe mencionar que se eliminó la interfaz de usuario en terminal que provee la biblioteca ya que, en esta implementación, la interacción con el usuario se realiza a través de HTTP.

```
1 import sys
2 from auth import MiBand3
3 import time
4 import os
5 import http.server
6 import socketserver
7
8 MAC_ADDR = ""
9 band = MiBand3(MAC_ADDR, debug=True)
10
11 class HttpToBle(http.server.SimpleHTTPRequestHandler):
12     global MAC_ADDR, band
13     def do_GET(self):
14         todo = self.headers.get("To-Do")
15         if todo == "vibrate":
16             band.send_custom_alert(5)
17             self.send_response(200)
18         elif todo == "connect":
19             MAC_ADDR = self.headers.get("Bluetooth-Address")
20             band = MiBand3(MAC_ADDR, debug=True)
21             band.setSecurityLevel(level = "medium")
22             if band.initialize():
23                 print("Initialized...")
24                 band.disconnect()
25                 sys.exit(0)
26         else:
27             band.authenticate()
28             self.send_response(200)
29
30
```

<sup>11</sup> <https://github.com/yogeshojha/MiBand3>

```
31 handler_object = HttpToBle
32
33 PORT = 8001
34
35 my_server = socketserver.TCPServer("", PORT), handler_object)
36
37 my_server.serve_forever()
```

### 4.3. Retrospectiva

El diseño e implementación de la extensión VisualCueTransform fueron procesos altamente complejos. VSC no expone el inspector de scope en su API, por lo que no hay forma de obtener los alcances que el IDE asigna a los *tokens*. Tampoco cuenta con formas eficaces y eficientes de identificar, desde el editor de código, la posición del último carácter ingresado, para determinar si se terminó de escribir una palabra reservada del lenguaje. Es difícil determinar qué paquetes de NPS son compatibles con el ambiente de ejecución que provee, ya que algunas funcionalidades pueden estar restringidas para la extensión o para el IDE. Por esto, es recomendable utilizar como referencia otras extensiones que se encuentren publicadas.

VisualCueTransform no fue publicada en el mercado de extensiones de VSC por dos razones. En primer lugar, porque el modo de retroalimentación háptica requiere de una configuración extremadamente particular y engorrosa para funcionar correctamente. En segundo lugar, porque el reconocimiento de alcance solo garantiza funcionar correctamente en los archivos utilizados en el caso de estudio. Esto se podría solucionar en un futuro si VSC disponibiliza el inspector de alcances en su API, o implementando un servidor de lenguajes para la versión apropiada de Python. A pesar de lo anterior, el código de la extensión será disponibilizado en un repositorio público, posterior a la defensa de la presente tesis.

# Capítulo 5

## Estudio con Usuarios

El capítulo 3 propone y valida experimentalmente un modelo conceptual de casos de uso de señales de código fuente. El capítulo 4 describe la implementación de una extensión de VSC capaz de generar cuatro tipos de NVCs para un conjunto acotado de eventos desencadenantes. El presente capítulo describe el estudio realizado para explorar la optimalidad de distintas NVCs, según el modelo del capítulo 3, utilizando la extensión descrita en el capítulo 4.

Para el estudio realizado se siguió la metodología de caso de estudio único holístico exploratorio, según el marco teórico propuesto por Robert K. Yin [29]. La elección de esta metodología se debe principalmente al reducido número de sujetos que cumplen con los criterios de inclusión y exclusión definidos: personas adultas, ciegas, y con conocimientos básicos de programación. Estos criterios se deben a que las personas debían ser capaces de dar consentimiento para participar del estudio, y entender conceptos propios de la programación, como “palabras reservadas” (*keywords*) o “alcance” (*scope*).

Las siguientes secciones del presente capítulo describen las componentes metodológicas de la investigación, el escenario experimental utilizado, el análisis realizado y los resultados obtenidos. Finalmente, se presentan conclusiones sobre el estudio y se relacionan con las hipótesis y preguntas de investigación de la presente tesis.

### 5.1. Preguntas de investigación

Para el presente caso de estudio único, se definen las siguientes preguntas de investigación, cuyas respuestas, en su conjunto, permiten realizar un análisis sobre la optimalidad de las NVCs.

Con respecto a la eficacia, eficiencia, satisfacción y esfuerzo percibido en la resolución de problemas de desarrollo por programadores ciegos:

1. ¿Cuál es la señal no-visual de código de mayor eficacia, eficiencia y satisfacción, y menor esfuerzo percibido en la resolución de problemas de lectura de código?
2. ¿Cuál es la señal no-visual de código de mayor eficacia, eficiencia y satisfacción, y menor esfuerzo percibido en la resolución de problemas de escritura de código?

3. ¿Cuál es la diferencia en la eficacia, eficiencia, satisfacción y esfuerzo percibido de señales no-visuales de código, entre los casos de uso, según el conocimiento de la base de código?

## 5.2. Unidad de análisis

La unidad de análisis del presente caso de estudio se desprende directamente de las preguntas de investigación. Corresponde al proceso de resolución de desafíos de programación de escritura y lectura, utilizando NVCs, por parte de personas naturales adultas, ciegas y que declaren tener experiencia programando en el lenguaje de programación Python.

Los desafíos presentados se resuelven utilizando el IDE VSC versión 1.57 y el lenguaje de programación Python versión 3.8.9. Las NVCs utilizadas se implementan en VSC a través de VisualCueTransform, la extensión descrita en el capítulo 4.

## 5.3. Diseño experimental

El escenario experimental consiste en que el participante del caso de estudio se enfrenta a tres tipos de problemas de programación: un primero en que debe leer una base de código en búsqueda de palabras específicas; un segundo en que debe leer, en la misma base de código del problema anterior sobre la cual se han hecho modificaciones, en búsqueda de palabras específicas; y un tercero en que debe escribir sobre una base de código, completando la implementación de un programa o corrigiendo errores de sintaxis. Para cada uno de estos tipos de problemas, existen cuatro bases de códigos distintas pero análogas entre ellas, en que se debe resolver los problemas con cuatro configuraciones de NVCs: NVCs inactivas, earcons, texto hablado (*speech*) y retroalimentación háptica.

Se diseñó una prueba dirigida con usuarios, específicamente para este estudio, que permitiera contrastar la eficacia, eficiencia, satisfacción usuaria y nivel de esfuerzo percibido en cada una de las configuraciones del escenario experimental. Para lograr esto, se creó un conjunto de desafíos de programación a medida, a ser resueltos en VSC.

El cumplimiento de las consideraciones éticas fue de principal importancia para la realización de este estudio. El diseño de las actividades y el protocolo experimental tuvo en primera consideración el respeto irrestricto por los participantes, en especial dado que los participantes de este estudio son personas ciegas. El diseño experimental planteado en esta sección cuenta con la aprobación del comité de ética de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. En el anexo B.2.1 se adjunta el certificado emitido por este comité.

Los participantes de la prueba consintieron con respecto a la recolección y uso de los datos para la presente tesis. Para este fin, se les envió de antemano, por correo electrónico, una versión digital del formulario de consentimiento, tal que fuera accesible utilizando lectores de pantalla. Además, se leyó el formulario antes de la aplicación del experimento, obteniendo el consentimiento de manera verbal. Los datos personales de los participantes, recogidos previo

a la aplicación, fueron solo los estrictamente pertinentes para verificar que se satisficiera los criterios de inclusión y exclusión. Estos datos no fueron ni serán publicados, distribuidos o almacenados.

La prueba consta de quince partes. Las partes 1, 4, 7 y 10 son análogas entre ellas, ya que corresponden a tareas de lectura de una base de código desconocida. A su vez, las partes 2, 5, 8 y 11 son análogas entre ellas, ya que corresponden a tareas de lectura de una base de código que se asume conocida. Las partes 3, 6, 9 y 12 también son análogas entre ellas, ya que corresponden a casos de escritura. Las partes 0, 13 y 14 recopilan datos sobre el participante, la experiencia general con respecto al experimento, y la satisfacción con respecto a las configuraciones de NVCs utilizadas. Las partes que son análogas entre ellas se diferencian según la configuración de NVCs que se debe utilizar para resolverlas. La tabla 5.1 presenta la configuración de NVCs a utilizar en cada parte del experimento.

Tabla 5.1: Esquema de configuración de señales visuales según partes del diseño experimental.

| <b>Partes</b>     | <b>Configuración NVCs</b> |
|-------------------|---------------------------|
| <b>1, 2, 3</b>    | Apagada                   |
| <b>4, 5, 6</b>    | Earcons                   |
| <b>7, 8, 9</b>    | Speech                    |
| <b>10, 11, 12</b> | Háptico                   |
| <b>0, 13, 14</b>  | No Aplica                 |

A continuación, se describen las actividades a realizar en cada parte del diseño experimental.

- Parte 0: Se recopila información sobre el participante con respecto a: condiciones visuales, nivel de experiencia programando, familiaridad con el lenguaje de programación Python, familiaridad con la aplicación Visual Studio Code, y familiaridad con otros ambientes de desarrollo integrado. Se da tiempo al participante para explorar el IDE y las configuraciones de NVCs disponibles sobre un archivo de ejemplo.
- Parte 1: Sobre la lectura de una base de código desconocida. En esta sección se presenta un programa desconocido sobre el cual se debe encontrar palabras reservadas del lenguaje Python. Esta sección contiene un requerimiento, y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 2: Sobre la lectura de una base de código conocida. En esta sección se presenta el mismo programa de la Parte 1, pero con algunas modificaciones. Nuevamente se deben encontrar palabras reservadas del lenguaje Python. Esta sección contiene un requerimiento, y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 3: Sobre la escritura de código en una base desconocida. En esta sección se presenta un programa que debe ser modificado, reemplazando un comentario por una palabra reservada. Esta sección mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.

- Parte 4: Sobre la lectura de una base de código desconocida. En esta sección se presenta un programa desconocido sobre el cual se debe encontrar cadenas de caracteres que podrían ser palabras reservadas del lenguaje Python, pero que se encuentran fuera del alcance apropiado. Esta sección contiene un requerimiento y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 5: Sobre la lectura de una base de código conocida. En esta sección se presenta el mismo programa de la Parte 4, pero con algunas modificaciones. Nuevamente se debe encontrar cadenas de caracteres que podrían ser palabras reservadas del lenguaje Python, pero que se encuentran fuera del alcance apropiado. Esta sección contiene un requerimiento y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 6: Sobre la escritura de código en una base desconocida. En esta sección se presenta un programa que debe ser modificado, corrigiendo un error de sintaxis en una palabra reservada. Esta sección mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 7: Sobre la lectura de una base de código desconocida. En esta sección se presenta un programa desconocido sobre el cual se debe encontrar palabras reservadas del lenguaje Python. Esta sección contiene un requerimiento y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 8: Sobre la lectura de una base de código conocida. En esta sección se presenta el mismo programa de la Parte 7, pero con algunas modificaciones. Nuevamente se deben encontrar palabras reservadas del lenguaje Python. Esta sección contiene un requerimiento y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 9: Sobre la escritura de código en una base desconocida. En esta sección se presenta un programa que debe ser modificado, corrigiendo un error de sintaxis en una palabra reservada. Esta sección mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 10: Sobre la lectura de una base de código desconocida. En esta sección se presenta un programa desconocido sobre el cual se debe encontrar palabras reservadas del lenguaje Python. Esta sección contiene un requerimiento y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 11: Sobre la lectura de una base de código conocida. En esta sección se presenta el mismo programa de la Parte 1, pero con algunas modificaciones. Nuevamente se deben encontrar palabras reservadas del lenguaje Python. Esta sección contiene un requerimiento, y mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.
- Parte 12: Sobre la escritura de código en una base desconocida. En esta sección se presenta un programa que debe ser modificado, reemplazando un comentario por una palabra reservada. Esta sección mide la correctitud de la respuesta, el tiempo utilizado y el nivel de esfuerzo percibido necesario para resolverlo.



Parte 13: Sobre las configuraciones de NVCs. En esta sección se mide la satisfacción en general y particular con respecto a las distintas configuraciones de NVCs utilizadas durante el experimento.

Parte 14: Sobre la experiencia del participante. En esta sección se realiza una entrevista semiestructurada al participante. Esta trata sobre la experiencia en general, las NVCs utilizadas y cómo se podrían incorporar en los procesos de desarrollo del participante.

### 5.3.1. Validez de constructo

Para garantizar la validez de constructo, el protocolo experimental fue revisado y validado por un experto en diseño de estudios empíricos involucrando sujetos humanos. En base al protocolo original, se realizó una aplicación piloto. En el piloto participó un programador, con cuya retroalimentación se calibró el protocolo, utilizando menos tecnicismos, y clarificando los objetivos de cada desafío. Además, se agregó una ventana de 10 minutos, al inicio de la aplicación, para que el participante se familiarice con todas las configuraciones de la extensión y el *hardware* utilizado. La versión del protocolo utilizada en la aplicación formal corresponde a la versión validada por el experto, con correcciones realizadas en base a la aplicación piloto.

Se determina el nivel de conocimiento de programación del participante a través de un cuestionario de autorreporte. Las fuentes de datos son las usuales para las mediciones de eficacia, eficiencia, satisfacción. Se mide la eficacia según la correctitud de las respuestas del usuario, conociendo de antemano las respuestas correctas. Se mide la eficiencia según el tiempo utilizado, y el esfuerzo percibido por el participante reportado en una escala de Likert. Se mide la satisfacción según un cuestionario de autorreporte y la Escala de Usabilidad de Sistemas [8]. Se aplica este último instrumento ya que es un estándar en ingeniería de sistemas, que permite establecer comparaciones a partir de un solo valor que representa la usabilidad general.

Por último, durante las aplicaciones del experimento se realizan grabaciones de la pantalla del computador, así como también del audio del lugar de aplicación. De esta manera, se respalda y comprueba la exactitud de las mediciones realizadas. Esto se realiza con el consentimiento explícito de cada participante. Cabe notar que el participante podía solicitar detener las grabaciones, o la eliminación de las grabaciones ya realizadas, en cualquier momento que cada participante estimara conveniente. Todos los procedimientos de recolección y manejo de datos se ajustaron a las consideraciones éticas generales del diseño experimental.

### 5.3.2. Validez interna

A pesar de tratarse de un caso de estudio único holístico exploratorio, cabe notar que los desafíos de programación propuestos en el diseño experimental son concordantes con el modelo conceptual de casos de uso propuesto. Se diferencian explícitamente casos de lectura y escritura, según la actividad que se solicita realizar al participante. Además, se diferencia entre bases de código conocidas y desconocidas. Esto se hace asumiendo que la base de código es conocida cuando se lee por segunda vez. De esta manera, los desafíos planteados son concordantes con el modelo conceptual utilizado. Con lo anterior, se tiene una relación



directa entre la unidad de análisis y el escenario experimental definido, lo cual minimiza las amenazas a la validez interna.

### **5.3.3. Validez externa**

Los resultados del presente caso de estudio no pueden ser generalizados más allá de la unidad de análisis. Esto se debe a que la muestra es extremadamente reducida, y no debe ser considerada como representativa de la población. A través del método de reclutamiento de muestreo de bola de nieve aplicado no se encontraron, en Chile, más personas ciegas con experiencia previa programando. Por otro lado, las condiciones de salud pública, al momento de la aplicación experimento, restringían los aforos y desplazamientos interregionales e internacionales. Con esto, la mayor amenaza a la validez del estudio es su validez externa. En un trabajo futuro, para atenuar esta amenaza a la validez, se debería replicar el estudio con una mayor cantidad de personas.

### **5.3.4. Validez ecológica**

Una vez definido el diseño experimental, se validó con programadores ciegos que las condiciones necesarias para la aplicación del experimento fueran las más cercanas a las condiciones en que ellos programan. Se validó que estuvieran familiarizados con el *hardware* a utilizar; se instó a utilizar sus audífonos personales; y se instó a utilizar su configuración personal del lector de pantalla de preferencia. Complementariamente, se procuró mantener el silencio y una conexión a internet estable en el lugar de aplicación del experimento. Todo esto, en su conjunto, aproxima las condiciones normales en que los participantes programan. Sin embargo, el ambiente controlado de laboratorio es en sí una limitación a la validez ecológica.

En un trabajo futuro, para paliar esta limitación a la validez, se debería replicar el estudio en un ambiente más natural y menos controlado. Esto se podría lograr aplicando el caso de estudio en los mismos lugares en que los participantes programan, utilizando sus propios computadores.

### **5.3.5. Confiabilidad**

Para garantizar la confiabilidad de los instrumentos utilizados, el estudio fue piloteado con uno de los participantes antes de su aplicación formal. Esto permitió reducir posibles ambigüedades en las instrucciones, que a su vez podrían producir errores de interpretación de los desafíos planteados. El sujeto que participó del piloto no participó de la aplicación formal posterior.

Una limitación a la confianza es que, al tratarse de un solo participante en la aplicación piloto, y un solo participante en la aplicación formal, no se puede asegurar la concordancia de respuestas entre múltiples sujetos para cada instancia de aplicación. Esto puede ser abordado como trabajo futuro, replicando la aplicación formal, pero con más participantes, para luego analizar la correlación entre sujetos de las respuestas proporcionadas.

### 5.3.6. Replicabilidad

Para el levantamiento de información del escenario experimental se elaboró un protocolo experimental. El protocolo detalla las instrucciones que se deben dar a los participantes al momento de realizar el experimento, los procesos de levantamiento de la información, los momentos de inicio y fin de las mediciones de tiempo, y las escalas utilizadas para medir la satisfacción y el esfuerzo percibido. El anexo B.1.2 contiene el protocolo utilizado en la ejecución final del experimento. Esto facilita que las operaciones realizadas durante este estudio puedan ser replicadas, esperando obtener resultados similares.

## 5.4. Instrumentos

Para garantizar que los participantes cumplan con los criterios de inclusión y exclusión, se creó un cuestionario de autorreporte a medida. Este cuestionario indaga en la percepción del participante sobre sus habilidades programando, y su experiencia previa con Python y NVDA. En el anexo B.1.1 se presenta una captura de pantalla del cuestionario utilizado.

Para las mediciones de eficacia, eficiencia, satisfacción y esfuerzo percibido por el usuario, se diseñó un protocolo experimental a medida. El protocolo solicita al participante realizar 12 desafíos de programación, relacionados directamente con las partes 1 a 12 del diseño experimental. Para cada desafío se mide el nivel de eficacia, como un porcentaje en el rango 0 % a 100 % según la relación entre la respuesta dada y el resultado esperado. Adicionalmente, se mide la eficiencia como la razón entre el tiempo utilizado para resolver el desafío, y tanto la cantidad de líneas como la cantidad de caracteres del archivo de código asociado. Al finalizar cada desafío, se pregunta al participante, en escala de 1 a 5, el nivel de esfuerzo percibido necesario para realizarlo. Al finalizar los 12 desafíos, se pregunta al participante sobre su satisfacción al utilizar cada configuración de la extensión. Se pregunta sobre la satisfacción general para cada configuración en escala de 1 a 5. Adicionalmente, se aplica la Escala de Usabilidad de Sistemas (*System Usability Scale*) [8] para cada configuración. En el anexo B.1.2 se presenta una transcripción del protocolo experimental.

Los desafíos de programación hacen referencia a un conjunto particular de archivos de código en el lenguaje Python. Estas piezas de código fueron diseñadas específicamente para el caso de estudio. Asimismo, la extensión de VSC que provee las configuraciones de NVCs fue creada a medida, con estos archivos de código en mente, de tal forma que se pueda emular la detección del alcance de las palabras reservadas.

Tanto la base de código utilizada en el caso de estudio, como la extensión que permite la transformación de señales visuales en no visuales, no son públicas. Previo a la realización del experimento, estas son desconocidas para el participante.

Las figuras 5.1 y 5.2 esquematizan la implementación del diseño experimental a través del protocolo. Se indica la cantidad de requerimientos, el tipo de actividad, los archivos necesarios de la base de código, y la configuración de la extensión utilizada. Las flechas indican que el archivo necesario en la sección siguiente es similar, pero con leves modificaciones.

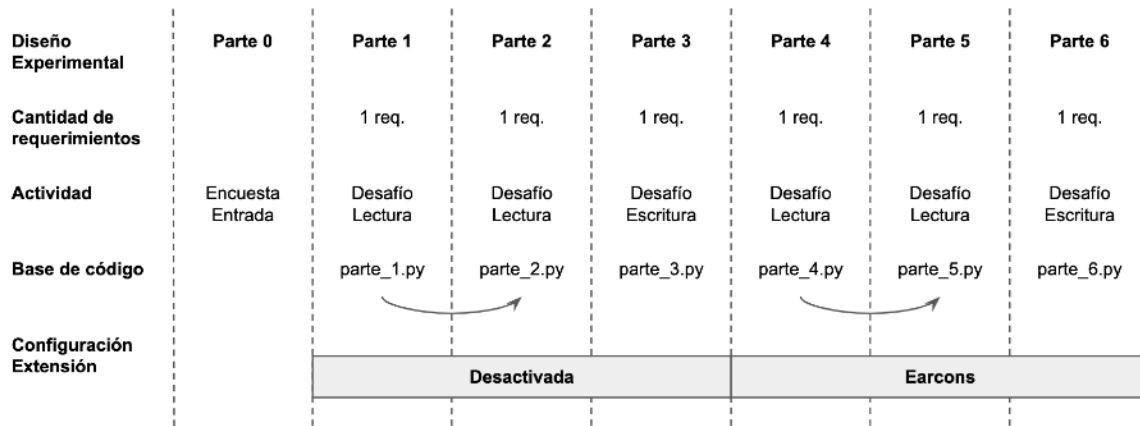


Figura 5.1: Esquema resumen sobre la implementación del diseño experimental. Parte 0 a Parte 6.

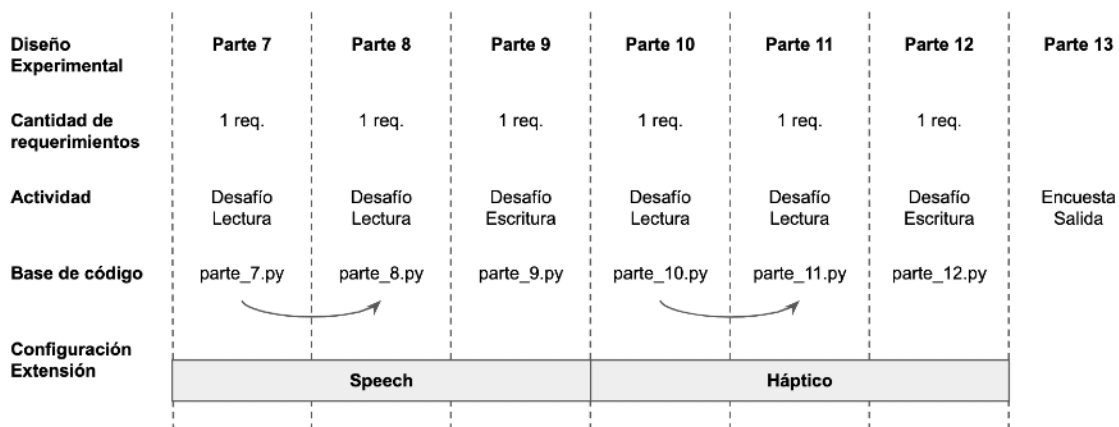


Figura 5.2: Esquema resumen sobre la implementación del diseño experimental. Parte 7 a Parte 13.

## 5.5. Participantes

El diseño experimental requiere que el participante esté familiarizado con algunos tecnicismos de computación. También es necesario que la persona pueda consentir la utilización de los datos recolectados. Por esto, fue necesario reclutar participantes adultos, ciegos, que declarasen tener experiencia previa programando. Además, cabe explicitar que era necesario ser capaz de interactuar con un computador a través de un ratón y teclado; poder utilizar, durante el experimento, una pulsera inteligente; e, idealmente, estar en el mismo lugar geográfico que el facilitador el experimento, para así garantizar las condiciones de validez ecológica.

El proceso de reclutamiento de participantes para el experimento siguió el método bola de nieve. Inicialmente se contactó a múltiples personas ciegas, referidas por familiares y amigos. Esto se hizo directamente, a través de correo electrónico, llamada telefónica, y/o mensaje

instantáneo; e indirectamente, a través de las principales instituciones educativas enfocadas en personas ciegas en Chile: Colegio Hellen Keller y Colegio Santa Lucía. Este proceso no fue exitoso, ya que ninguno de los referidos contaba con conocimientos básicos de programación. En una segunda búsqueda, a través del contacto con instituciones de educación superior, se contactó a dos potenciales participantes que cumplían con los criterios. Ambos candidatos eran estudiantes de las carreras de ingeniería civil en computación o informática de nivel universitario, a los cuales se aplicó el cuestionario de autorreporte. Esto verificó su conocimiento y nivel de dominio de las herramientas tecnológicas necesarias para el experimento.

Antes de la aplicación, los participantes fueron informados de los objetivos del experimento, así como también que sus resultados serían publicados en la presente tesis. Los dos sujetos participaron del experimento de manera libre y voluntaria. Ambos participantes leyeron y firmaron un documento de consentimiento informado. Adicionalmente, el día de la aplicación del experimento, se leyó en voz alta el consentimiento informado y se solicitó autorización para grabar el audio de la sala, y la pantalla del computador utilizado para la resolución de los desafíos de programación,

La realización del estudio, con estos participantes, fue aprobada y visada por el comité de ética de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. En el anexo B.2.1 se incluye el certificado emitido por dicho comité.

## 5.6. Aplicación

El experimento se ejecutó en dos instancias, una con cada participante. La primera aplicación consistió en una aplicación piloto. El objetivo de esta instancia era verificar que el protocolo experimental estuviera completo, así como también evaluar que se pudieran garantizar las condiciones de aplicación, por ejemplo, la estabilidad de las conexiones inalámbricas. La segunda aplicación corresponde a la ejecución efectiva del experimento, sobre la cual se reportan los resultados. En cada instancia de aplicación se dispuso de 2 horas para la realización de las actividades.

Ambas aplicaciones se realizaron en ambientes de laboratorio, procurando que la aplicación del experimento no se viera interrumpida. Se preparó el lugar del experimento, de manera tal que se encontrara a disposición del usuario: un computador portátil en el cual el sujeto realiza los desafíos de programación, un computador portátil como servidor HTTP y Bluetooth según requiere la extensión de VSC, y una pulsera MiBand 3 para retroalimentación háptica. Cabe mencionar que el computador portátil con el que el usuario interactúa cuenta con VSC versión 1.57 y Python versión 3.8.9 instalados. Además, el ambiente de ejecución de VSC solo tiene instalada y activa extensión especificada en el capítulo 4, que implementa NVCs. Con la finalidad de controlar las variables asociadas a hardware de soporte, los dispositivos utilizados fueron los mismos tanto en la aplicación piloto como en la aplicación efectiva. Las especificaciones técnicas de cada pieza de hardware utilizada en el caso de estudio único se encuentra en la sección B.3 de los anexos.

Antes de comenzar el experimento, se dio tiempo al participante para familiarizarse con el computador portátil provisto, e instalar su configuración de NVDA. Cada participante utilizó

su propia configuración de NVDA portable, además de audífonos personales. Posteriormente, se leyó el consentimiento informado y se solicitó autorización para grabar la pantalla y el audio del computador, así como también el audio de la sala. Una vez obtenida la autorización, se procedió a iniciar las grabaciones.

Durante el experimento, el facilitador leyó desde el protocolo experimental las instrucciones para la realización del experimento y para el uso de la extensión. También se asignó un espacio de 10 minutos, al inicio del experimento, para que el participante se familiarizara con la configuración de VSC. Al principio de cada desafío de programación se inició un cronómetro cuando el participante indicó estar listo para enfrentarlo, y se detuvo cuando el participante declara haberlo resuelto. El facilitador además garantizó que la configuración de la extensión fuera la adecuada para cada etapa del experimento.

Ya que las aplicaciones fueron realizadas durante el periodo de desconfinamiento de la pandemia por Coronavirus SARS-CoV-2, se siguieron todos los protocolos de higiene establecidos por el Ministerio de Salud para la fase 3 [13]. Esto implica: distanciamiento interpersonal, uso de mascarilla y alcohol gel, lavado de manos, y respeto del aforo máximo en lugares cerrados.

## **5.7. Análisis de respuestas**

Se realizaron dos tipos de análisis sobre las respuestas de la prueba de usabilidad. Se analizó cuantitativamente las mediciones de eficiencia, eficacia, satisfacción y esfuerzo percibido en los desafíos de programación. Estos corresponden a las Parte 1 a 13 del diseño experimental. Complementariamente se analizó cualitativamente las respuestas de las preguntas abiertas y discusión sostenida al final del experimento. Esto corresponde a la Parte 14 del diseño experimental.

### **5.7.1. Análisis cuantitativo**

Este análisis se realiza sobre las respuestas a las Partes 1 a 13 de la prueba de usabilidad. Esto corresponde a los 12 desafíos de programación y las mediciones de satisfacción con respecto a las configuraciones de NVCs

Sobre los 12 desafíos de programación se analiza la eficiencia, eficacia, y esfuerzo percibido. En los casos de lectura, la eficacia se calcula como el porcentaje de ocurrencias de palabras clave detectadas por el participante, respecto de la cantidad de ocurrencias totales en el archivo. En los casos de escritura, la eficacia se define como 0 % si el usuario no identifica la línea en la que debe escribir, ni escribe la palabra reservada esperada; 50 % si el usuario solo identifica la línea en la que debe escribir o solo escribe la palabra reservada esperada; y 100 % si el usuario identifica la línea en la que debe escribir y escribe la palabra reservada esperada. Con respecto a la eficiencia, se analiza el tiempo total requerido para entregar una solución al desafío, el tiempo ponderado por línea en el archivo de código asociado al desafío, y el tiempo ponderado por carácter en el archivo de código asociado al desafío. Se asume entonces una mayor eficiencia en los desafíos asociados a las mediciones de menor tiempo por línea y/o menor tiempo por carácter. Respecto al esfuerzo percibido, se utilizó directamente

la respuesta del participante como medida.

La tabla 5.2 presenta las especificaciones de los desafíos de lectura, siendo “NVC” el tipo de configuración de señal no-visual utilizada, “Conocida?” si la base de código se considera familiar para el participante, “Líneas” la cantidad de líneas de comentario o código en el archivo de código asociado al desafío, “Caráct” la cantidad de caracteres totales en el archivo de código asociado al desafío, y “Solución” la respuesta esperada con respecto a la cantidad de ocurrencias de palabras reservadas que plantea el desafío.

Tabla 5.2: Especificaciones de problemas de lectura de código.

| Desafío | NVCs         | Conocida? | Líneas | Caráct | Solución |
|---------|--------------|-----------|--------|--------|----------|
| 1       | Desactivadas | FALSE     | 43     | 1362   | 10       |
| 2       | Desactivadas | TRUE      | 47     | 1447   | 9        |
| 4       | Earcons      | FALSE     | 56     | 2215   | 3        |
| 5       | Earcons      | TRUE      | 54     | 2115   | 3        |
| 7       | Speech       | FALSE     | 93     | 2909   | 14       |
| 8       | Speech       | TRUE      | 93     | 2909   | 9        |
| 10      | Háptico      | FALSE     | 48     | 1678   | 5        |
| 11      | Háptico      | TRUE      | 48     | 1677   | 3        |

La tabla 5.3 presenta las especificaciones de los desafíos de escritura, siendo “NVC” el tipo de configuración de señal no-visual utilizada, “Conocida?” si la base de código se considera familiar para el participante, “Líneas” la cantidad de líneas de comentario o código en el archivo de código asociado al desafío, “Caráct” la cantidad de caracteres totales en el archivo de código asociado al desafío, y “Solución” la respuesta esperada con respecto a la línea en que se debe reemplazar un comentario o resolver un problema de sintaxis, y la palabras reservadas que se debe escribir o corregir.

Tabla 5.3: Especificaciones de problemas de escritura de código.

| Desafío | NVCs         | Conocida? | Líneas | Caráct | Solución          |
|---------|--------------|-----------|--------|--------|-------------------|
| 3       | Desactivadas | FALSE     | 9      | 170    | línea 13, None    |
| 6       | Earcons      | FALSE     | 23     | 705    | línea 17, return  |
| 9       | Speech       | FALSE     | 19     | 463    | línea 26, else    |
| 12      | Háptico      | FALSE     | 16     | 407    | línea 9, continue |

Con respecto a las mediciones generales de satisfacción por configuración de NVC, asociada a la Parte 13 del diseño, se utilizó directamente la respuesta del participante como medida. El análisis de las respuestas de cada aplicación del instrumento SUS se hace de la manera estándar, obteniendo un puntaje en escala de 0 a 100 [8].

## 5.7.2. Análisis cualitativo

Tanto las respuestas obtenidas en la Parte 14 del experimento como los comentarios emitidos durante el transcurso de los desafíos de programación fueron usados para interpretar los resultados obtenidos en las Partes 1 a 13. Con estas respuestas se indaga en las dificultades encontradas por el participante al utilizar cada tipo de configuración de la extensión, así como también en los procesos más importantes realizados para encontrar las soluciones de los desafíos. Para esto, se consideran los juicios de valor emitidos y las estrategias descritas por los participantes, y se contrastan con las acciones realizadas durante la prueba de usabilidad.

## 5.8. Resultados e interpretación

A continuación se presentan las respuestas e interpretan los resultados respecto a eficacia, eficiencia y esfuerzo percibido en los 12 desafíos de programación. La interpretación se realiza diferenciando según el caso de uso, ya sea este de lectura o escritura.

### 5.8.1. Casos de uso de lectura

La tabla 5.4 a continuación presenta las respuestas y mediciones de los desafíos de lectura de código. En esta tabla la columna “Respuestas” contiene la respuesta entregada para el desafío por el participante con respecto a la cantidad de palabras reservadas identificadas; la columna “t [s]” indica el tiempo, en segundos, utilizado por el participante; la columna “Esfuerzo” contiene la respuesta entregada con respecto al esfuerzo percibido al final del desafío; la columna “Eficacia” indica la medición de eficacia que, en este caso, corresponde a la razón, en porcentaje, entre la respuesta y solución del desafío; la columna “t/Línea” corresponde la medición de tiempo ponderada por la cantidad de líneas de código en el archivo asociado al desafío; y la columna “t/Caráct” corresponde la medición de tiempo ponderada por la cantidad de caracteres en el archivo asociado al desafío.

Tabla 5.4: Resultados de problemas de lectura de código.

| Desafío | Respuesta | t [s] | Esfuerzo | Eficacia | t/Línea | t/Caráct |
|---------|-----------|-------|----------|----------|---------|----------|
| 1       | 8         | 135   | 2        | 80 %     | 3.1395  | 0.0991   |
| 2       | 9         | 123   | 2        | 100 %    | 2.6170  | 0.085    |
| 4       | 3         | 114   | 2        | 100 %    | 2.0357  | 0.0515   |
| 5       | 3         | 93    | 2        | 100 %    | 1.7222  | 0.044    |
| 7       | 14        | 240   | 3        | 100 %    | 2.5806  | 0.0825   |
| 8       | 9         | 201   | 2        | 100 %    | 2.1613  | 0.0691   |
| 10      | 5         | 206   | 2        | 100 %    | 4.2917  | 0.1228   |
| 11      | 3         | 112   | 2        | 100 %    | 2.3333  | 0.0668   |

El participante resolvió la totalidad de los desafíos de lectura. En estas actividades se

reporta un esfuerzo promedio de 2.125, eficacia promedio de 98 %, tiempo por línea promedio de 2.6102 segundos, y tiempo por carácter promedio de 0.0776 segundos.

Con respecto a la eficacia, para todos los desafíos de lectura, excepto el Desafío 1, el participante entrega la respuesta correcta. Dado que tanto el Desafío 1 como el Desafío 2 corresponden a la lectura de bases similares de código, se puede considerar la eficacia promedio entre ellos, 90 %, como la eficacia de los desafíos de lectura con NVCs desactivadas. Esto se interpreta como que el uso de NVCs, independiente del tipo de retroalimentación, mejora la eficacia en la solución de desafíos de programación. Esto responde parcialmente la primera pregunta de investigación.

Con respecto a la eficiencia, todos los desafíos de lectura en que el participante utilizó NVCs activas reportan menor tiempo por línea y tiempo por carácter, a excepción del Desafío 10. Este desafío tiene la particularidad de ser el primero en que el participante utilizó la retroalimentación háptica. Al observar la eficiencia en el desafío siguiente, Desafío 11, que se realizó con el mismo tipo de retroalimentación, se nota que el tiempo vuelve a disminuir por debajo de los casos con NVCs inactivas. Con esto, se interpreta que el proceso de familiarización con el sistema de retroalimentación háptica puede disminuir la eficiencia del programador. Una vez se familiariza con el sistema, la eficiencia vuelve a ser mejor que con NVCs inactivas. Adicionalmente, la mayor eficiencia, tanto por línea de código como por carácter, se logra utilizando retroalimentación en base a *earcons*. Esto podría deberse a la baja latencia del estímulo, en contraste con la retroalimentación háptica, o bien, a la discernibilidad del estímulo, en comparación con la retroalimentación en base a texto hablado. Esto responde parcialmente la primera pregunta de investigación.

Con respecto al esfuerzo percibido, todos los desafíos reportan el mismo nivel, a excepción del Desafío 7. Este desafío, junto con el Desafío 8, tienen la particularidad de ser los que tienen asociado el archivo con el mayor número de líneas. Dado que el Desafío 8 tiene el mismo número de líneas, y utiliza la misma configuración de NVCs, se interpreta que el cambio en el nivel de esfuerzo percibido se debe a la novedad del tamaño del archivo. Por el contrario, no se atribuye este comportamiento al hecho de que la base de código sea familiar, ya que este efecto no se ve en los otros pares de desafíos. Se interpreta entonces que el uso de NVCs, independiente del tipo de retroalimentación, no tiene un efecto sobre el nivel de esfuerzo percibido. Esto responde parcialmente la primera pregunta de investigación.

En síntesis, se infiere que las NVCs mejoran la eficacia y eficiencia, sin requerir un mayor nivel de esfuerzo, por parte de un programador ciego. Esto es válido para problemas de lectura de código, y es independiente de la familiaridad que el programador tenga con la base.

### 5.8.2. Casos de uso de escritura

La tabla 5.5 a continuación presenta las respuestas y mediciones de los desafíos de escritura de código. En esta tabla la columna “Respuestas” contiene la respuesta entregada para el desafío por el participante con respecto a las líneas modificadas del archivo asociado al desafío; la columna “t [s]” indica el tiempo, en segundos, utilizado por el participante; la columna “Esfuerzo” contiene la respuesta entregada con respecto al esfuerzo percibido al final del



desafío; la columna “Eficacia” indica la medición de eficacia que, en este caso, corresponde a la razón, en porcentaje, entre la respuesta y solución del desafío; la columna “t/Línea” corresponde la medición de tiempo ponderada por la cantidad de líneas de código en el archivo asociado al desafío; y la columna “t/Caráct” corresponde la medición de tiempo ponderada por la cantidad de caracteres en el archivo asociado al desafío.

Tabla 5.5: Resultados de problemas de escritura de código.

| Desafío   | Respuesta            | t [s] | Esfuerzo | Eficacia | t/Línea | t/Caráct |
|-----------|----------------------|-------|----------|----------|---------|----------|
| <b>3</b>  | línea 13<br>línea 29 | 80    | 1        | 100 %    | 8.8889  | 0.4706   |
| <b>6</b>  | línea 30<br>línea 31 | 103   | 1        | 0 %      | 4.4783  | 0.1461   |
| <b>9</b>  | línea 26             | 120   | 1        | 100 %    | 6.3158  | 0.2592   |
| <b>12</b> | línea 9              | 84    | 1        | 100 %    | 5.25    | 0.2064   |

El participante resolvió la totalidad de los desafíos de escritura. En estas actividades se reporta un esfuerzo promedio de 1, eficacia promedio de 75 %, tiempo por línea promedio de 6.2333 segundos, y tiempo por carácter promedio de 0.2795 segundos.

Con respecto a la eficacia, para todos los desafíos de escritura, excepto el Desafío 6, el participante entrega la respuesta correcta. El Desafío 6 fue errado por el participante ya que el lector de pantalla no le permitía identificar que la palabra “Return” estaba escrita con mayúscula sin realizar una lectura carácter a carácter. Además, el participante no utilizó la extensión en esa ocurrencia, por lo que no se desencadenó la NVC. Dado lo anterior, se interpreta como que el uso de NVCs, independiente del tipo de retroalimentación, no tiene un efecto respecto de la eficacia en la solución de desafíos de programación. Esto responde parcialmente la segunda pregunta de investigación.

Con respecto a la eficiencia, todos los desafíos de escritura en que el participante utilizó NVCs activas reportan menor tiempo por línea y tiempo por carácter. La mayor eficiencia, tanto por línea de código como por carácter, se logra utilizando retroalimentación en base a *earcons*. Análogamente a los casos de lectura, esto podría deberse a la baja latencia del estímulo, en contraste con la retroalimentación háptica, o bien, a la discernibilidad del estímulo, en comparación con la retroalimentación en base a texto hablado. Con esto se responde parcialmente la segunda pregunta de investigación.

Con respecto al esfuerzo percibido, todos los desafíos reportan el mismo nivel. Se interpreta entonces que el uso de NVCs, independiente del tipo de retroalimentación, no tiene un efecto sobre el nivel de esfuerzo percibido. Esto responde parcialmente la segunda pregunta de investigación.

En síntesis, se infiere que las NVCs mejoran solo la eficiencia, sin requerir un mayor nivel de esfuerzo, por parte de un programador ciego. Esto es válido para problemas de escritura de código, y es independiente de la familiaridad que el programador tenga con la base.

### 5.8.3. Optimalidad por tipo de retroalimentación

La tabla 5.6 a continuación presenta las respuestas y mediciones de satisfacción y usabilidad por tipo de retroalimentación. En esta tabla la columna “Satisfacción” contiene la respuesta entregada por el participante respecto de su nivel de satisfacción al utilizar la extensión configurada con cada tipo de retroalimentación en una escala Likert de 1 a 5. La columna “SUS” contiene el resultado de la evaluación de usabilidad con el instrumento SUS. Los resultados de este último instrumento van en una escala de 0 a 100. Este valor se interpreta tal que un mayor puntaje corresponde a una mayor usabilidad del sistema, donde 68 es el valor usualmente aceptado como cota mínima para poder indicar que un sistema es considerado como usable[8]. Esto permite comparar los tipos de retroalimentación utilizando este parámetro.

Tabla 5.6: Resultados de análisis satisfacción y SUS.

|                     | Satisfacción | SUS   |
|---------------------|--------------|-------|
| <b>Desactivadas</b> | 5            | -     |
| <b>Earcons</b>      | 5            | 87.50 |
| <b>Speech</b>       | 5            | 77.50 |
| <b>Háptica</b>      | 3            | 55.00 |

Con respecto al nivel de satisfacción, todas las configuraciones de la extensión logran el puntaje mínimo de aprobación, excepto la retroalimentación háptica. En la entrevista posterior con el participante, este indica que se debe a la alta latencia que presenta el sistema. El tiempo que toma la interacción desde que el participante selecciona/clica/escribe una palabra clave, hasta que la pulsera vibra, es suficiente como para hacer pensar al participante que la palabra con la que se interactuó no es reservada para el lenguaje. Por esto, cuando se utilizó este modo de retroalimentación, el participante prefirió verificar que las palabras fueran reservadas leyendo carácter a carácter.

Con respecto a los resultados del instrumento SUS, el mayor puntaje lo obtiene la retroalimentación mediante *earcons*. Esto es consistente con los resultados para ambos tipos de actividades, en que es este el tipo de estímulo cuando se evalúa eficacia, eficiencia y esfuerzo percibido. En contrapartida, el menor puntaje tanto en la escala de satisfacción como en la escala SUS lo obtiene la retroalimentación háptica. Esto también es consistente con los resultados en las mediciones de eficacia, eficiencia y esfuerzo percibido. Se interpreta entonces que los resultados de la aplicación de estos instrumentos refuerzan los resultados de las mediciones presentadas en la subsección anterior, contribuyendo a responder las preguntas 1 y 2 de este caso de estudio.

Se consultó al participante sobre sus apreciaciones de los métodos de retroalimentación auditiva. El participante justifica su preferencia por la retroalimentación mediante *earcons*, mencionando que eran fáciles de distinguir del audio utilizado por el lector de pantalla. Con respecto a la retroalimentación con texto hablado, menciona que resulta más incómodo que los *earcons* ya que debe distinguir la palabra utilizada como retroalimentación de las

que narra el lector de pantalla. También influye en su incomodidad el que la velocidad de narración de la extensión sea menor que la velocidad de narración del lector de pantalla. La extensión no provee una configuración para la velocidad de narración, ni la puede obtener del lector de pantalla. Estas consideraciones afectaron, durante los casos de lectura de código, la accesibilidad de las NVCs, teniendo que repetirse el desencadenamiento de la misma.

En base a lo resultados expuestos, se puede terminar de responder la primera y segunda pregunta de investigación de este caso de estudio. La NVC óptima, tanto en los casos de lectura como de escritura, es la retroalimentación mediante *earcons*. Esta reporta la mayor eficacia, eficiencia, y satisfacción, y el menor nivel de esfuerzo percibido. Y, a partir de la entrevista final al participante, se desprende que esto se puede deber tanto a la menor latencia y mejor discernibilidad de este tipo de estímulo.

#### 5.8.4. Optimalidad por familiaridad con la base de código

Ya que eficacia y esfuerzo percibido no varían según el tipo de señal no visual, se procedió a analizar la variación en el tiempo por línea y tiempo por carácter en cada configuración de NVCs. En la tabla 5.7 continuación, la columna “dif t/Línea” indica la diferencia porcentual entre el tiempo por línea cuando la base es conocida con respecto a cuando es desconocida; y la columna “dif t/Caráct” indica la diferencia porcentual entre el tiempo por carácter cuando la base es conocida con respecto a cuando es desconocida.

Tabla 5.7: Diferencias de tiempo según conocimiento de la base de código.

|                    |                    | dif t/Línea | dif t/Carácter |
|--------------------|--------------------|-------------|----------------|
| <b>Problema 1</b>  | <b>Problema 2</b>  | 16.64 %     | 14.23 %        |
| <b>Problema 4</b>  | <b>Problema 5</b>  | 15.40 %     | 14.56 %        |
| <b>Problema 7</b>  | <b>Problema 8</b>  | 16.25 %     | 16.24 %        |
| <b>Problema 10</b> | <b>Problema 11</b> | 45.63 %     | 45.60 %        |

De estos resultados, se desprende que la mejora en la eficiencia es similar, alrededor de 15 % cuando se pasa de una tarea sobre una base código desconocida a una tarea sobre la misma base de código, desconocida. Sin embargo, esta diferencia en la eficiencia no se podría atribuir al uso de NVCs, ya que cuando las NVCs están inactivas, la variación en la eficiencia es la misma.

En el caso de la retroalimentación háptica, se observa una mayor variación en la eficiencia, del 45 %. Sin embargo, se debe considerar que la eficiencia cuando la base de código era desconocida, era la más baja, peor incluso al caso de NVCs apagadas. Además, una vez que el participante se familiariza con la extensión y la base de código, no se logra una eficiencia significativamente mayor al promedio.

En base a lo anterior, se desprende que la variación en la eficiencia, debido a la familiarización con la base de código, es independiente del tipo de NVC utilizada. Con esto, se puede responder la tercera pregunta de investigación de este caso de estudio. No se puede distinguir

un tipo de NVC óptima según si el caso de uso sea de lectura, y potencialmente de escritura, entre una base de código conocida o desconocida.

## 5.9. Conclusiones

La metodología de caso de estudio único permitió explorar las implicancias del uso de distintas NVCs como mecanismos para la transmisión de información contextual del código. Se eligió esta metodología ya que la población de programadores ciegos en Chile es extremadamente reducida, ya que gran parte de las personas con esta discapacidad no logra acceder a la educación superior [27].

Para dotar de validez al estudio, la metodología requiere controlar múltiples variables: se debe definir correctamente la unidad de análisis, se debe utilizar instrumentos apropiados para cada medición, se debe seguir un protocolo experimental, y se debe aproximar a las condiciones normales en que ocurriría el fenómeno estudiado. El diseño experimental de este estudio cumple con todos los requisitos metodológicos. Además, se realizó una aplicación piloto con uno de los participantes. Esto refuerza la validez de constructo del estudio. A pesar de lo anterior, el estudio cuenta con fuertes amenazas a su validez externa, ya que la muestra es extremadamente reducida, por lo que los resultados obtenidos no deben ser generalizados.

El caso de estudio permitió profundizar el entendimiento sobre los procesos y estrategias de programadores ciegos para la resolución de tareas de desarrollo. En el capítulo 3 se valida empíricamente el aporte de las VCs en los procesos de lectura y escritura de código por parte de programadores videntes. Esto es consistente con los resultados obtenidos en el caso de estudio, en que la eficiencia de programadores ciegos es consistentemente mejor cuando se utilizan NVCs. Esto significa que la implementación de mecanismos no-visuales para la transmisión de información contextual del código a programadores ciegos puede tener un efecto positivo en la accesibilidad y usabilidad de IDEs, herramientas fundamentales en los procesos de desarrollo.

En base a los resultados, se aprecian efectos cuantificables de las NVCs respecto de la usabilidad para la unidad de análisis. Esto permite sugerir las NVCs óptimas para cada caso de uso. Con esto se da respuesta a las preguntas de investigación 1 y 2 del caso de estudio. Respecto de la pregunta de investigación 1: “¿Cuál es la señal no-visual de código de mayor eficacia, eficiencia y satisfacción, y menor esfuerzo percibido en la resolución de problemas de lectura de código?” Los resultados sugieren la retroalimentación en base a *earcons* como la NVC óptima en casos de uso de lectura. Respecto de la pregunta de investigación 2: “¿Cuál es la señal no-visual de código de mayor eficacia, eficiencia y satisfacción, y menor esfuerzo percibido en la resolución de problemas de escritura de código?” Los resultados también sugieren la retroalimentación en base a *earcons* como la NVC óptima. En su conjunto, estas respuestas refuerzan la validez de la hipótesis H1 de esta tesis: “La optimalidad de una señal de código depende del caso de uso.”, ya que es a partir de esta dependencia que se obtienen variaciones en las mediciones que permiten sugerir un tipo de NVC en particular.

La entrevista final con los participantes permite entender las características que hacen a

una NVC óptima por sobre otras. La capacidad de distinguirse de la retroalimentación del lector de pantalla, y la latencia con la que se percibe el estímulo son factores fundamentales. Considerando esto, futuras implementaciones de la extensión en que se utilicen mecanismos hápticos de menor latencia, podrían presentar mediciones de usabilidad superiores. Esto puede ser fundamental en los casos de lectura, en que la demanda cognitiva por estímulos auditivos es superior.

El conjunto de NVCs utilizado es acotado. En base al estudio de la literatura se reconocen otros tipos de señales auditivas, como *spearcons*, y otros periféricos táctiles como CodeSkimmer. Se puede entonces ampliar las capacidades de la extensión para soportar estos y otros métodos de retroalimentación, y encontrar las NVCs óptimas para cada caso de uso en base al nuevo conjunto. La retroalimentación por *spearcons* no se evaluó debido a que la literatura sugiere que no sería optimal. Con respecto al uso de CodeSkimmer, no se contaba con el hardware necesario, ya que es hecho a medida y de propiedad de los investigadores que reportaron sobre su uso. Para el conjunto utilizado en este estudio, las NVCs basadas en *earcons* superan las mediciones de usabilidad de NVCs desactivadas, son accesibles, y se apreció en la realización del experimento la capacidad para desencadenarlos correctamente según el caso de uso. Con esta evidencia se valida la hipótesis H2 de la presente tesis: “*La información contextual transmitida por una señal visual del código puede ser transmitida por una señal no-visual del código óptima, desencadenada según el caso de uso*”.

A pesar de que la interpretación de los resultados permita validar las hipótesis de la presente tesis, el estudio tiene limitaciones. La cantidad de participantes es baja, lo que representa la principal amenaza a la validez externa del estudio. Por esto, se propone como parte del trabajo futuro la replicación del estudio, pero con una mayor cantidad de participantes. Se sugiere una muestra de alrededor de 15 participantes, basado en el trabajo relacionado enfocado en la misma población objetivo [4, 5, 6, 7]. Esto se podría realizar con un plan de reclutamiento de participantes en otras regiones del país y latinoamerica, siguiendo una estrategia más agresiva de *snowballing* y contactos dirigidos.

# Capítulo 6

## Consideraciones de Diseño

El estudio de la literatura permite entender los desafíos que un programador ciego enfrenta en los distintos procesos de desarrollo de software. Para superar estos desafíos, es fundamental que las aplicaciones que utilicen, como lectores de pantalla y ambientes de desarrollo integrado, sean accesibles.

En este capítulo se presentan consideraciones de diseño enfocadas en la transmisión óptima de información contextual del código a programadores ciegos. Estas consideraciones se hacen en base a los aprendizajes y resultados del caso de estudio, el proceso de implementación de VisualCueTransform, y el análisis del trabajo relacionado. Se debe notar que la evidencia empírica observada a través del caso de estudio no puede ser generalizada al resto de la población, dado el tamaño reducido de la muestra. El capítulo cuenta con tres secciones, una para cada elemento computacional relevante en el proceso de transmisión.

### 6.1. Señales no-visuales de código

Los resultados del caso de estudio único sugieren que, en el caso de programadores ciegos, el uso de NVCs para la transmisión de información contextual del código, mejora la accesibilidad y usabilidad del IDE. Esto es válido con respecto a la realización de tareas de lectura y escritura de código.

Entre el conjunto de NVCs evaluadas en el caso de estudio, los *earcons* son el mecanismo óptimo. Sin embargo, no todos los sistemas cuentan con las mismas capacidades para la generación de señales. Por esto, a continuación se listan las características más relevantes para la elección de mecanismos no-visuales a utilizar, y cómo estas se relacionan con los hallazgos.

- **Accesibilidad:** la señal debe ser perceptible por el usuario. En el caso de programadores ciegos, se deben utilizar mecanismos que no dependan de la vista. Un ejemplo de esto son mecanismos basados en sonido, como *earcons*, *spearcons* y *texto hablado*; o mecanismos basados en el tacto (o en general, el sistema somatosensorial) como relieve, vibraciones, y temperatura. Otros sistemas sensoriales, como gusto, olfato y equilibrio (sistema vestibular) podrían ser utilizados, sin embargo, son alternativas no convencionales.

En el caso de estudio único, se exploró la accesibilidad de las NVCs usadas a través de entrevistas con los participantes. Así, los participantes describieron los *earcons* como accesibles ya que eran usables, no interferían con el lector de pantalla NVDA, y permitían acceder a la información contextual del código de manera efectiva.

- Oportunidad: se debe desencadenar el estímulo correcto en el momento oportuno. Se debe capturar correctamente los eventos que se definan como desencadenantes. Para esto, se debe considerar que la navegación convencional sobre una base de código, en un lenguaje de programación basado en texto, es línea a línea. Eventos relevantes son el comienzo y fin de la lectura de una nueva línea; en inicio y fin de la lectura de una cadena particular; la selección de una porción del código; el ingreso de un nuevo carácter; el ingreso de un carácter separador; y el ingreso de un salto de línea. En general, la lectura y escritura por parte de programadores ciegos suele realizarse a través del teclado, prescindiendo del uso del ratón [4].

El estímulo desencadenado debe ser el que el programador espera y entiende como propio para la retroalimentación respecto del evento desencadenante y la señal del código transmitida. Esto se concretó en el caso de estudio a través de la exploración previa de las configuraciones de la extensión. En esta instancia el programador logró establecer la asociación entre los eventos desencadenantes: clic, selección y término de escritura, con los distintos sonidos y vibraciones que VisualCueTransform produce. De esta manera, durante la realización de los desafíos, cuando se emitía un *earcon* después de escribir una palabra, el programador sabía que la palabra escrita se consideraba reservada para el lenguaje.

- Celeridad: el estímulo debe ser percibido poco tiempo después del desencadenamiento, y su duración debe ser acotada. Esto permite al programador establecer la relación entre el evento desencadenante, y la NVC desencadenada. Acotar la duración del estímulo restringe la posibilidad que se solape con otros, asociados a otros eventos, previniendo la sobrecarga cognitiva.

En el caso de estudio, esto se hizo limitando la duración de la reproducción de los archivos de audio a un segundo. Sin embargo, en la entrevista final, los participantes mencionaron que la alta latencia desde que se interactuaba con una palabra clave hasta que se producía la retroalimentación háptica sería la razón fundamental para que este mecanismo no fuera adoptado.

Esto se pudiera mejorar utilizando estándares de menor latencia, como Bluetooth 5.2 en vez de dispositivos que dependen de BLE, como es el caso de la MiBand 3.

- Discernibilidad: el estímulo se debe distinguir de otros estímulos. Se debe considerar que múltiples estímulos pueden ocurrir en simultáneo. Por ejemplo, la NVC al seleccionar una palabra reservada, ocurre simultáneamente con la lectura de la palabra por parte del lector de pantalla. En este caso, el programador debe poder diferenciar cuál es el estímulo de cada emisor.

Esto se puede conseguir estudiando de antemano que los estímulos utilizados no formen parte de la biblioteca de sonidos del lector de pantalla. En el caso de estudio único, los *earcons* exhibieron esta característica ya que el sonido de campana que se utilizó no coincidía con ningún sonido utilizado por NVDA o el IDE. En contrapartida, la

retroalimentación utilizando texto hablado se confundía fácilmente con la voz artificial del lector de NVDA. Según los participantes, esto ocurría debido a la similitud en el tono de voz de ambos estímulos.

En síntesis, en la aplicación del caso de estudio, todas estas características se encontraron en los *earcons*. Se diferenciaron de las señales basadas en texto hablado por su discernibilidad, ya que estas últimas se pueden confundir con el estímulo del lector de pantalla. También se diferencian de la retroalimentación háptica, pero por su celeridad, ya que la implementación a través de la pulsera MiBand 3 generaba una latencia adicional. Con esto, para computadores que lo permitan, se sugieren los *earcons* como el tipo de NVC predilecto.

## 6.2. Implementación de NVCs en IDEs

Los ambientes de desarrollo integrado pueden proveer de manera directa o indirecta los mecanismos para la accesibilidad de señales de código fuente. Se considera directa a la implementación de la captura de eventos desencadenantes, evaluación del estado del sistema al capturar el evento, y generación del estímulo no-visual. Se considera indirecta a la disponibilización de servicios que permitan implementar los mismos procesos a través de extensiones u otras aplicaciones.

Si se provee de manera directa, se debe tener en cuenta las consideraciones realizadas en la sección anterior, priorizando la utilización de NVCs accesibles, oportunas, rápidas y discernibles. Los resultados del caso de estudio sugieren los *earcons* como un estímulo apropiado que cumple con estas características. En el caso particular de VSC, VisualCueTransform es un ejemplo de una implementación directa. Se logra la captura de los eventos desencadenantes utilizando de la API del IDE, y se logra la emulación de la evaluación de alcance a través del análisis de sintaxis.

Si se provee de manera indirecta, se debe facilitar el acceso al registro de eventos en el editor de código; acceso al motor que implemente VCs, por ejemplo, en el caso VSC, el motor de tokenización, tematización e inspector de alcance; y servicios configurables para la generación de estímulos, por ejemplo, un servicio de transformación de texto en audio (*text-to-speech*) en que se pueda ajustar el idioma y un servicio de reproducción de audio en que se pueda ajustar la velocidad de la narración. Todos estos servicios se pudieran implementar en la práctica, por ejemplo, en VSC, a través de nuevos servicios y *end-points* en su API de ejecución.

## 6.3. Implementación de NVCs en lectores de pantalla

Los lectores de pantalla también pueden proveer de manera directa o indirecta los mecanismos para la accesibilidad de señales de código fuente mediante audio. Se considera directa a la implementación de la captura de eventos desencadenantes, evaluación del estado del sistema al capturar el evento, y generación del estímulo auditivo. Se considera indirecta a la



disponibilización de servicios que permitan implementar la captura de eventos y la generación del estímulo auditivo.

Si se provee de manera directa, se debe tener en cuenta las consideraciones realizadas en la sección anterior, priorizando la utilización de NVCs accesibles, oportunas, céleres y discernibles, como por ejemplo, *earcons*. Pero en este caso, además, se debe considerar los requisitos para la evaluación de sintaxis. Por esto, se requeriría implementar los servicios de tokenización y asignación de alcances para cada lenguaje de programación soportado. Esto se podría lograr, por ejemplo en el caso de NVDA, a través de extensiones que incorporen estos servicios.

Si se provee de manera indirecta, se debe facilitar a un servicio capaz de suscribirse a eventos generados por otra aplicación, que debe contener el texto o el audio a reproducir; y acceso al servicio de reproducción de audio. Se puede implementar la interacción con estos servicios a través de una API.

También se debe permitir realizar configuraciones con respecto al volumen y velocidad de reproducción para cada proveedor de eventos que se suscriba. En el caso de NVDA, esto se puede lograr a través de un archivo de configuraciones modificable por el usuario. De esta manera, el usuario puede decidir los parámetros que hagan céleres y discernibles céleres las señales utilizadas.

Se debe notar que este tipo de implementación no fue explorado en la presente tesis, debido a que NVDA depende de la utilización de la API de accesibilidad de Windows, y la comunicación por BLE depende de paquetes Unix. Por esto, los mecanismos evaluados no eran compatibles en un mismo sistema operativo. Debido a lo anterior, estas consideraciones solo se basan en el estudio de la documentación del lector de pantalla NVDA, utilizado en el caso de estudio, y no en la experiencia práctica implementando.

# Capítulo 7

## Conclusiones

Las señales de código fuente que proveen los IDE son fundamentales en los procesos de desarrollo. Estas facilitan las actividades de lectura y escritura de código al programador. Por esto, al diseñar un IDE, se deben considerar las necesidades de accesibilidad de todos sus usuarios. Hoy en día, las implementaciones de transmisión de información contextual del código que proveen los IDE suele considerar solo VCs.

Para hacer accesible la información contextual del código a programadores ciegos, se deben utilizar mecanismos de retroalimentación no-visual. La elección del tipo de señal no-visual a utilizar no debe ser arbitraria. Algunos tipos pueden interferir con los procesos y herramientas que el programador utiliza habitualmente al desarrollar, o ser indiscernibles en el contexto de uso. Además, se desconoce cómo se compara la usabilidad de los distintos tipos de mecanismos para las tareas de programación. Se infiere, que por estos vacíos en el estado del arte, no existen guías de diseño para hacer accesible la información contextual del código.

Se definió como objetivo general de la presente tesis: “*determinar NVCs que resulten óptimas para la transmisión de información contextual del código a programadores ciegos. Esto en el contexto del IDE Visual Studio Code, utilizando el lenguaje de programación Python.*”. Para abordarlo se plantearon los cuatro objetivos específicos que se discuten a continuación.

El primer objetivo específico: “*Identificar, describir y clasificar los tipos de VC, la información que entregan y las funciones que cumplen*” fue abordado a través del estudio del estado del arte y la documentación del sistema de retroalimentación visual de VSC. Como resultado de este proceso se identificaron las señales visuales del código, de la aplicación y del contexto, y la importancia de cada una en los procesos de desarrollo. Esto permitió acotar el alcance de la presente tesis a las señales visuales del código asociadas a palabras reservadas, ya que son elementos fundamentales para la navegación e interpretación de una base de código.

El segundo objetivo específico: “*Identificar, describir y clasificar los distintos casos de uso en que los programadores utilizan las VCs.*” fue abordado a través del estudio del estado del arte sobre exploración de código. A partir de esto, se planteó un modelo conceptual de casos de uso, que propone organizar las funcionalidades de las señales del código según la actividad realizada y la familiaridad con la base de código.

El modelo propuesto fue evaluado empíricamente a través de una prueba pareada con alumnos videntes, estudiantes de ingeniería civil en computación. A partir de los resultados de esta prueba, se evidenció que existe una diferencia estadísticamente significativa en el nivel de esfuerzo percibido por el programador, según si las tareas realizadas son de lectura o escritura de código. Con esto se confirmó que se pueden distinguir casos de uso según el tipo de tarea realizada, lo que a su vez validó parcialmente el modelo propuesto. Con respecto a la diferenciación de casos de uso por nivel de familiaridad con la base de código, no se apreció una diferencia estadísticamente significativa, con lo que se descartó esta componente del modelo propuesto. Por todo lo anterior, la prueba pareada realizada permitió probar la validez de la hipótesis H1: “*La optimalidad de una señal de código depende del caso de uso*”.

El tercer objetivo específico: “*Definir e implementar un mecanismo que permita transformar una VC particular en una NVC según la detección de un caso de uso.*” se logró completar a través de la implementación de VisualCueTransform, una extensión para VSC. Esta extensión permite capturar interacciones del usuario con la base de código, generando una NVC en respuesta. La extensión puede ser configurada para utilizar *earcons* (sonidos breves que se asocian a eventos particulares, por ejemplo, el *beep* de un celular al recibir un mensaje), texto hablado o vibraciones como métodos no-visuales de retroalimentación.

El cuarto objetivo específico: “*Explorar la optimalidad de los distintos tipos de NVCs como mecanismos de transmisión de información sobre el código según el caso de uso.*” se abordó a través de la realización de un caso de estudio único holístico exploratorio. En este participaron dos programadores ciegos, ambos estudiantes de carreras afines a la computación. En el caso de estudio se midió la optimalidad de NVCs basadas en *earcons*, texto hablado, y retroalimentación háptica utilizando la extensión VisualCueTransform.

El escenario experimental del caso de estudio consistió en que el participante debía resolver doce desafíos de escritura y lectura de código. En estos procesos, se midió la correctitud de las respuestas a cada desafío, el tiempo utilizado, el esfuerzo percibido por el participante, y la satisfacción con cada una de las NVCs. El análisis e interpretación de los resultados permitió probar la validez de la hipótesis H2: “*La información contextual transmitida por una señal visual del código puede ser transmitida por una señal no-visual del código óptima, desencadenada según el caso de uso.*” Esto, ya que se evidenció que las NVCs basadas en *earcons* son optimales, tanto para los casos de uso de escritura como de lectura de código. Además, se identificó la celeridad como una característica fundamental de las NVCs a utilizar por programadores ciegos. Por otro lado, a pesar de los malos resultados de las NVCs basadas en vibraciones, se reconoce el potencial de este tipo de retroalimentación en los casos de lectura de código. Esto se debe a que cuenta con las características idóneas para desempeñar este rol, siempre y cuando el mecanismo que la implemente pueda garantizar una baja latencia.

Basado en el estudio del estado del arte y en los resultados de los experimentos realizados, se presentaron consideraciones de diseño para la implementación de NVCs en ambientes de desarrollo integrado. Las consideraciones tratan sobre las características principales que deben tener las señales no-visuales utilizadas para la transmisión de información contextual del código: accesibilidad, oportunidad, celeridad y discernibilidad. Además, se discute sobre cómo se pueden implementar estos mecanismos, ya sea a través del IDE o de la aplicación de lectura de pantalla. Dado los resultados del caso de estudio, en las consideraciones de diseño

se sugiere el uso de *earcons* como NVC en VSC para el lenguaje Python, tanto para los casos de lectura como de escritura.

La metodología de caso de estudio holístico exploratorio permitió obtener e interpretar resultados relevantes sobre un fenómeno complejo con una muestra pequeña. El reclutamiento de participantes para el mismo fue complejo debido a lo acotada de la población de personas ciegas con estudios de programación en Chile. No obstante, los resultados obtenidos son prometedores en la medida que dan primeras luces en el estudio de la usabilidad de mecanismos no-visuales para la transmisión de información contextual del código. Como trabajo futuro se propone la replicación del estudio, buscando aumentar el número de participantes, por ejemplo, siguiendo una estrategia de muestreo en bola de nieve para así cubrir un mayor número de casos. De esta manera, se podría aumentar la validez externa de los resultados obtenidos.

La extensión implementada permite expandir los tipos de NVCs a generar. Por ejemplo, se podría agregar la retroalimentación a través de *spearcons* (discurso hablado que se comprime en duración para similar un *earcon*). No se consideró este tipo de retroalimentación en el caso de estudio realizado, ya que este mecanismo se ha utilizado en la literatura, con resultados sub-optimales para la exploración de código. Como trabajo futuro, se podría replicar el caso de estudio incorporando este tipo de retroalimentación y reforzar experimentalmente los resultados obtenidos en investigaciones previas.

Las VCs estudiadas fueron en particular las asociadas a las palabras reservadas del lenguaje de programación; sin embargo, existen otros tipos de VCs. Por ejemplo, en Python, las asociadas a los elementos literales en la composición de *strings*. Se propone entonces, cómo trabajo futuro, estudiar el desempeño de los mismos tipos de NVCs, pero asociadas a nuevas VCs de origen. Esto permitirá explorar si existe una dependencia entre la usabilidad de las NVCs utilizadas según el tipo de VC originario.

En la presente tesis, se abordó el problema de la transmisión de información contextual del código desde el punto de vista del IDE. Sin embargo, este problema también puede ser abordado desde el punto de vista del lector de pantalla. Esto permitiría el acceso a información y eventos adicionales como, por ejemplo, la velocidad de lectura o la última palabra leída. Se propone entonces el desarrollo de una implementación alternativa, que dote al lector de pantalla de las funcionalidades de VisualCueTransform. De esta manera, se podría explorar el efecto de la velocidad de lectura configurable sobre la usabilidad de NVCs basadas en texto, como el texto hablado y *spearcons*.

Finalmente, la definición del problema implicaba que la solución debía corresponder a una única NVC que transmitiera la información contextual del código. Sin embargo, es factible desencadenar múltiples señales del código en simultáneo. Por ejemplo, utilizar un *earcon* y una vibración al identificar una palabra clave. Esto podría mejorar la usabilidad de los procesos de escritura y lectura de código, no tan solo en programadores ciegos, sino que también en programadores videntes. Esto último podría darse complementando VCs con NVCs. Se propone como trabajo futuro el estudio de la optimalidad de la transmisión de información contextual del código a través de estímulos combinados.

# Bibliografía

- [1] ACM. Acm computing classification system, 2021.
- [2] ALBUSAYS, K. Exploring auditory cues to locate code errors and enhance navigation for developers who are blind. *ACM SIGACCESS Accessibility and Computing*, 120 (Jan. 2018), 11–15.
- [3] ALBUSAYS, K., AND LUDI, S. Eliciting programming challenges faced by developers with visual impairments. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering* (May 2016), ACM.
- [4] ALBUSAYS, K., LUDI, S., AND HUENERFAUTH, M. Interviews and observation of blind software developers at work to understand code navigation challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (Oct. 2017), ACM.
- [5] ARMALY, A., RODEGHERO, P., AND McMILLAN, C. AudioHighlight: Code skimming for blind programmers. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (Sept. 2018), IEEE.
- [6] ARMALY, A., RODEGHERO, P., AND McMILLAN, C. A comparison of program comprehension strategies by blind and sighted programmers. In *Proceedings of the 40th International Conference on Software Engineering* (May 2018), ACM.
- [7] BAKER, C. M., MILNE, L. R., AND LADNER, R. E. StructJumper. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Apr. 2015), ACM.
- [8] BROOKE, J. *Usability Evaluation in Industry*. Taylor & Francis, 01 1996, ch. SUS – a quick and dirty usability scale, pp. 189–194.
- [9] CARBONNELLE, P. Pypl popularity of programming language index, 2021.
- [10] CARBONNELLE, P. Top integrated development environment index, 2021.
- [11] CODE, V. S. Extension api, 2021.
- [12] CODE, V. S. Syntax highlight guide, 2021.
- [13] DE CHILE, G. Actualización plan “paso a paso”, 2021.
- [14] FALASE, O., SIU, A. F., AND FOLLMER, S. Tactile code skimmer. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility* (Oct. 2019), ACM.
- [15] GOOGLE. Acerca de - google maps, 2021.
- [16] GUERREIRO, J., AND GONÇALVES, D. Text-to-speeches. In *Proceedings of the 16th*

*international ACM SIGACCESS conference on Computers & accessibility - ASSETS '14* (2014), ACM Press.

- [17] HADWEN-BENNETT, A., SENTANCE, S., AND MORRISON, C. Making programming accessible to learners with visual impairments: A literature review. *International Journal of Computer Science Education in Schools* 2, 2 (May 2018), 3–13.
- [18] HUTCHINSON, J., AND METATLA, O. An initial investigation into non-visual code structure overview through speech, non-speech and spearcons. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems* (Apr. 2018), ACM.
- [19] ISO. Ergonomics of human-system interaction — part 11: Usability: Definitions and concepts, 2018.
- [20] KAYE, H. S., JANS, L. H., AND JONES, E. C. Why don't employers hire and retain workers with disabilities? *Journal of Occupational Rehabilitation* 21, 4 (Mar. 2011), 526–536.
- [21] LAZARILLO. Lazarillo app - asistente inteligente de orientación e información, 2021.
- [22] LUDI, S., SIMPSON, J., AND MERCHANT, W. Exploration of the use of auditory cues in code comprehension and navigation for individuals with visual impairments in a visual programming environment. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility* (Oct. 2016), ACM.
- [23] MONARES, A., OCHOA, S., PINO, J., LOPEZ, T. R., AND NOGUERA, M. Using unconventional awareness in emergency responses. *IEEE Latin America Transactions* 12, 1 (Jan. 2014), 62–68.
- [24] NORMAN, D. A. *Memory and attention: An introduction to human information processing*. Wiley, 1980.
- [25] PANDEY, M., KAMESWARAN, V., RAO, H. V., O'MODHRAN, S., AND ONEY, S. Understanding accessibility and collaboration in programming for people with visual impairments. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (Apr. 2021), 1–30.
- [26] SCHMANDT, C., AND MULLINS, A. AudioStreamer. In *Conference companion on Human factors in computing systems - CHI '95* (1995), ACM Press.
- [27] SENADIS. II estudio nacional de la discapacidad. un nuevo enfoque para la inclusión, 2015.
- [28] WALKER, B. N., LINDSAY, J., NANCE, A., NAKANO, Y., PALLADINO, D. K., DINGLER, T., AND JEON, M. Spearcons (speech-based earcons) improve navigation performance in advanced auditory menus. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 55, 1 (July 2012), 157–182.
- [29] YIN, R. K. *Case study research : design and methods*. SAGE Publications, 2003.
- [30] ZAYOUR, I., AND HAJJDIAB, H. How much integrated development environments (IDEs) improve productivity? *Journal of Software* 8, 10 (Oct. 2013).
- [31] ZELADA, E. Ajuste dinámico de dificultad en videojuegos mediante biofeedback. <http://repositorio.uchile.cl/handle/2250/173728>, 2019. Undergraduate thesis.

# Anexos

## Anexo A

### Prueba pareada

#### A.1. Instrumentos

##### A.1.1. Transcripción formulario

A continuación se presenta el contenido del formulario, en su formato A, utilizado por el grupo A, transcrito textualmente.

Sección 1: Experimento. En este formulario encontrarás instrucciones para realizar el experimento. Debes tener instalado Visual Studio Code. Se solicita dirección de correo electrónico.

Sección 2: Preparación. Abre Visual Studio Code. Instala la extensión Plain Theme. Para esto navega a Files > Preferences > Color Theme, escribe “plain theme” y da Enter. En el menú que aparecerá a la izquierda selecciona la segunda opción “Plain Theme 0.1.0” y da click en el botón “Install”. Se desplegará un menú de opciones. Selecciona “Plain (Dark)”. Usaremos este tema hasta la parte 4. Descarga desde Material Docente de U-Cursos el archivo “Codigo Experimento”, y descomprimelo en un directorio de fácil acceso (Ej: Documentos). En Visual Studio Code abre la carpeta “Codigo Experimento”. Para esto navega a Files > Open Folder, y busca y selecciona la carpeta descomprimida, luego selecciona OK. En este formulario encontrarás instrucciones para realizar el experimento. Se solicita dirección de correo electrónico. Vista de la aplicación una vez abierta la carpeta. Captura de pantalla de la vista.

Sección 3: Parte 0. A continuación deberás responder algunas preguntas preliminares sobre tu nivel de habilidad y conocimiento. Es fundamental que seas honesto/a. ¿Tienes alguna de las siguientes condiciones visuales? con opciones múltiples Daltonismo o Acromatopsia; Miopía, Hipermetropía o Astigmatismo; Ceguera parcial o total; Otra (permite especificar) ¿Cuál es tu nivel de experiencia programando? en escala de 1 a 5 en que 1 es Nunca he Programado y 5 es Programo regularmente. Visual Studio Code es un ambiente de desarrollo integrado ¿Habías usado Visual Studio Code antes? con opciones Sí y No. ¿Has usado algún otro ambiente de desarrollo integrado? con opciones Sí y No. ¿Cómo describirías tu nivel de habilidad con Python? en escala de 1 a 5 en que 1 es No soy hábil programando en Python y 5 es Soy excelente desarrollando en Python.



Sección 4: Parte 1. A continuación se te presentará una serie de preguntas con respecto al archivo “parte\_1.py”. Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code. Toma un tiempo para leer el código. En el lenguaje Python la palabra “return” se considera una palabra clave. ¿Cuántas veces se utiliza la palabra “return”? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python todo lo que viene después del símbolo numeral (#), en una misma línea de código, se considera un comentario. ¿Cuántas líneas de código contienen comentarios? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python lo que se encuentra entre comillas (“”) se considera un String (cadena de texto). ¿Cuántos Strings identificas? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. ¿Qué estrategia usaste para resolver las preguntas de esta parte? recibe una respuesta de texto largo.

Sección 5: Parte 2. A continuación se te presentará una serie de preguntas con respecto al archivo “parte\_2.py”. Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code. Este archivo es similar en contenido al archivo “parte\_1.py”, pero se han realizado algunas modificaciones. En el lenguaje Python todo lo que viene después del símbolo numeral (#), en una misma línea de código, se considera un comentario. ¿Cuántas líneas de código contienen comentarios? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python la palabra “return” se considera una palabra clave. ¿Cuántas veces se utiliza la palabra return fuera de un comentario? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python lo que se encuentra entre comillas (“”) se considera un String (cadena de texto). ¿Cuántos Strings identificas fuera de un comentario? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. ¿Qué estrategia usaste para resolver las preguntas de esta parte? recibe una respuesta de texto largo.

Sección 6: Parte 3. A continuación se te presentará una serie de preguntas con respecto al archivo “parte\_3.py”. Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code. En Python, algunas palabras clave pueden ser usadas como valores. En este ejercicio se debe seguir la lógica presentada en el archivo, y reemplazar el comentario #value por la palabra clave apropiada. Una vez hayas cambiado el archivo, copia y pega la línea 13 como respuesta a esta pregunta. Recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. Describe cómo resolviste el problema. Recibe una respuesta de texto largo.

Sección 7: Parte 4 - Cambiamos de tema. A continuación se te presentará una serie de preguntas con respecto al archivo “parte\_4.py”. Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code. Además, ahora debes cambiar el tema de Visual Studio Code. Para esto navega a Files > Preferences > Color Theme. Selecciona “Dark+ (default dark)”. Usaremos este tema para todas las partes siguientes. Toma un tiempo para leer el código. En el



lenguaje Python la palabra “return” se considera una palabra clave. ¿Cuántas veces se utiliza la palabra “return” fuera de un comentario? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python todo lo que viene después del símbolo numeral (#), en una misma línea de código, se considera un comentario. ¿Cuántas líneas de código contienen comentarios? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python lo que se encuentra entre comillas (“”) se considera un String (cadena de texto). ¿Cuántos Strings identificas? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. ¿Qué estrategia usaste para resolver las preguntas de esta parte? recibe una respuesta de texto largo.

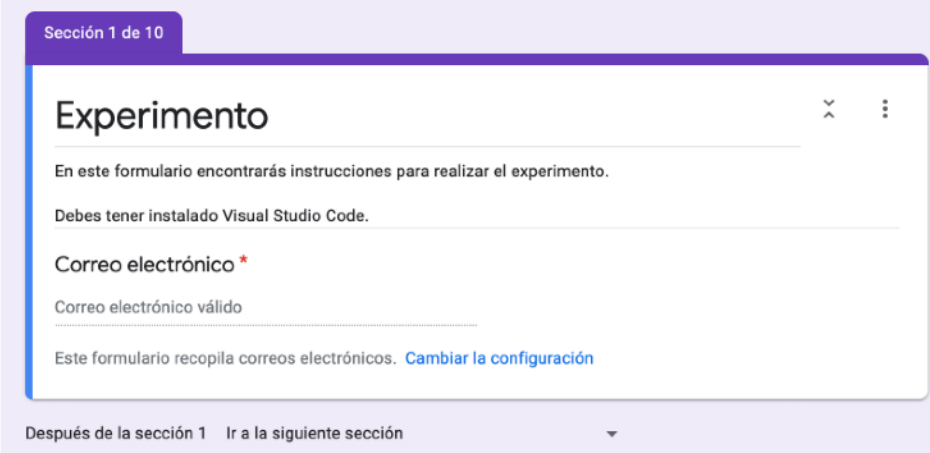
Sección 8: Parte 5. A continuación se te presentará una serie de preguntas con respecto al archivo “parte\_5.py”. Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code. Este archivo es similar en contenido al archivo “parte\_4.py” pero se han realizado algunas modificaciones. En el lenguaje Python todo lo que viene después del símbolo numeral (#), en una misma línea de código, se considera un comentario. ¿Cuántas líneas de código contienen comentarios? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python la palabra “return” se considera una palabra clave. ¿Cuántas veces se utiliza la palabra return fuera de un comentario? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. En el lenguaje Python lo que se encuentra entre comillas (“”) se considera un String (cadena de texto). ¿Cuántos Strings identificas fuera de un comentario? recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. ¿Qué estrategia usaste para resolver las preguntas de esta parte? recibe una respuesta de texto largo.

Sección 9: Parte 6. A continuación se te presentará una serie de preguntas con respecto al archivo “parte\_6.py”. Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code. Una de las líneas de código del archivo contiene un error. Debes cambiar esa línea por una correcta. Una vez hayas cambiado el archivo, copia y pega la línea cambiada como respuesta a esta pregunta. Recibe una respuesta de texto breve. ¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? en escala de 1 a 5 en que 1 es Nada de esfuerzo y 5 es Mucho esfuerzo. Describe cómo resolviste el problema. Recibe una respuesta de texto largo.

Sección 10: Parte 7. Retrospectiva. Con respecto a los ejercicios realizados, manifiesta que tan de acuerdo estás con las siguientes afirmaciones. Resaltar el código utilizando colores ayuda a explorarlo. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda a entenderlo. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda a encontrar lo que se busca. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda a identificar palabras clave del lenguaje. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda a encontrar errores en el código. En escala de 1 a 5 en que 1 es

Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda en la escritura de nuevo código. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda a corregir errores. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda a evitar errores. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores ayuda a familiarizarse con el lenguaje. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores resulta cómodo. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores resulta práctico. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. Resaltar el código utilizando colores mejora la experiencia de usuario. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo. ¿Cómo cambió tu estrategia para resolver este experimento al cambiar el resaltado del código?. En escala de 1 a 5 en que 1 es Muy en desacuerdo y 5 es Muy de acuerdo.

### A.1.2. Capturas formulario



The screenshot shows a web form titled "Experimento" within a purple-themed interface. At the top left, it says "Sección 1 de 10". The form content includes: a title "Experimento" with a close and menu icon; a paragraph "En este formulario encontrarás instrucciones para realizar el experimento."; a requirement "Debes tener instalado Visual Studio Code."; a required field "Correo electrónico\*" with a placeholder "Correo electrónico válido"; and a footer note "Este formulario recopila correos electrónicos. [Cambiar la configuración](#)". At the bottom, there is a navigation bar with "Después de la sección 1" and "Ir a la siguiente sección" with a dropdown arrow.

Figura A.1: Captura de pantalla sección 1 formato A.

## Preparación

Abre Visual Studio Code.

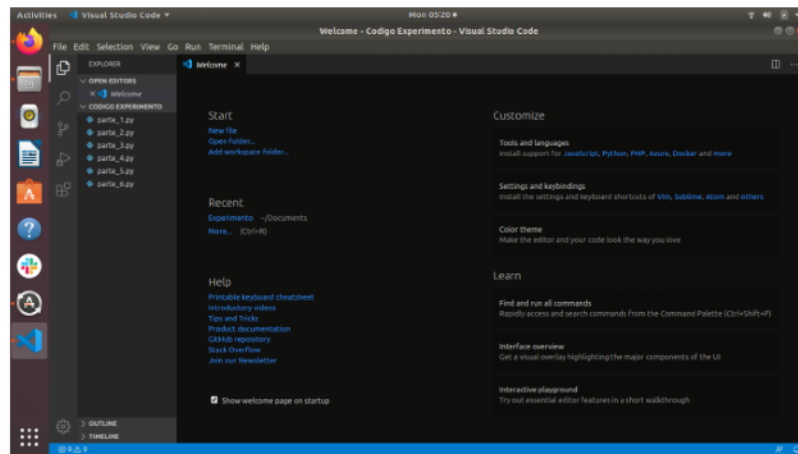
Instala la extensión Plain Theme. Para esto navega a Files > Preferences > Color Theme, escribe "plain theme" y da Enter. En el menú que aparecerá a la izquierda selecciona la segunda opción "Plain Theme 0.1.0" y da click en el botón "Install". Se desplegará un menú de opciones. Selecciona "Plain (Dark)".

Usaremos este tema hasta la parte 4.

Descarga desde Material Docente de U-Cursos el archivo "Codigo Experimento", y descomprimelo en un directorio de fácil acceso (Ej: Documentos)

En Visual Studio Code abre la carpeta "Codigo Experimento". Para esto navega a Files > Open Folder, y busca y selecciona la carpeta descomprimida, luego selecciona OK.

Vista de la aplicación una vez abierta la carpeta.



Después de la sección 2 Ir a la siguiente sección

Figura A.2: Captura de pantalla sección 2 formato A.

## Parte 0

A continuación deberás responder algunas preguntas preliminares sobre tu nivel de habilidad y conocimiento. Es fundamental que seas honesto/a.

¿Tienes alguna de las siguientes condiciones visuales?

- Daltonismo o Acromatopsia
- Miopía, Hipermetropía o Astigmatismo
- Ceguera parcial o total
- Otra...

¿Cuál es tu nivel de experiencia programando? \*

- Nunca he programado    1    2    3    4    5    Programo regularmente
- 

Visual Studio Code es un ambiente de desarrollo integrado ¿Habías usado Visual Studio Code antes? \*

- Sí
- No

¿Has usado algún otro ambiente de desarrollo integrado? \*

- Sí
- No

¿Conoces el lenguaje de programación Python? \*

- Sí
- No

¿Cómo describirías tu nivel de habilidad con Python? \*

- No soy habil programando en Python    1    2    3    4    5    Soy excelente desarrollando en Python
- 

Figura A.3: Captura de pantalla sección 3 formato A.

## Parte 1

A continuación se te presentará una serie de preguntas con respecto al archivo "parte\_1.py". Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code.

Toma un tiempo para leer el código

En el lenguaje Python la palabra "return" se considera una palabra clave. ¿Cuántas veces se utiliza la palabra "return"? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

Nada de esfuerzo    1    2    3    4    5    Mucho esfuerzo

En el lenguaje Python todo lo que viene después del símbolo numeral (#), en una misma línea de código, se considera un comentario. ¿Cuántas líneas de código contienen comentarios? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

Nada de esfuerzo    1    2    3    4    5    Mucho esfuerzo

En el lenguaje Python lo que se encuentra entre comillas (") se considera un String (cadena de texto). ¿Cuántos Strings identificas? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

Nada de esfuerzo    1    2    3    4    5    Mucho esfuerzo

¿Qué estrategia usaste para resolver las preguntas de esta parte? \*

Texto de respuesta largo

## Parte 2

A continuación se te presentará una serie de preguntas con respecto al archivo "parte\_2.py". Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code.

Este archivo es similar en contenido al archivo "parte\_1.py", pero se han realizado algunas modificaciones.

En el lenguaje Python todo lo que viene después del símbolo numeral (#), en una misma línea de código, se considera un comentario. ¿Cuántas líneas de código contienen comentarios? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

Nada de esfuerzo    1    2    3    4    5    Mucho esfuerzo

En el lenguaje Python la palabra "return" se considera una palabra clave. ¿Cuántas veces se utiliza la palabra return fuera de un comentario? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

Nada de esfuerzo    1    2    3    4    5    Mucho esfuerzo

En el lenguaje Python lo que se encuentra entre comillas ("") se considera un String (cadena de texto). ¿Cuántos Strings identificas fuera de un comentario? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

Nada de esfuerzo    1    2    3    4    5    Mucho esfuerzo

¿Qué estrategia usaste para resolver las preguntas de esta parte? \*

Texto de respuesta largo

## Parte 3

A continuación se te presentará una serie de preguntas con respecto al archivo "parte\_3.py". Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code.

En Python, algunas palabras clave pueden ser usadas como valores. En este ejercicio se debe seguir la lógica presentada en el archivo, y reemplazar el comentario #value por la palabra clave apropiada. \*

Una vez hayas cambiado el archivo, copia y pega la línea 13 como respuesta a esta pregunta

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

|                  |                       |                       |                       |                       |                       |                |
|------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                  | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Nada de esfuerzo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Mucho esfuerzo |

Describe cómo resolviste el problema \*

Texto de respuesta largo

Después de la sección 6 Ir a la siguiente sección

Figura A.6: Captura de pantalla sección 6 formato A.

## Parte 4 - Cambiamos de tema

A continuación se te presentará una serie de preguntas con respecto al archivo "parte\_4.py". Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code.

Además, ahora debes cambiar el tema de Visual Studio Code. Para esto navega a Files > Preferences > Color Theme. Selecciona "Dark+ (default dark)".

Usaremos este tema para todas las partes siguientes.

Toma un tiempo para leer el código

En el lenguaje Python la palabra "return" se considera una palabra clave. ¿Cuántas veces se utiliza la palabra "return" fuera de un comentario? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

1      2      3      4      5

Nada de esfuerzo                        Mucho esfuerzo

En el lenguaje Python todo lo que viene después del símbolo numeral (#), en una misma línea de código, se considera un comentario. ¿Cuántas líneas de código contienen comentarios? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

1      2      3      4      5

Nada de esfuerzo                        Mucho esfuerzo

En el lenguaje Python lo que se encuentra entre comillas (") se considera un String (cadena de texto). ¿Cuántos Strings identificas? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

1      2      3      4      5

Nada de esfuerzo                        Mucho esfuerzo

¿Qué estrategia usaste para resolver las preguntas de esta parte? \*

Texto de respuesta largo



## Parte 5

A continuación se te presentará una serie de preguntas con respecto al archivo "parte\_5.py". Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code.

Este archivo es similar en contenido al archivo "parte\_4.py" pero se han realizado algunas modificaciones.

En el lenguaje Python todo lo que viene despues del simbolo numeral (#), en una misma linea de código, se considera un comentario. ¿Cuántas lineas de código contienen comentarios? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

|                  |                       |                       |                       |                       |                       |                |
|------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                  | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Nada de esfuerzo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Mucho esfuerzo |

En el lenguaje Python la palabra "return" se considera una palabra clave. ¿Cuántas veces se utiliza la palabra return fuera de un comentario? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

|                  |                       |                       |                       |                       |                       |                |
|------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                  | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Nada de esfuerzo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Mucho esfuerzo |

En el lenguaje Python lo que se encuentra entre comillas (") se considera un String (cadena de texto). ¿Cuántos Strings identificas fuera de un comentario? \*

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

|                  |                       |                       |                       |                       |                       |                |
|------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                  | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Nada de esfuerzo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Mucho esfuerzo |

¿Qué estrategia usaste para resolver las preguntas de esta parte? \*

Texto de respuesta largo

## Parte 6



A continuación se te presentará una serie de preguntas con respecto al archivo "parte\_6.py". Para abrirlo, selecciónalo en el panel izquierdo de Visual Studio Code.

Una de las líneas de código del archivo contiene un error. Debes cambiar esa línea por una correcta. \*

Una vez hayas cambiado el archivo, copia y pega la línea cambiada como respuesta a esta pregunta

Texto de respuesta breve

¿Cuánto esfuerzo es requerido para contestar la pregunta anterior? \*

|                  |                       |                       |                       |                       |                       |                |
|------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                  | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Nada de esfuerzo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Mucho esfuerzo |

Describe cómo resolviste el problema \*

Texto de respuesta largo

Después de la sección 9 Ir a la siguiente sección



Figura A.9: Captura de pantalla sección 9 formato A.

## Parte 7



Retrospectiva.

Con respecto a los ejercicios realizados, manifiesta que tan de acuerdo estás con las siguientes afirmaciones.

Resaltar el código utilizando colores ayuda a explorarlo \*

|                   |                       |                       |                       |                       |                       |                |
|-------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                   | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Muy en desacuerdo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Muy de acuerdo |

Resaltar el código utilizando colores ayuda a entenderlo \*

|                   |                       |                       |                       |                       |                       |                |
|-------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                   | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Muy en desacuerdo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Muy de acuerdo |

Resaltar el código utilizando colores ayuda a encontrar lo que se busca \*

|                   |                       |                       |                       |                       |                       |                |
|-------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                   | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Muy en desacuerdo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Muy de acuerdo |

Resaltar el código utilizando colores ayuda a identificar palabras clave del lenguaje \*

|                   |                       |                       |                       |                       |                       |                |
|-------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                   | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Muy en desacuerdo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Muy de acuerdo |

Resaltar el código utilizando colores ayuda a encontrar errores en el código \*

|                   |                       |                       |                       |                       |                       |                |
|-------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                   | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Muy en desacuerdo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Muy de acuerdo |

Resaltar el código utilizando colores ayuda en la escritura de nuevo código \*

|                   |                       |                       |                       |                       |                       |                |
|-------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------|
|                   | 1                     | 2                     | 3                     | 4                     | 5                     |                |
| Muy en desacuerdo | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Muy de acuerdo |

Figura A.10: Captura de pantalla sección 10 formato A, primera parte.

Resaltar el código utilizando colores ayuda a corregir errores \*

1 2 3 4 5

Muy en desacuerdo      Muy de acuerdo

---

Resaltar el código utilizando colores ayuda a evitar errores \*

1 2 3 4 5

Muy en desacuerdo      Muy de acuerdo

---

Resaltar el código utilizando colores ayuda a familiarizarse con el lenguaje \*

1 2 3 4 5

Muy en desacuerdo      Muy de acuerdo

---

Resaltar el código utilizando colores resulta cómodo \*

1 2 3 4 5

Muy en desacuerdo      Muy de acuerdo

---

Resaltar el código utilizando colores resulta práctico \*

1 2 3 4 5

Muy en desacuerdo      Muy de acuerdo

---

Resaltar el código utilizando colores mejora la experiencia de usuario \*

1 2 3 4 5

Muy en desacuerdo      Muy de acuerdo

---

¿Cómo cambió tu estrategia para resolver este experimento al cambiar el resaltado del código? \*

Texto de respuesta largo

---

Figura A.11: Captura de pantalla sección 10 formato A, segunda parte.

# Anexo B

## Caso de estudio

### B.1. Instrumentos

## B.1.1. Cuestionario de autoreporte

### CUESTIONARIO DE AUTOREPORTE

A continuación podrá encontrar un cuestionario de autoreporte, dirigido a personas ciegas, con preguntas que permitirán determinar si es usted un/a candidato/a apropiado/a para la participación en un experimento sobre la experiencia al desarrollar en programadores ciegos.

**Edad:** \_\_\_\_\_

**Género:** \_\_\_\_\_

#### Nivel educacional

- a. Básica incompleta
- b. Básica completa
- c. Media incompleta
- d. Media completa
- e. Superior incompleta
- f. Superior completa
- g. Otro, especifique \_\_\_\_\_

**¿Cuál es su nivel de experiencia programando? Indique, del 1 al 5, en que 1 corresponde a no tener experiencia programando, y 5 corresponde a ser muy experimentado.**

\_\_\_\_\_

**¿Cómo evaluaría su habilidad para el desarrollo en Python? Indique, del 1 al 5, en que 1 corresponde a no tener experiencia programando, y 5 corresponde a ser muy experimentado.**

\_\_\_\_\_

**Visual Studio Code es un ambiente de desarrollo integrado ¿Has usado Visual Studio Code?**

- a. Sí
- b. No

**NVDA (Non Visual Desktop Access) es un lector de pantalla que permite a personas ciegas usar computadores ¿Has utilizado NVDA?**

- a. Sí
- b. No

## B.1.2. Transcripción protocolo experimental

[Buenos días/Buenas tardes], mi nombre es Roberto Puga y seré el encargado de aplicar el experimento.

Antes de comenzar, para esta experiencia es necesario grabar el audio de esta sala, ¿está de acuerdo con esto?

(Pausa en espera de consentimiento)

Contextualizando, el objetivo de este experimento es medir qué tan eficaz resulta una extensión de Visual Studio Code en la transformación de señales visuales en señales no-visuales. Los resultados de este experimento serán anónimos.

Se le ha provisto de un computador portátil con sistema operativo Windows, y el programa lector de pantallas NVDA. El computador graba la pantalla para análisis de interacciones. Además, cuenta con el ambiente de desarrollo integrado Visual Studio Code.

En el siguiente experimento se le pide que resuelva una serie de 12 desafíos de programación. Para esto usted deberá leer el código que se le provee, y, en algunos casos, también escribir. Además, según sea el caso, deberá utilizar la extensión de Visual Studio Code que le permitirá acceder, de distintas formas, a la metainformación del código que esté leyendo. No es necesario escribir programas que resuelvan los desafíos, solo explorar los archivos provistos.

Durante cada uno de estos desafíos se medirá el tiempo utilizado para resolverlos, sin embargo, el foco no es resolverlos en el menor tiempo posible, sino que llegar a una respuesta con la que se sienta seguro y satisfecho. Se iniciará un cronómetro cuando usted indique que está listo/a para iniciar el siguiente desafío y se detendrá cuando usted indique haberlo resuelto.

Es ideal que no se hagan preguntas al aplicador del experimento durante su realización, a menos que estas sean indispensables para su continuidad. Puede solicitar, en cualquier momento, que se repitan las instrucciones entregadas.

El computador se encuentra abierto y encendido. Hay una ventana de Visual Studio Code abierta y el ambiente de trabajo se encuentra en el directorio apropiado para ejecutar los desafíos. Se encuentra abierto el archivo “prueba.py”. Si así lo requiere, puede tomarse unos minutos para familiarizarse tanto con el computador, como con la configuración de NVDA utilizada.

(Pausa para familiarización con los dispositivos)

En el lenguaje de programación Python existen palabras claves, estas son cadenas de caracteres que están reservadas y que se interpretan de una forma particular. Estas pueden servir para representar operadores lógicos, para el control de flujo, para el almacenamiento de información, o ser constantes. Que el lenguaje de programación interprete una palabra

como “clave” dependerá del contexto en que se encuentre.

**Para el primer desafío**, usted deberá abrir y leer el archivo “parte\_1.py”, e identificar cuantas veces se utiliza la palabras “return” como palabras clave. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente ¿comenzamos?

(Pausa para resolver el desafío 1)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el segundo desafío**, usted deberá abrir y leer el archivo “parte\_2.py”, este es similar al archivo anterior, pero se han introducido cambios. Usted debe identificar cuantas veces se utiliza la palabra “return” como palabra clave. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente, una vez lo haga se detendrá el cronómetro. ¿comenzamos?

(Pausa para resolver el desafío 2)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el tercer desafío**, usted deberá abrir y leer el archivo “parte\_3.py”, encontrar el comentario “#completar” y reemplazarlo por una palabra clave apropiada siguiendo la estructura del archivo. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe escribir directamente en el archivo. ¿Comenzamos?

(Pausa para resolver el desafío 3)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

Para los siguientes 3 desafíos, debe utilizar la extensión configurada para identificar palabras clave del lenguaje, y retroalimentar con sonidos. Para activar este modo de funcionamiento, presione en el teclado la combinación Ctrl+Alt+7. Esta extensión le permitirá, al escribir o seleccionar una palabra, determinar si es clave en el lenguaje de programación Python.

**Para el cuarto desafío**, usted deberá leer, en la pantalla del computador, el contenido del archivo “parte\_4.py”, e identificar cuantas veces se utiliza la palabras “return”, pero no



como palabras clave, por ejemplo, dentro de un comentario o de un String. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente. ¿Comenzamos?

(Pausa para resolver el desafío 4)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el quinto desafío**, usted deberá abrir y leer el archivo “parte\_5.py”. Este es similar al archivo anterior, pero se han introducido cambios. Usted debe identificar cuantas veces se utiliza la palabras “return”, pero no como palabras clave, por ejemplo, dentro de un comentario o de un String. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente. ¿Comenzamos?

(Pausa para resolver el desafío 5)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como y “poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el sexto desafío**, usted deberá abrir y leer el archivo “parte\_6.py”. Este cuenta con un error de syntaxis. Usted debe identificar el número de línea en que se encuentra el error, y corregirlo. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. ¿Comenzamos?

(Pausa para resolver el desafío 6)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

Para los siguientes 3 desafíos, debe cambiar la configuración de la extensión, para esto, presione en el teclado la combinación Ctrl+ALT+8. Esta vez, se emitirá un sonido inerte, en vez de texto hablado, tanto al seleccionar como al escribir una palabra clave.

**Para el séptimo desafío**, usted deberá abrir y leer el archivo “parte\_7.py”, e identificar cuantas veces se utilizan la palabra None como palabra claves. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente. ¿Comenzamos?

(Pausa para resolver el desafío 7)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío?

Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el octavo desafío**, usted deberá abrir y leer el archivo “parte\_8.py”. Este es similar al archivo anterior, pero se han introducido cambios. Usted debe identificar cuantas veces se utiliza la palabra None como palabra clave. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente. ¿Comenzamos?

(Pausa para resolver el desafío 8)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el noveno desafío**, usted deberá abrir y leer el archivo “parte\_9.py”. Este cuenta con un error de syntaxis. Usted debe identificar el número de línea en que se encuentra el error, y corregirlo. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. ¿Comenzamos?

(Pausa para resolver el desafío 9)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

Para los 3 desafíos finales, debe poner en su muñeca no dominante la pulsera inteligente provista, luego, cambiar la configuración de la extensión, para esto, presione en el teclado la combinación Ctrl+Alt+9. Esta vez, tanto al seleccionar como al escribir una palabra clave, la pulsera vibrará.

**Para el décimo desafío**, usted deberá abrir y leer el archivo “parte\_10.py”, e identificar cuantas veces se utilizan las palabras for y while como palabras claves. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente. ¿Comenzamos?

(Pausa para resolver el desafío 10)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el undécimo desafío**, usted deberá abrir y leer el archivo “parte\_11.py”. Este es similar al archivo anterior, pero se han introducido cambios. Usted debe identificar cuantas

veces se utilizan las palabras `for` y `while` como palabras claves. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe entregar verbalmente. ¿Comenzamos?

(Pausa para resolver el desafío 11)

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

**Para el último desafío**, usted deberá leer, en la pantalla del computador, el contenido del archivo “`parte_12.py`”, encontrar el comentario “`#completar`” y reemplazarlo por una palabra clave apropiada siguiendo la estructura del archivo. Cuando esté listo/a para iniciar, comenzará a correr el cronómetro. El resultado lo debe escribir directamente en el archivo. ¿Comenzamos?

(Pausa para resolver el desafío 12) 9 00:01:24

En una escala del 1 al 5 ¿Cómo calificaría el esfuerzo que ha demandado este desafío? Considere 1 como “muy poco esfuerzo”, y 5 como “mucho esfuerzo”.

(Pausa en espera de evaluación de esfuerzo)

A continuación, usted debe evaluar, del 1 al 5, siendo 1 el valor más bajo y 5 el valor más alto, su satisfacción al utilizar cada una de las opciones de la extensión.

Del 1 al 5, ¿qué tan satisfecho se sintió al resolver desafíos con la extensión apagada?

Del 1 al 5, ¿qué tan satisfecho se sintió al resolver desafíos con la extensión encendida utilizando sonido inerte?

Del 1 al 5, ¿qué tan satisfecho se sintió al resolver desafíos con la extensión encendida utilizando texto hablado?

Del 1 al 5, ¿qué tan satisfecho se sintió al resolver desafíos con la extensión encendida utilizando vibraciones?

Con respecto al sistema de retroalimentación con sonidos inertes, indique del 1 al 5, en que 1 es “muy en desacuerdo” y 5 es “muy de acuerdo”, su apreciación sobre las siguientes afirmaciones:

- Creo que usaría el sistema frecuentemente
- Encuentro que el sistema es innecesariamente complejo
- Pensé que el sistema era fácil de usar
- Pienso que necesitaría ayuda de otra persona para poder utilizar el sistema

- Encontré que las funcionalidades del sistema están bien integradas
- Pensé que habían muchas inconsistencias en el sistema
- Imaginaría que la mayoría de las personas aprenderían a usar el sistema muy rápidamente
- Encontré el sistema muy engorroso de usar
- Me sentí muy seguro utilizando el sistema
- Necesité aprender muchas cosas antes de poder utilizar el sistema

Con respecto al sistema de retroalimentación con discurso hablado, indique del 1 al 5, en que 1 es “muy en desacuerdo” y 5 es “muy de acuerdo”, su apreciación sobre las siguientes afirmaciones:

- Creo que usaría el sistema frecuentemente
- Encuentro que el sistema es innecesariamente complejo
- Pensé que el sistema era fácil de usar
- Pienso que necesitaría ayuda de otra persona para poder utilizar el sistema
- Encontré que las funcionalidades del sistema están bien integradas
- Pensé que habían muchas inconsistencias en el sistema
- Imaginaría que la mayoría de las personas aprenderían a usar el sistema muy rápidamente
- Encontré el sistema muy engorroso de usar
- Me sentí muy seguro utilizando el sistema
- Necesité aprender muchas cosas antes de poder utilizar el sistema

Con respecto al sistema de retroalimentación con vibraciones, indique del 1 al 5, en que 1 es “muy en desacuerdo” y 5 es “muy de acuerdo”, su apreciación sobre las siguientes afirmaciones:

- Creo que usaría el sistema frecuentemente
- Encuentro que el sistema es innecesariamente complejo
- Pensé que el sistema era fácil de usar
- Pienso que necesitaría ayuda de otra persona para poder utilizar el sistema
- Encontré que las funcionalidades del sistema están bien integradas
- Pensé que habían muchas inconsistencias en el sistema

- Imaginaría que la mayoría de las personas aprenderían a usar el sistema muy rápidamente
- Encontré el sistema muy engorroso de usar
- Me sentí muy seguro utilizando el sistema
- Necesité aprender muchas cosas antes de poder utilizar el sistema

Según su percepción, cómo ordenaría las configuraciones de la extensión, de la más a la menos útil (desactivada, sonidos, texto hablado, vibración)

Hemos terminado la aplicación del experimento, muchas gracias por su participación. ¿Tiene algún comentario adicional con respecto al mismo?

## **B.2. Certificación**

## B.2.1. Certificación comité de ética FCFM



### CERTIFICACION N°002

#### CERTIFICACION COMITÉ DE ÉTICA Y BIOSEGURIDAD PARA LA INVESTIGACIÓN FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS UNIVERSIDAD DE CHILE

El Comité de Ética y Bioseguridad para la Investigación de la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile certifica haber analizado el Proyecto de Investigación, titulado “***Transformación de señales visuales de código fuente en entornos de desarrollo integrado para personas ciegas***”, del estudiante de magister Roberto Puga Franco y cuyo profesor guía es el académico Francisco Gutiérrez, del Departamento de Ciencias de la Computación.

La metodología del proyecto incluye la participación de dos personas no videntes. Se grabará tanto la pantalla del computador utilizado durante la experiencia, la voz con instrucciones entregadas y las preguntas realizadas por el facilitador, así como las respuestas entregadas por el participante durante su ejecución. Los datos serán almacenados en computadores de trabajo del equipo de investigación, y respaldado en el Departamento de Ciencias de la Computación.

Los participantes entregarán su consentimiento informado, para lo cual, el facilitador del experimento leerá en voz alta el formulario de consentimiento y pedirá a viva voz a los participantes la aceptación del mismo y se solicitará la firma de los documentos y se dejará un registro en formato audio de esta actividad.

El análisis del proyecto y en particular la metodología permite certificar que:

- i) El proyecto cumple con los estándares nacionales e internacionales de ética de la investigación, de acuerdo a la Declaración Universal de los Derechos Humanos, el Pacto de

Derechos Civiles y Políticos, el Pacto de Derecho Económicos Sociales y Culturales, las leyes chilenas y el Documento oficial de ética para la investigación de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

ii) El Comité de Ética considera que la investigación no vulnera la dignidad de los sujetos, no constituye una amenaza bajo ninguna circunstancia ni causa daño.

iii) Dejamos constancia que el profesor Gutiérrez y el estudiante de magister Puga, serán responsables por eventuales daños causado a las personas por errores que puedan cometerse durante la investigación.



**Dr. Carlos Conca R.**  
Profesor Titular  
Departamento de Ingeniería  
Matemática

*Firmado electrónicamente por*  
**Dra. Marcela Munizaga M.**  
Profesora Titular  
Directora Académica y de Investigación

  
**Dr. Patricio Jorquera E.**  
Secretario



Santiago, 12 de mayo de 2021

MMM/CCR/PJE/ra

Facultad de Ciencias Físicas y Matemáticas (FCFM) | Universidad de Chile  
Comité de Ética y Bioseguridad  
[sediraca@ing.uchile.cl](mailto:sediraca@ing.uchile.cl)

2



Documento emitido con Firma Electrónica Avanzada por la Universidad de Chile.  
La autenticidad puede ser verificada en:  
<https://ceropapel.uchile.cl/validacion/609c3aec45f292001c268ed3>

Figura B.3: Certificado emitido por el comité de ética FCFM. Página 2.

## **B.3. Hardware**

### **B.3.1. Computador para interacción usuaria**

- Fabricante: HP
- Modelo: HP Laptop 14-cf0xxx
- Procesador: Intel Core i3-8130U
- RAM: 8GB
- Disco: 256GB
- Sistema Operativo: Windows 10 Home

### **B.3.2. Servidor bluetooth**

- Fabricante: Lenovo
- Modelo: Lenovo ThinkPad Yoga 15
- Procesador: Intel Core i5-5200U
- RAM: 4GB
- Disco: 256GB
- Sistema Operativo: Ubuntu 18.04

### **B.3.3. Pulsera retroalimentación háptica**

- Fabricante: Xiaomi
- Modelo: Xiaomi Mi Band 3
- Conectividad: Bluetooth 4.2 BLE
- Dimensiones: 46.9mm x 17.9mm x 12mm
- Peso: 20g