UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

¿Cómo diseñan software los estudiantes de grado? Una revisión sistemática de la literatura

Patsy Helen Jones Dinelli

Programa de Posgrado en Ingeniería de Software
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Mayo de 2022

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

¿Cómo diseñan software los estudiantes de grado? Una revisión sistemática de la literatura

Patsy Helen Jones Dinelli

Tesis de Maestría presentada al Programa de Posgrado en Ingeniería de Software, Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Magíster en Ingeniería de Software.

Directores de tésis:
Diego Vallespir
Silvana Moreno

Director académico:
Diego Vallespir

Montevideo – Uruguay
Mayo de 2022

Jones Dinelli, Patsy Helen

¿Cómo diseñan software los estudiantes de grado? Una revisión sistemática de la literatura / Patsy Helen Jones Dinelli. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2022.

xi, 118 p.: il.; 29, 7cm.

Directores de tésis:

Diego Vallespir

Silvana Moreno

Director académico:

Diego Vallespir

Tesis de Maestría – Universidad de la República, Programa en Ingeniería de Software, 2022.

Referencias bibliográficas: p. 86 – 90.

1. Diseño de Software, 2. Estudiantes de grado, 3. Aprendizaje, 4. Ingeniería de software, 5. Enseñanza. I. Vallespir, Diego, Moreno, Silvana, . II. Universidad de la República, Programa de Posgrado en Ingeniería de Software. III. Título.

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

Dr. Ing. Daniel Calegari

Dr. Ing. Libertad Tansini

Msc. Ing. Leticia Perez

Montevideo – Uruguay
Mayo de 2022

Agradecimientos

A Pablo,
A mis padres y a mis hermanas,
A mis tutores Diego y Silvana por su generosidad y compromiso en la elaboración de esta tesis, que fue todo un desafío para mi.

RESUMEN

El diseño de software es un proceso creativo y fundamental para construir software de calidad. Realizar un diseño de software no es una tarea simple, no hay una única solución a un problema de diseño y son necesarios diferentes conocimientos sobre programación, diseño y otras habilidades para desarrollar un diseño de calidad.

A nivel educativo, el diseño es una disciplina compleja de entender para los estudiantes universitarios, y el éxito (es decir, construir un buen diseño) parece requerir un cierto nivel de desarrollo cognitivo que pocos estudiantes logran.

Este trabajo realizado pretende contribuir al conocimiento existente sobre la enseñanza y el aprendizaje del diseño de software por parte de estudiantes de grado o próximos a recibirse. A partir de una Revisión Sistemática de la Literatura (SLR) se busca conocer cómo diseñan software los estudiantes de grado y los problemas o dificultades con los que se encuentran tanto a nivel de formación como de concepto al estudiar o realizar un diseño de software.

La revisión sistemática realizada describe el motivo de la revisión, las preguntas de investigación, la estrategia de búsqueda, los criterios de inclusión/exclusión, la extracción y síntesis de los datos y los resultados utilizando la forma de presentar revisiones sistemáticas propuestas por Kitchenham and Charters (2007) y Kitchenham et al. (2015). Además, se elabora un protocolo de trabajo que fue utilizado en la ejecución de un piloto inicial con el propósito de validarlo, ajustarlo y mejorarlo. El protocolo final elaborado sirve cómo insumo para futuras investigaciones que busquen obtener nuevos resultados.

Como resultado de la SLR se seleccionaron 14 estudios que responden a la pregunta "¿cómo diseñan software los estudiantes de grado?". Estos 14 estudios identifican diferentes tipos de investigación, siendo casos de estudio (*case study*) y "experimento" los principales tipos de investigación encontrados. Las técnicas más utilizadas por los estudiantes al realizar un diseño de software en estos estudios son patrones de diseño, diagramas de clase, de secuencia y de objeto. Estos estudios identifican numerosos problemas que tienen los estudiantes al diseñar software, entre ellos, la falta de experiencia, la falta de análisis de los requisitos, la forma en que se dictan y formulan los cursos de diseño, la falta

de material y la complejidad que tiene el enseñar y aprender diseño.

Palabras claves:
Diseño de Software, Estudiantes de grado, Aprendizaje, Ingeniería de software, Enseñanza.

ABSTRACT

Software design is a creative and fundamental process to build quality software. Produce software design isn´t a simple task: there is no a single solution to a design problem and different knowledge about programming, design and other skills are needed to develop quality software.

At educational level, design is a complex discipline for university students to understand, and its success (that is, to build a good design) seems require certain level of cognitive development that few students achieve.

This work pretends to contribute to the existing knowledge about teaching and learning software design by undergraduate students or next to be received. From a Systematic Literature Review (SLR) we seek to know how undergraduate students design software and the problems or difficulties they find at training, studying or building a software design.

The systematic review describes the reason for the review, the research questions, the search strategy, the inclusion/exclusion criteria, the data extraction and synthesis, and the results using the method of systematic reviews proposed by Kitchenham and Charters (2007) and Kitchenham et al. (2015). In addition, a work protocol is elaborate that was used in the execution of an initial pilot with the purpose of validating, adjusting and improving it. The final protocol produced serves as an input for future research that seek to find new results.

As a result of the SLR, 14 studies were selected that respond the research question "how do undergraduate students design software?". These 14 studies identify different kinds of research, and the main kinds of research found have been study cases and "experiment". The techniques most used by students when doing a software design in these studies are design patterns, class diagrams, sequence diagrams and object diagrams. This studies identify many problems students have when designing software, including lack of experience, lack of requirements analysis, the way software design courses are taught and formulated, the lack of material and the complexity of teaching and learning

design.

Keywords:
Software design, Undergraduate, learn, Software engineering, teach.

Tabla de contenidos

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

El diseño de software es un tema difícil de entender (Nandigam et al., 2008), su éxito requiere de cierto nivel de desarrollo cognitivo; es necesario contar con una variedad de habilidades y conocimientos, que muchas veces no son bien enseñadas o aprendidas a nivel universitario (Cross, 2005).

Además, el diseño de software es una de las fases más importantes del ciclo de vida del software, porque puede ser usada como puente entre los requisitos y el código. Sin embargo, muchos estudiantes encuentran que los cursos de diseño de software son generalmente muy abstractos y aburridos, lo cual baja la motivación y las ganas de aprender (Cheng and Chen, 2018). Un objetivo fundamental es que los estudiantes graduados sean capaces de diseñar un sistema de software.

Este trabajo de tesis se centra en conocer si existen estudios científicos sobre la enseñanza y el aprendizaje del diseño de software por parte de estudiantes de grado o próximos a recibirse.

1.1. Motivación

Diseñar software es una tarea compleja, que requiere de conocimiento sobre programación y diseño, además de habilidades para resolver problemas. Los problemas de diseño son complejos en el sentido que involucran muchos conceptos interrelacionados en varios niveles de abstracción (Stuurman et al., 2016). En la frase "*Design as a 'wicked' problem*" de Budgen (2003), se resume las dificultades a las cuales se enfrenta el diseñador al diseñar software o la difícil tarea que éste tiene; por ejemplo que un problema de diseño no tenga

una solución definitiva (Bourque et al., 2014).

Diseño según la IEEE se define como "el proceso de definir la arquitectura, componentes, interfaces y otras características de un sistema o componente" o "el resultado de ese proceso". Visto como un proceso, el diseño de software es la actividad del ciclo de vida de ingeniería de software en el cuál los requisitos de software son analizados para producir una descripción de la estructura interna del software que sirve como base para su construcción (Bourque et al., 2014). El diseño de software juega un rol importante en el desarrollo de software. Durante el diseño de software los ingenieros realizan varios modelos que forman una clase de huella a una solución a ser implementada (Bourque et al., 2014) y en el ciclo de vida del software, el diseño de software es considerado como la tarea más desafiante comparada con otras (Hussain et al., 2017).

El propósito del diseño es simplemente la producción de una solución a un problema. El problema puede ser resumido por alguna forma de especificación de requisitos, y la tarea del diseñador es proporcionar una descripción de cómo estos requisitos deben ser cumplidos. El proceso de diseño involucra al diseñador en la evaluación de diferentes opciones, y en la toma de decisiones que pueden ser complejas e involucrar compensaciones entre factores como tamaño, velocidad y facilidad de adaptación, así como otros factores de problemas específicos (Budgen, 2003).

El acto de diseñar no se basa en una estrategia analítica, dirigida a identificar una única solución a un problema determinado por leyes físicas, sino en un proceso altamente creativo. Es muy poco probable identificar una única solución a un problema determinado. Un estudio experimental mostró que hay personas que diseñan mejor que otras (Curtis et al., 1988). Sin embargo, dado que el número de grandes diseñadores es muy pequeño, se necesita encontrar la manera de proporcionar las habilidades apropiadas de diseño a un grupo más amplio de la manera que sea más efectiva (Budgen, 2003).

Este trabajo tiene como punto de partida un conjunto de 6 artículos de investigación. Estos artículos estudian cómo los estudiantes realizan diseños de software basados en un caso de estudio. El caso de estudio consiste en diseñar una super alarma de reloj y clasificar los diseños realizados en 6 categorías según las características de los mismos (Tenenberg and Fincher, 2005). El caso de estudio realizado a nivel multi-nacional y multi-institucional explora la comprensión de los estudiantes sobre el proceso de diseño de software. Los participantes fueron reclutados entre estudiantes y educadores, incluyendo 314

participantes de 21 instituciones diferentes de EEUU, Reino Unido, Suiza y Nueva Zelanda. Los grupos de estudio se dividieron en tres tipos: 136 estudiantes de primera competencia (estudiantes que pueden programar al menos un problema del conjunto propuesto por McCracken (2004)), 150 estudiantes graduados (definidos como los estudiantes que tienen aprobada la octava parte de la carrera de licenciatura en Ciencias de la Computación) y 28 docentes que ocupaban cargos y enseñaban en programas de pregrado en Ciencias de la Computación. Para realizar la tarea de diseño a los participantes se les entregó una página que describía el caso de estudio y el comportamiento deseado para diseñar una super alarma de reloj que busca ayudar a los estudiantes a gestionar sus patrones de sueño. Los participantes debían realizar una solución inicial sobre la cual alguien que no necesariamente serían ellos podría trabajar. Los participantes realizaron esta tarea de forma individual sin realizar preguntas entre ellos o con los tutores. Este experimento es sobre el cual se basan los 6 artículos que fueron el conjunto de partida de este trabajo. En la tabla 1.1 se presenta el título de los 6 artículos.

En el artículo *Categorizing student software designs: Methods, results, and implication* de Eckerdal et al. (2006a) el objetivo es examinar las habilidades de los estudiantes avanzados que participaron en el experimento de Tenenberg and Fincher (2005) cuando diseñan software. Se propone y responde a la pregunta "¿Pueden los estudiantes próximos a graduarse diseñar sistemas de software?" agrupando los diseños en diferentes categorías según la semejanza semántica entre estos y se los analiza. Además se evalúan sus antecedentes académicos como avance en la carrera, cantidad de cursos de computación asistidos y conocimiento en lenguajes de computación. Los resultados obtenidos son que la mayoría de los estudiantes próximos a recibirse no pueden diseñar efectivamente un sistema de software.

El artículo *Can Graduating Students Design Software Systems?* de Eckerdal et al. (2006b) es una extensión del artículo de Eckerdal et al. (2006a). En este artículo se analiza la información demográfica y los antecedentes académicos obtenidos de los diferente estudiantes de grado que participaron del experimento; y se los relaciona con la forma en que diseñan. Los resultados obtenidos son que la mayoría de los estudiantes próximos a graduarse no pueden diseñar un sistema de software. De los factores demográficos analizados se observó que la edad del estudiante próximo a graduarse y el número de lenguajes que el estudiante usa no afecta su desempeño al realizar un diseño. Con respecto al género, las

mujeres realizan principalmente diseños que incluyen una visión parcial del sistema, identificando algunas partes pero no como éstas se relacionan. Los hombres, mayormente realizan diseños con información en texto, dibujos o detalles de la implementación poco importantes, sin brindar una visión general del sistema. Sin embargo el tomar más cursos de computación y la familiaridad de los estudiantes con los lenguajes que usan, mejoran las técnicas y calidad de diseño. El avance académico no muestra una mejora significativa en la calidad o realización del diseño.

En el artículo *Can graduating students design: revisited* de Loftus et al. (2011) el objetivo es investigar cómo diseñan software los estudiantes próximos a recibirse en grupo. Cada grupo realiza en 50 minutos un diseño de software sobre el tema de la "Alarma de reloj" del experimento de Tenenberg and Fincher (2005). Los diseños son evaluados por otro grupo de estudiantes que determinan cual es el mejor diseño. Los resultados obtenidos confirman que los estudiantes próximos a recibirse no pueden realizar diseños de software aunque los diseñen en grupos.

En el artículo *Graduating Students' Designs - Through a Phenomenographic Lens* de Thomas et al. (2014) el objetivo es investigar mediante un análisis fenomenográfico cómo "producen un diseño" los estudiantes de grado próximo a recibirse. A un grupo de 35 estudiantes se les pide "producir un diseño" sobre el requisito de la "Alarma de reloj" del experimento de Tenenberg and Fincher (2005). Los diseños se clasifican en categorías según sus características. Los resultados obtenidos muestran que los diseños realizados son mejores que los obtenidos en experimentos anteriores. Sin embargo aunque los estudiantes tengan conocimiento de los mismos métodos que los profesionales, la falta de experiencia no les permite realizar diseños elegantes.

En el artículo *Can Students Design Software? The Answer Is More Complex ThanYou Think* de Hu (2016) el objetivo es evaluar las habilidades de diseño de un grupo de estudiantes de una clase de diseño según la calidad de los diseños realizados. Los diseños son evaluados por el mismo instructor según un conjunto de criterios definidos y limitado a la experiencia del mismo. La conclusión es que los estudiantes de grado generalmente pueden diseñar software de forma inteligente si se brinda una educación de calidad.

El diseño de software no es una tarea simple y no hay una única solución a un problema de diseño. Esto genera el interés de conocer las razones por las cuales no es una tarea fácil. Además de conocer cuales son los problemas a los

Tabla 1.1: Estudios utilizados como punto de partida de la tesis.

Estudio
Categorizing studen software designs: Methods, results, and implications (Eckerdal et al., 2006a)
Can Graduating Students Design Software Systems? (Eckerdal et al., 2006b)
Can graduating students design: revisited (Loftus et al., 2011)
Graduating Students'Designs - Through a Phenomenographic Lens (Thomas et al., 2014)
Can Students Design Software ? The Answer Is More Complex Than You Think (Hu, 2016)
Students designing software: a multi-national, multi-institutional study (Tenenberg and Fincher, 2005)

que se enfrentan y que posibles soluciones existen para realizar un buen diseño de software, especialmente si son diseñadores novatos como los estudiantes de grado.

1.2. Objetivos del trabajo

El objetivo general de esta tesis es investigar la existencia de *artículos* sobre la forma en que los estudiantes de grado diseñan software, respondiendo a la pregunta de investigación **¿Como diseñan software los estudiantes de grado?**. Queremos conocer los tipos de estudios que existen sobre cómo diseñan software los estudiantes de grado. Conocer con que tipo de problemas o dificultades se encuentran los estudiantes al diseñar o aprender diseño, y como estos problemas y/o dificultades son evaluados o tratados. Averiguar si estos problemas se han podido mitigar, y cuales han sido las acciones tomadas para que no sucedan o sucedan de forma menos frecuente. Nos centramos en el diseño detallado.

1.3. Trabajo realizado

Para lograr el objetivo de esta tesis realizamos una revisión sistemática de la literatura para examinar si existen publicaciones sobre este tema. Utilizamos el método Revisión Sistemáticas de Literatura (SLR) para Ingeniería de Software propuesto por Kitchenham et al. (2015). La SLR es un método sistemático,

explícito y reproducible para identificar, evaluar y sintetizar el trabajo producido por investigadores, académicos y profesionales en un área elegida (Fink, 2019).

Nos interesa encontrar todas las publicaciones sobre este tema utilizando esta metodología bien definida. Definimos un protocolo de trabajo y nos aseguramos de seguir los pasos definidos en el mismo. El protocolo de la SRL fue ejecutado una vez utilizando un piloto, lo que nos permitió ajustar el protocolo de la SRL. Finalmente ejecutamos la SLR y obtuvimos los resultados para analizar y obtener conclusiones. La SLR fue ejecutada por un revisor que es el autor de esta tesis y supervisada por otro revisor que es uno de los co-tutores de la tesis.

1.4. Estructura del documento

La tesis se divide en 5 capítulos. El capítulo 2, "Revisiones Sistemáticas de la Literatura en Ingeniería de Software" describe todos los conceptos para entender y conducir una revisión sistemática de literatura (SLR) en Ingeniería de software. El capítulo 3, "Protocolos de trabajo" presenta la metodología de trabajo, mencionando los ajustes realizados al protocolo hasta obtener el protocolo de la SLR. El capítulo 4, "Ejecución de la Revisión Sistemática de literatura" presenta la ejecución del protocolo de la SLR y los resultados obtenidos relacionados con el objetivo de la tesis. Finalmente, el capítulo 5, "Conclusiones y trabajos a futuro" presenta las conclusiones del trabajo realizado y especifica posibles tareas a abordar como trabajo futuro.

Además, contiene 4 anexos. El anexo 1, "Planilla Selección de Cadena de búsqueda 2" se presenta la planilla con los 200 artículos devueltos al ejecutar la cadena de búsqueda 2. El anexo 2, "Formulario de Extracción de Datos" presenta el formulario de extracción final de protocolo. El anexo 3, "Protocolo de la SLR de la tesis" se presenta la versión final del protocolo de la SLR. El anexo 4, "Ejecución del piloto 1" presenta la ejecución del piloto 1 para evaluar el protocolo 1.

Capítulo 2

Revisiones Sistemáticas de la Literatura en Ingeniería de Software

Este capítulo presenta los conceptos necesarios para entender y conducir una revisión sistemática de la literatura en Ingeniería de Software. Se basa en Guidelines for performing Systematic Literature Reviews in Software Engineering (Kitchenham and Charters, 2007) y en el libro Evidence-based software engineering and systematic reviews (Kitchenham et al., 2015).

Una Revisión Sistemática de Literatura (o SLR la sigla comúnmente adoptada en ingeniería de software por convención (Kitchenham et al., 2015)) es un método para identificar, evaluar e interpretar todas las investigaciones relevantes de una pregunta de investigación particular, un área temática o un fenómeno de interés (Kitchenham and Charters, 2007). El objetivo de una SLR es proporcionar una imagen de la evidencia existente que sea completa, entendible y válida, tanto la identificación, el análisis y la interpretación, por lo tanto deben ser conducidas de manera científica y rigurosa (Wohlin et al., 2012). La SLR esta principalmente preocupada con el problema de agregar evidencia empírica la cual debe ser obtenida usando una variedad de técnicas y contextos, lo cual es comúnmente el caso de la ingeniería de software (Brereton et al., 2007). Los procedimientos seguidos para realizar la revisión deben ser lo mas objetivos, analíticos y repetibles posibles. En el caso ideal, si esa SLR fuera repetida por otros investigadores, debería seleccionar los mismo estudios de entrada y obtener las mismas conclusiones. Una SLR se considera un estu-

dios secundario porque produce material agregado de un conjunto de estudios primarios (Kitchenham et al., 2015). Un estudio secundario es un estudio que revisa todos los estudios relacionados a una pregunta de investigación específica con el objetivo de integrar/sintetizar evidencia relacionada con una pregunta de investigación específica (Kitchenham and Charters, 2007). Un estudio primario es un estudio empírico en el que directamente se hacen ediciones sobre los objetos de interés, mediante encuestas, experimentos, casos de estudio, etc (Kitchenham et al., 2015). Hay otros tipos de revisiones que son los estudios de Mapeos Sistemáticos y las Revisiones Terciarias. Un estudio de Mapeo Sistemático es un tipo de SLR, que surge si al examinar el dominio inicial de una SLR se descubre que es probable que exista muy poca evidencia o el tema sea muy amplio (Kitchenham and Charters, 2007). Su objetivo es clasificar la literatura relevante y los estudios con respecto a las categorías definidas (Pizard et al., 2019). Las Revisiones Terciarias son revisiones sistemáticas de revisiones sistemáticas, surge cuando ya existe un numero de SLR y se quiere responder preguntas de investigación más amplias (Kitchenham and Charters, 2007).

Algunas de las razones por las cuales se lleva a cabo una revisión sistemática de literatura son:

- resumir la evidencia existente sobre un tratamiento o tecnología.
- identificar temas en la investigación actual con el propósito de sugerir áreas para realizar una mayor investigación.
- proveer un marco de referencia/antecedentes para posicionar adecuadamente nuevas actividades de investigación.

Establecer la necesidad de realizar una revisión sistemática en ingeniería de software está más motivado por los requerimientos de los investigadores (para obtener logros académicos) que por problemas reales que se den en la práctica (Kitchenham et al., 2015). La motivación más importante para llevar a cabo una revisión sistemática o mapeo según un estudio de Zhang and Babar (2013) son:

a) obtener nuevos hallazgos de investigación.

b) describir y organizar el estado del arte de un área particular.

Una característica importante de una revisión sistemática es que es sistemática porque se lleva a cabo siguiendo un conjunto de procedimientos bien definidos

que suelen especificarse como parte del protocolo de la revisión (Kitchenham et al., 2015).

Realizar una SLR involucra varias actividades, que se agrupan en las siguientes tres fases: Planificar, Ejecutar e Informar la revisión (Brereton et al., 2007). En la figura 2.1 se detallan las tres fases con sus actividades.

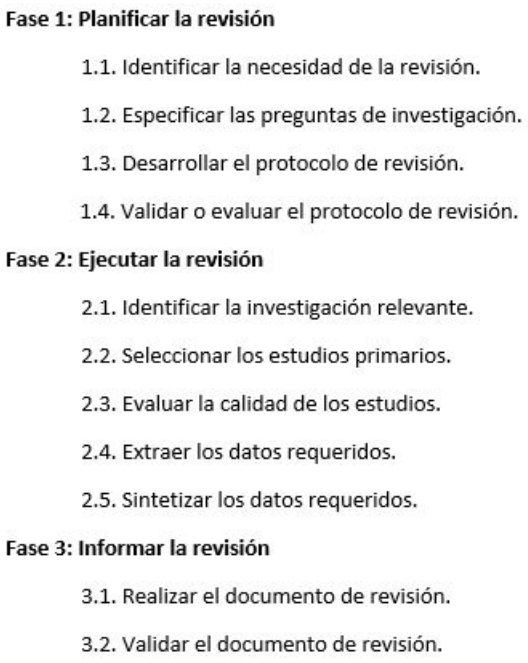

Figura 2.1: Resumen del proceso de la SLR. (basado en Kitchenham et al. (2015))

2.1. Actividades de cada fase de una SLR

A continuación se describen las diferentes actividades que tiene cada una de las fases de una SLR.

Fase 1: Planificar la revisión Sistemática.

Durante la planificación se especifican las tareas de diseño y se indica cómo el estudio será llevado a cabo. Cada tarea debe estar propiamente documentada en el protocolo de revisión (Kitchenham et al., 2015). Las actividades que involucra la fase 1 son:

1.1 Identificar la necesidad de la revisión.

1.2 Especificar las preguntas de investigación.

1.3 Desarrollar el protocolo de revisión.

1.4 Validar o evaluar el protocolo de revisión.

Fase 2: Ejecutar la revisión.

En esta fase se ejecuta el protocolo de revisión definido en la fase 1. Cualquier cambio que ocurra requiere que el protocolo sea modificado para que este cambio u otra circunstancia quede reflejado en el mismo. Todo cambio debe ser cuidadosamente documentado (Kitchenham et al., 2015). Las actividades que involucra la fase 2 son:

2.1 Identificar la investigación relevante.

2.2 Seleccionar los estudios primarios.

2.3 Evaluar la calidad de los estudios.

2.4 Extraer los datos requeridos.

2.5 Sintetizar los datos.

Fase 3: Informar la Revisión.

En esta fase se documenta y difunde a diferentes audiencias la revisión sistemática. Las actividades que involucra la fase 3 son:

3.1 Realizar el documento de revisión.

3.2 Validar el documento de revisión.

2.1.1. Fase 1: Planificar la Revisión Sistemática.

Una SLR es una actividad que consume mucho tiempo y requiere prestar mucha atención a los detalles, esto hace que la planificación sea un factor clave para lograr el éxito (Pizard and Acerenza, 2017). El resultado de esta fase debería ser un protocolo de revisión que contenga el propósito y los procedimientos de la revisión (Stapic et al., 2012). El foco se centra en desarrollar un protocolo de revisión, que juega un rol preponderante en la planificación de la revisión, proporcionando un marco para hacer y documentar las decisiones de diseño necesarias (Kitchenham et al., 2015). El protocolo pretende minimizar el sesgo en

el estudio, definiendo por adelantado cómo la revisión sistemática debe llevarse a cabo. El protocolo define el plan detallado para los revisores, especificando el proceso a seguir y las condiciones a aplicar para seleccionar estudios primarios, condiciones de borde, medidas de calidad, etc. Además, detalla quien debe realizar que tarea y subtarea. Es un documento sustancial y está sujeto a control de versiones. Todos los integrantes del equipo de revisión deben ser parte activa en el desarrollo del mismo (Brereton et al., 2007). Antes de desarrollar y validar el protocolo, los investigadores deben asegurar que la revisión es necesaria y factible (Kitchenham et al., 2015).

2.1.1.1. Identificar la necesidad de la revisión

Antes de comenzar la SLR es necesario chequear la no existencia de revisiones ya realizadas o en curso y la real necesidad de una nueva revisión sobre el tema. Esta tarea se puede realizar buscando revisiones en los motores de búsqueda relacionados al área temática. Si se identifica que una o mas revisiones contestan la pregunta de interés, se debe evaluar si cuenta con la calidad suficiente (Pizard et al., 2019).

2.1.1.2. Especificar las preguntas de investigación

Especificar las preguntas de investigación es una parte crítica de la planificación y constituye la base de toda la revisión. Decide qué estudios primarios serán incluidos o excluidos, que datos serán extraídos y como se van a sintetizar los mismos para poder responder a las preguntas de investigación. La naturaleza de las preguntas de investigación depende mucho del tipo de revisión que se quiere llevar a cabo (Kitchenham et al., 2015). Las preguntas se utilizan para construir cadenas de búsqueda, y son la parte del protocolo que no debe cambiar luego que el protocolo es aceptado (Brereton et al., 2007).

Para revisiones sistemáticas, las preguntas se refieren a la evaluación de una tecnología de ingeniería de software o de un proceso de investigación en particular. El término tecnología es usado en un sentido amplio para abarcar métodos o procesos de ingeniería de software, características particulares relacionadas a la gestión como atributos o de equipos de ingeniería de software (Kitchenham et al., 2015). Es importante en cualquier revisión formular la o las preguntas correctas. Estas preguntas pueden ser revisadas o re-formuladas durante el desarrollo del protocolo (Kitchenham et al., 2015). Una pregunta

correcta según Kitchenham and Charters (2007) es la que:

- sea significativa e importante tanto para profesionales como investigadores.
- de lugar a cambios o incremente la confianza en el valor de las practicas actuales de ingeniería de software.
- identifica diferencias entre las creencias comunes y la realidad.

2.1.1.3. Desarrollar el protocolo de revisión

El protocolo de revisión es un plan documentado donde se describe cómo se lleva a cabo la revisión. Un protocolo es valioso porque reduce la posibilidad de sesgo de los investigadores, limitando la influencia de las expectativas de estos; puede ser evaluado por otros investigadores y sirve como base para el reporte de revisión. Es importante que un protocolo esté estructurado de forma de ser fácilmente utilizado como un documento de referencia por un equipo de revisión y ser actualizado según sea necesario. Es un documento vivo que es probable sea actualizado mientras se lleva a cabo la revisión, donde todo cambio debe estar documentado y justificado (Kitchenham et al., 2015). Se recomienda que todos los miembros de la revisión sistemática participen activamente en el desarrollo del protocolo de revisión; y se realice una prueba piloto del mismo para de esta forma detectar posibles errores (Brereton et al., 2007) .

2.1.1.4. Validar o evaluar el protocolo de revisión

Validar y evaluar el protocolo es una actividad muy importante por ser el protocolo un elemento crítico en una revisión sistemática. Los investigadores deben establecer un procedimiento para evaluar el protocolo. Si hay financiación, a un grupo de expertos independientes se le debería solicitar que revisen el protocolo, y estos mismos expertos deberían revisar el protocolo final. En un estudio de maestría son los tutores a los cuales se les debe presentar el protocolo para revisión y críticas (Kitchenham and Charters, 2007).

2.1.2. Fase 2: Ejecutar la revisión.

Una vez que el protocolo está terminado y validado por los investigadores se puede poner en marcha la revisión. El objetivo de una revisión sistemática es encontrar tantos estudios primarios relacionados a la pregunta de investigación como sea posible, utilizando estrategias de búsqueda que logren un nivel de

completitud aceptable. La rigurosidad en el proceso de búsqueda distingue las revisiones sistemáticas de las tradicionales (Kitchenham and Charters, 2007). En esta etapa se establece la estrategia de búsqueda para obtener la mayor cantidad de publicaciones que permitan responder la o las preguntas de investigación (Kitchenham et al., 2015).

2.1.2.1. Identificar la investigación relevante

El foco de identificar la investigación es la identificación de los estudios primarios relevantes. Un elemento importante de una revisión sistemática es idear una estrategia de búsqueda que permita encontrar la mayor cantidad de estudios primarios relevantes para responder las preguntas de investigación. La estrategia de búsqueda involucra una combinación de métodos de investigación (Kitchenham et al., 2015). La rigurosidad en el proceso de investigación es un factor que distingue revisiones sistemáticas de revisiones tradicionales (Kitchenham and Charters, 2007). La principal actividad en este paso involucra la especificación de cadenas de búsqueda y aplicarlas en las bibliotecas digitales. Sin embargo, también incluye búsquedas manuales a revistas y conferencias, como también en sitios web o enviando preguntas a investigadores. Las búsquedas sistemáticas de estudios primarios basados en referencias a otros estudios se denomina "snowballing". La estrategia de búsqueda es una compensación entre encontrar todos los estudios relevantes y no tener un alto numero de falsos positivos que deben ser excluidos manualmente (Wohlin et al., 2012). Un método de búsqueda ampliamente utilizado son las búsquedas automáticas utilizando recursos como librerías digitales y sistemas indexados. Otros métodos incluyen búsqueda manual en revistas seleccionadas o en conferencias, chequeando artículos que son citados en el artículos incluido en la revisión (backward snowballing) y chequeando artículos que citan el articulo incluido en la revisión (forward snowballing). La estrategia de búsqueda debe lograr un nivel aceptable de completitud, este nivel de completitud varía según el tipo de revisión que se lleve a cabo. Para SLR cuantitativas, donde se compara tecnologías de ingeniería de software, un alto nivel de completitud es esencial. Para las SLR cualitativas donde se evalúan entre otros el riesgo, beneficios, factores motivacionales o que revisan los procesos de las SLR, un nivel mas bajo de completitud es aceptable (Kitchenham et al., 2015). Se consideran dos aspectos de la completitud del conjunto de artículos encontrados para seguir

una estrategia de búsqueda. El primer aspecto se relaciona con la *completitud del objetivo*, cuan completo el conjunto de artículos debe ser y el segundo aspecto se relaciona con la *evaluación de la completitud*, una vez se tiene el objetivo como sabemos si lo hemos alcanzado. En la completitud del objetivo se decide cuando detener la búsqueda y esto depende del nivel de completitud necesario para responder las preguntas de investigación en una revisión. Hay dos formas fundamentales para evaluar la completitud de un conjunto de estudios encontrados en la búsqueda de la literatura. La primera forma es utilizar el juicio personal, especialmente si los integrantes del equipo son investigadores experimentados. La segunda forma es calcular el recall y la precisión que se calculan de la siguiente forma: Recall → son todos los estudios relevantes encontrados en la búsqueda sobre los estudios relevantes totales. Precisión → son todos los estudios relevantes encontrados en la búsqueda sobre el número total de estudios encontrados. Lo ideal es tener un Recall alto para de esta manera encontrar la mayoría si no todos los estudios relevantes, y una alta precisión para que la carga de revisar los documentos por los revisores sea baja (Kitchenham et al., 2015). El sesgo de publicación refiere al problema de que los resultados positivos es mas probable que sean publicados que los resultados negativos. Y el concepto de positivo y negativo depende del punto de vista del investigador. El sesgo de publicación puede llevar a un sesgo sistemático en la SLR si no se realizan esfuerzos especiales para abordar este tema (Kitchenham and Charters, 2007).

2.1.2.2. Seleccionar los estudios primarios

Los criterios de selección de estudios primarios de una revisión son formulados para identificar aquellos estudios que son capaces de brindan evidencias para responder las preguntas de investigación. Los criterios son generalmente expresados como dos conjuntos: uno con los criterios de inclusión y otro con los de exclusión. Algunos criterios son bastante genéricos y fáciles de interpretar. Los criterios de selección pueden necesitar ser revisados varias veces durante la revisión (Kitchenham et al., 2015). Luego que los artículos primarios candidatos fueron obtenidos, estos necesitan ser evaluados para conocer su relevancia real. Los criterios de selección de los estudios primarios intentan identificar aquellos estudios primarios que brindan evidencia directa para responder las preguntas de investigación. Para reducir sesgos, los criterios de selección se deben esta-

blecer durante la definición del protocolo, aunque estos pueden ser refinados durante el proceso de búsqueda. Los criterios de inclusión y exclusión se deben basar en las preguntas de investigación (Kitchenham and Charters, 2007). El proceso de selección es un proceso de varios pasos. Inicialmente, luego que el conjunto de estudios primarios candidatos es identificado, se puede leer título y abstract y de esta forma descartar los estudios que son claramente irrelevantes. Si se tiene un conjunto de estudios candidatos muy grande se pueden aplicar las siguientes estrategias: 1) Refinar el string de búsqueda para mejorar el Recall y la precisión, 2) reducir el alcance de la revisión, 3) usar una herramienta de minería de texto para respaldar el proceso de selección, 4) aumentar el número de integrantes del equipo de revisión. Si el resultado del proceso de selección es un número grande de estudios los investigadores puede elegir completar el proceso usando solamente una muestra de los estudios (Kitchenham et al., 2015). Cuando un estudio de selección es llevado a cabo por un equipo de revisores (dos o más), se puede medir el acuerdo entre investigadores usando el coeficiente Kappa (K) (Kitchenham et al., 2015). El índice Kappa queda definido en la ecuación 2.1.

$$K = \frac{(acuerdoactual - acuerdoesperadoporcasualidad)}{alcancedeestarmejorqueporcasualidad} \tag{2.1}$$

El valor inicial del índice Kappa debe ser documentado en el documento final. Cada desacuerdo debe ser discutido y resuelto (Kitchenham and Charters, 2007). En la práctica es muy difícil determinar si un artículo es candidato solo usando título, abstract o keywords. En ingeniería de software los keywords no son consistentes entre las principales revistas y organizaciones como ACM o IEEE. En la mayoría de los casos es necesario leer la conclusión antes de tomar una decisión de incluir o excluir un estudio primario particular (Brereton et al., 2007). Si la revisión es realizada por un solo revisor (como un estudiante de maestría o doctorado) se debe considerar discutir los estudios incluidos o excluidos con un tutor u otro revisor. Alternativamente, los revisores individuales puede aplicar el test-retest, que es realizar la re-evaluación eligiendo de forma aleatoria un conjunto los estudios primarios ya analizados y chequear la consistencia de las decisiones de inclusión/exclusión (Kitchenham and Charters, 2007).

2.1.2.3. Evaluar la calidad de los estudios

La evaluación de la calidad permite analizar hasta qué punto los resultados de un estudio empírico son válidos y libres de sesgo. Para las revisiones sistemáticas evaluar la calidad de un estudio primario contribuye a mejorar el valor de la revisión de varias maneras. Por ejemplo, explicar cuándo diferencias en la calidad de estudios primarios puede explicar diferencias en los resultados del estudio o guiar la interpretación de revisiones futuras. Para evaluar la calidad se debe desarrollar una checklist de evaluación de la calidad para cada estudio (Kitchenham et al., 2015). Las checklist usualmente derivan de un conjunto de factores que pueden sesgar los resultados del estudio (Kitchenham and Charters, 2007). La evaluación de la calidad de los estudios primarios es importante para apoyar el proceso de inclusión/exclusión y para asignar ponderaciones a estudios específicos durante la etapa de síntesis (Brereton et al., 2007).

2.1.2.4. Extraer los datos requeridos

El objetivo de la extracción de datos es definir el formulario de extracción de datos donde se guarda la información obtenida por los investigadores de los estudios primarios. Para reducir los sesgos, el formulario de extracción debe ser definido y realizado en un piloto luego que el protocolo de estudio es definido (Kitchenham and Charters, 2007). Generalmente diferentes tipos de datos son extraídos para diferentes tipos de revisiones aunque todos incluyen algunos datos que son estándar, como por ejemplo el año y el tipo de publicación. Otra información que sea extraída de los estudios primarios depende mucho de las preguntas de investigación de la revisión. Adicionalmente, para los datos cualitativos, la extracción puede realizarse en equipo de dos o más revisores trabajando en conjunto para llegar a un acuerdo sobre los datos que se van a extraer. Si no se logra llegar a un acuerdo sobre los datos a extraer entre los investigadores, sera necesaria una revisión de la descripción de los datos y posiblemente de las preguntas de investigación (Kitchenham et al., 2015). El formulario de extracción debe ser diseñado para obtener toda la información para responder las preguntas de investigación de la SLR y del estudio de evaluación de la calidad de los estudios. En la mayoría de los casos, la extracción de datos define un conjunto de valores numéricos que se deben extraer de todos los estudios (por ejemplo intervalos de confianza). Los datos numéricos son importantes para realizar cualquier intento de resumir los

resultados de un conjunto de estudios primarios y son un prerrequisito para el meta-análisis. Con el formulario de extracción se debe realizar un piloto sobre ejemplos de estudios primarios. Además de incluir todas las preguntas necesarias para responder las preguntas de investigación y los criterios de evaluación, los formularios de extracción deben incluir información estándar (Kitchenham and Charters, 2007). En SLR con un solo revisor, como los estudiantes de maestría o doctorado, otras técnicas de verificación pueden ser utilizadas. Por ejemplo los tutores pueden realizar la extracción de datos en una muestra aleatoria de estudios primarios y sus resultados ser verificados contra los resultados del estudiante. Alternativamente, se puede usar un proceso de test-retest donde el investigador realizar una segunda extracción de una selección aleatoria de estudios primarios para verificar la consistencia de la extracción de datos (Kitchenham and Charters, 2007).

2.1.2.5. Sintetizar los datos

La síntesis de datos es el paso final de la fase de realizar la revisión. Durante esta actividad los datos extraídos son juntados y resumidos. En general, hay dos tipos de síntesis de datos: descriptiva (o narrativa) y cualitativa. La síntesis debe ser definida en el protocolo y esta determinado por el tipo de preguntas de investigación, pero también por el tipo de los estudios disponibles y la calidad de los datos (Stapic et al., 2012) El sintetizar los datos involucra recolectar y resumir los resultados incluidos en los estudios primarios. La actividad de síntesis de datos debe ser especificada en el protocolo de revisión, sin embargo algunos puntos no pueden ser resueltos hasta que los datos son analizados (Kitchenham and Charters, 2007). La síntesis narrativa, nos cuenta una historia que se origina desde las primeras evidencias. La evidencia e interpretación son estructuras, usando por ejemplo tabulación de datos, clustering o conteo de votos como una herramienta descriptiva. La síntesis narrativa se puede aplicar a estudios con datos cualitativos o cuantitativos, o a la combinación de ambos (Wohlin et al., 2012). La síntesis cualitativa la información también debe ser presentada en forma tabular, sin embargo los resultados de diferentes estudios, su resultado debe ser presentado en una forma comparable (Kitchenham and Charters, 2007). Independientemente del tipo de síntesis, las síntesis deben comenzar con la creación de un resumen que incluya los estudios. Los estudios incluidos en la revisión son generalmente presentados en una tabla que contiene todos los

detalles importantes. En la misma tabla o en otra, se pueden presentar elementos de calidad del estudio y riesgo de sesgo. Además, este proceso descriptivo debe ser explícitamente riguroso y ayudar a concluir si los estudios son similares y confiables para sintetizar (Wohlin et al., 2012).

2.1.3. Fase 3: Informar la revisión

Esta es la fase final de una revisión sistemática que involucra el escribir los resultados de la revisión y difundir los mismos a las potenciales partes interesadas (Kitchenham and Charters, 2007). Para asegurar que la revisión sistemática o el estudio de mapeo son beneficiosos para la comunidad científica, deben publicarse en revistas de investigación o en artículos de conferencias que sean de relevancia o arbitrados. Todos los detalles del proceso de revisión, la extracción de datos y el análisis deben ser reportados, incluyendo los criterios por los cuales se excluyeron artículos marginales (Kitchenham et al., 2015).

2.1.3.1. Realizar el reporte de revisión

Es donde se debe mostrar que se ha usado apropiadamente el proceso de una SRL, y que se ha utilizado de forma rigurosa. En particular se debe mostrar (Kitchenham et al., 2015): Trazabilidad para proveer al lector un link claro entre las preguntas de investigación y los datos necesarios para responder a dichas preguntas. Esto es más fácil en las revisión sistemáticas cuantitativas pero mucho más difícil de demostrar en las revisiones sistemáticas cualitativas. Repetibilidad para asegurar que la metodología está definida claramente y con suficiente detalle para que pueda ser repetida por otros investigadores.

El reporte debe tener una estructura bien definida. Aunque puede tener variaciones, se sugiere que contenga las secciones que se muestran en la tabla 2.1.

2.1.3.2. Validar o evaluar el reporte de revisión

Es donde el autor del reporte tiene la responsabilidad de leer y revisar el mismo, con el objetivo de asegurar que las siguientes situaciones sean verdaderas (Kitchenham et al., 2015):

- las preguntas de investigación están claramente especificadas y completamente respondidas.

Tabla 2.1: Secciones que debe contener el reporte de revisión (Kitchenham et al., 2015).

Sección	**Descripción**
Título	Especifica el tema del estudio y su metodología.
Abstract	Resumen abreviado de las diferentes partes del documento.
Introducción	Poner en contexto, justificar el porqué de la revisión sistemática y de las preguntas de investigación.
Antecedentes	Información del tema de la revisión y de estudios anteriores.
Metodología	Se deben incluir los elementos principales del protocolo.
Resultados	Se describe la salida del proceso de selección como de la extracción de datos.
Análisis	Se describe la salida del proceso de s´ıntesis. Para el estudio de mapeo puede consiste de tabulación y agrupación.
Discusión	Se realiza el resumen de las evidencias relacionadas a las preguntas de investigación, las limitaciones del estudio y las amenazas a la validez.
Conclusiones	Se realiza la interpretación general de los resultados del estudio y cuán bien se han respondido las preguntas de investigación. Además se realizan recomendaciones.

- la metodología de investigación está completa y correctamente especificada.
- Hay trazabilidad desde las preguntas de investigación hasta los dato de extracción, datos de síntesis y conclusiones.
- Todas las tablas y figuras usadas para presentar los resultados son correctas e internamente consistentes.
- En el caso de una revisión sistemática, las conclusiones están claramente escritas y dirigidas tanto para investigadores como practicantes.

Estos reportes suelen además ser evaluados en el marco de su difusión: un artículo de una revista académica será revisada previa a su publicación, expertos revisarán una tesis de doctorado. Si el protocolo fue revisado por un grupo de expertos previo a la ejecución de la revisión entonces se recomienda

que el mismo grupo revise el informe final. El proceso de evaluación puede utilizar la lista de verificación de calidad para revisiones sistemáticas (Pizard et al., 2019).

Capítulo 3

Protocolos de trabajo

Este capítulo presenta el protocolo de la SLR realizada para responder a la pregunta de investigación: ¿cómo diseñan software los estudiantes de grado?

Para obtener el protocolo final desarrollamos un primer protocolo que fue utilizado en un piloto. Luego de ejecutado el piloto ajustamos el protocolo incluyendo diferentes mejoras al mismo.

Los pasos que seguimos para obtener el protocolo se presenta en la figura 3.1. Definimos el protocolo y lo validamos ejecutando el piloto. Solamente fue necesario ejecutar un piloto para obtener el protocolo final, pero no necesariamente tendría que haber sido así. Por eso nuestro procesos identificaba la posibilidad de varios pilotos si estos fueran necesarios.

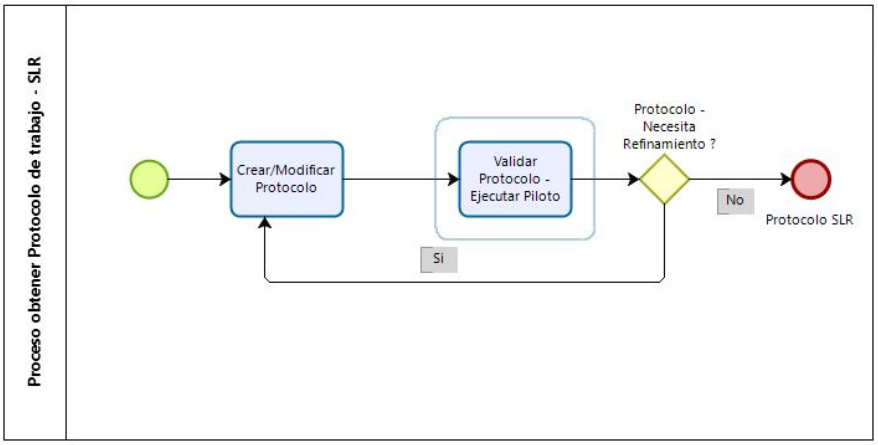

Figura 3.1: Proceso obtener protocolo SLR

La figura 3.2 presenta lo finalmente sucedido en el transcurso de la tesis. Primero, creamos un protocolo inicial al que llamamos “Protocolo Inicial”

y lo validamos ejecutando un piloto (Piloto 1). La ejecución del Piloto 1 mostró carencias y cuestiones a mejorar en el protocolo inicial. Las correcciones al protocolo inicial generaron un nuevo protocolo que es el protocolo final de la SLR.

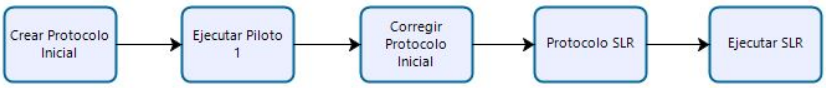

Figura 3.2: Proceso obtención protocolo SLR de la tesis

3.1. Antecedentes

En esta sección presentamos el punto "1.1. Identificar la necesidad de la revisión" de la figura 2.1 basado en Kitchenham et al. (2015). Lo primero que realizamos fueron búsquedas preliminares para conocer si existían revisiones sistemáticas de la literatura sobre el tema de esta tesis. Utilizamos la biblioteca digital SCOPUS ejecutando la cadena de búsqueda definida en la tabla 3.1. Definimos la cadena de búsqueda basados en la pregunta de investigación para verificar si existía alguna publicación sobre diseño de software que nombre estudiantes de grado y sea una revisión sistemática de literatura.

Tabla 3.1: Cadena de búsqueda en SCOPUS para evaluar la necesidad de la SLR.

(TITLE-ABS (software design AND student*) OR AUTHKEY (software design AND student*)) AND (TITLE-ABS (systematic review OR systematic literature review OR review*) OR AUTHKEY (systematic review OR systematic literature review OR review*))

Esta cadena la ejecutamos el día 05/03/2018 y devolvió 38 estudios. El criterio de evaluación que utilizamos para conocer si los estudios están relacionados con el objetivo de la tesis, fue leer título y resumen de cada uno de los estudios. Como resultado de las lecturas no encontramos estudios relacionados al objetivo de nuestra investigación. Por este motivo consideramos necesario realizar una SLR para conocer cómo diseñan software los estudiantes de grado.

3.2. Protocolo Inicial

En esta sección definimos diferentes etapas del protocolo inicial de la SLR, presentamos los puntos "1.2.Especificar las preguntas de investigación", "2.1.Identificar la investigación relevante", "2.2.Seleccionar los estudios primarios" y "2.4.Extraer los datos requeridos" de la figura 2.1 basado en Kitchenham et al. (2015).

3.2.1. Especificación de las preguntas de investigación

La pregunta de investigación general que interesa responder es: **¿Cómo diseñan software los estudiantes de grado?**. Además nos interesan conocer los problemas y dificultades con los que se encuentran los estudiantes de grado en el aprendizaje y resolución de problemas de diseño de software.

Para responder a la pregunta de investigación general definimos un conjunto de preguntas de investigación más específicas (ver tabla 3.2). Estas preguntas son las que guían la SLR.

Tabla 3.2: Preguntas de investigación Protocolo Inicial.

PI	**Pregunta**
PI1	¿Existen reportes de cómo diseñan software los estudiantes de grado?
PI2	¿Cuáles son las técnicas/métodos/principios de diseño de software que utilizan los estudiantes de grado en los diseños de software realizados?
PI3	¿Con que tipo de problemas se encuentran los estudiantes de grado al realizar un diseño de software?
PI4	¿Es evaluada la calidad de los diseños de software realizados por los estudiantes de grado?
PI4.1	¿Cómo se realiza la evaluación de la calidad del diseño de software realizado por los estudiantes de grado?

3.2.2. Estrategia de búsqueda

La cadena de búsqueda la obtenemos analizando las preguntas de investigación de la SLR. No todas las preguntas generan un término para la cadena de búsqueda, hay preguntas que están incluidas en otras. Un ejemplo es la pregunta PI1 que incluye todas las preguntas, por ser una pregunta general. En la tabla 3.3 relacionamos cada término de la pregunta con su correspondiente término en la cadena de búsqueda.

Tabla 3.3: Relación entre el término de la pregunta y la cadena de búsqueda.

Término pregunta	**Término cadena búsqueda**
Diseño de software	software design
Ingeniería de Software	software engineering
Estudiantes	undergraduate, student
enseñanza/aprendizaje	teach, learn, train

La cadena de búsqueda general que obtenemos se presenta en la tabla 3.4. La cadena se estructura en 3 partes. La primera parte hace referencia a lo que se quiere investigar, el "diseño de software". La segunda parte se refiere indistintamente a la población de estudio que son los estudiantes de grado y sus sinónimos, como a la enseñanza o aprendizaje y sus sinónimos. Buscamos todos los estudios que mencionen a estudiantes y/o aprendizaje y/o enseñanza. La tercera parte se refiere al contexto que es Ingeniería de Software.

Tabla 3.4: Cadena de búsqueda general.

((software design) and (student or undergraduate or teach or learn or train) and (software engineering))

La librería digital que utilizamos en la SLR como recurso de búsqueda es SCOPUS, por ser un buscador que indexa varias librerías. En la tabla 3.5 mostramos la cadena de búsqueda utilizada en SCOPUS. Esta cadena es una posible derivación de la cadena de búsqueda general.

Tabla 3.5: Cadena de búsqueda Protocolo Inicial en SCOPUS.

TITLE-ABS ({software design} AND (student* OR undergraduate* OR teach* OR learn* OR train*)) OR AUTHKEY ({software design} AND (student* OR undergraduate* OR teach* OR learn* OR train*)) AND (LIMIT-TO (LANGUAGE , "English")) AND $PUBYEAR \geq 1995$

La cadena de búsqueda la aplicamos sobre el título, resumen y palabras claves de los estudios indexados en los recursos de búsqueda. Utilizamos AUTHKEY para obtener sólo los estudios cuyas palabras claves son las elegidas por el autor y que en su opinión son las que mejor reflejan el estudio. Comúnmente se utiliza KEY en lugar de AUTHKEY. El utilizar KEY trae el problema de obtener estudios cuyas palabras claves son elegidas por los proveedores de contenido y estandarizadas basados en vocabularios que están disponibles públicamente. También limitamos la búsqueda a publicaciones en inglés y posteriores a 1995.

El filtro de publicaciones posteriores a 1995 lo decidimos porque al analizar el "*Analyzer search result*" de SCOPUS encontramos que desde el año 1995 se comienza a tener una cantidad significativa de artículos sobre el tema de estudio. Cuando ejecutamos la cadena en marzo de 2018 existían alrededor de 200 artículos mas en los años anteriores al 1995. Decidimos no incluir en la cadena de búsqueda el área de investigación para no perder estudios que estén clasificados equivocadamente en un área de investigación diferente pero correspondan a la ingeniería de software o no se clasifiquen por área de investigación.

La selección de los estudios primarios de la SLR está basado en los criterios de inclusión y exclusión definidos en la tabla 3.6.

Tabla 3.6: Criterios de inclusión/Exclusión.

INCLUIR	**EXCLUIR**
El artículo está vinculado al diseño detallado de software en estudiantes de grado.	Escrito en un idioma que no es inglés.
El artículo responde directamente una o más de las preguntas de investigación.	El artículo no está disponible en texto completo.
	Es una *keynote*, *workshop*, presentación, reporte de opinión o de experiencia.
	Es un artículo duplicado.
	Es un artículo corto de 2 o menos paginas.

En la figura 3.3 describimos el proceso de selección que consta de 4 etapas.

El proceso comienza con los estudios resultantes de aplicar la cadena de búsqueda sobre los recursos mencionados anteriormente. La primera etapa del proceso de selección se denomina *Seleccionar estudios primarios* . En esta etapa participan dos revisores, realizando la lectura de título y resumen (en paralelo) de todos los estudios devueltos por la cadena de búsqueda y aplicando a cada estudio los criterios de inclusión y exclusión. Cada revisor registra en una planilla su decisión (0=Rechazado; 1=Aceptado; 2=A discutir) y las observaciones y comentarios. Esta planilla está definida en el Anexo 1.

Durante la segunda etapa, denominada *Clasificar estudios seleccionados*, uno

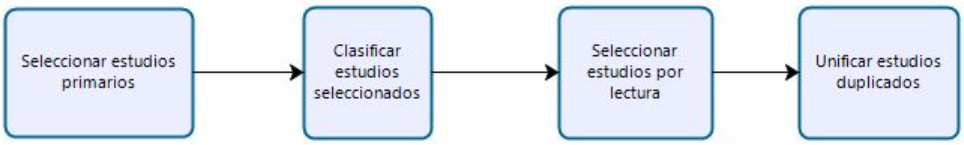

Figura 3.3: Proceso de selección de los estudios - Protocolo Inicial.

de los revisores realiza la unificación de ambas planillas de decisión unificando criterios de selección. Según la decisión que cada revisor marcó en su planilla individual, se toma la decisión final para cada estudio de acuerdo al criterio presentado en la tabla 3.7. Todos los estudios que son marcados como "a discutir" son evaluados por ambos revisores en una reunión de consenso. La reunión de consenso puede ser presencial o mediante e-mail. En esa reunión mediante consenso se acepta o rechaza cada estudio en discusión. Si para algún estudio no se llega a consenso se solicita la opinión de un tercer revisor (revisor experto). La idea es incluir la mayor cantidad de estudios, por este motivo para los casos en que un revisor selecciona "aceptado" y el otro revisor selecciona "a discutir" se decide aceptar el estudio primario.

Tabla 3.7: Casos según decisión de cada revisor.

			Revisor B	
		Aceptado	**Rechazado**	**A discutir**
	Aceptado	Aceptado	A discutir	Aceptado
Revisor A	**Rechazado**	A discutir	Rechazado	Rechazado
	A discutir	Aceptado	Rechazado	Experto

La tercera etapa, denominada *Seleccionar estudios por lectura* , tiene como entrada los estudios que resultaron aceptados de la etapa *Clasificar estudios seleccionados*. En esta etapa, cada revisor realiza la lectura completa de los estudios aplicando los criterios de inclusión/exclusión y registra en la planilla de selección su decisión (0=Rechazado; 1=Aceptado; 2=A discutir). Si para algún estudio no se llega a consenso se solicita la opinión del tercer revisor para tomar una decisión.

En la cuarta etapa, denominada *Unificar estudios duplicados* se tiene como entrada todos los estudios aceptados de la etapa *Seleccionar estudios por lectura*.

En esta etapa se analiza si hay estudios que son parte de otros de los estudios seleccionados. Se evalúa entre los revisores que acciones tomar para cada caso, si dejar todos los estudios, eliminar los que tienen menos contenido o el publicado más recientemente.

El resultado de la cuarta etapa son los estudios que son la entrada al proceso de extracción de datos.

Evaluar la calidad de los estudios primarios y quedarnos con los de calidad mas alta, mejora los resultados y evita sesgos en los mismos (Kitchenham et al., 2015). Sin embargo en esta tesis decidimos no evaluar la calidad de los estudios devueltos en el proceso de selección y seleccionar los estudios según cumplan con los criterios de inclusión/exclusión.

3.2.3. Extracción de datos

Para realizar la extracción de datos definimos un formulario de extracción de datos que contempla todas las preguntas de investigación. El formulario lo dividimos en dos partes. La primera parte es para obtener información general de un estudio. La segunda parte es para obtener información específica relacionada a las preguntas de investigación de nuestra SLR.

La validación del formulario de extracción la realiza cada revisor. Para realizar esta validación cada revisor extrae información de los primeros cinco estudios que son devueltos en el proceso de selección al ejecutar el piloto 1 y completa el formulario de extracción según la información obtenida de cada estudio. Si durante este proceso, alguno de los revisores sugiere algún cambio al formulario de extracción, solo se lo realiza si mediante consenso con el resto de los revisores se avala la modificación. Luego de la modificación del formulario de extracción, se realiza nuevamente la extracción. Todas las modificaciones que surgen al formulario de extracción son aceptadas previo a comenzar con la extracción de la totalidad de los estudios devueltos en el proceso de selección.

A continuación se presenta de forma concisa el formulario de extracción.

Parte 1 - Información general.

- ID. Selección
- Título
- Autores
- Resumen
- Fuente

- Año de Publicación
- Tipo de publicación (revista, conferencia, revista técnica)
- Universidad u Organización
- País

Parte 2 - Información específica.

- Objetivo del estudio
- El objetivo principal del estudio es conocer como diseñan software los estudiantes de grado (Sí, parcialmente)
- Métodos, técnicas y principios de diseño de software utilizados para enseñar
    - Artefactos (Sí/No) ¿Cuáles?
        - Seudocódigo
        - Diagrama de clases
        - Diagrama de secuencia
        - Patrones de diseño
    - Principios y métodos de diseño (Sí/No) ¿Cuáles?
        - Principios de diseño
        - Métodos de diseño
    - ¿Se nombran herramientas de diseño? (Sí/No) ¿Cuáles?
- Problemas o dificultades detectados.
    - Falta de material (Sí/No)
        - Requisitos mal formulados (o no entendibles)
        - Requisitos con escasas definición, información o incompletos
        - Problemas de diseño mal definidos
    - Problemas en los cursos de diseño de software (Sí/No)
        - Mal dictado el curso por los docentes
        - Mal formulado el curso
    - Problemas de los estudiantes con Diseño de Software (Sí/No)
        - Falta de motivación
        - Falta de experiencia
        - Mala formación de los estudiantes
        - Falta de conocimientos previos en diseño de software

- Evaluación de calidad del diseño realizado
    - Por atributos de calidad (Sí/No)
    - Por análisis de diseño final (Sí/No)

En el punto “El objetivo principal del estudio es conocer cómo diseñan software los estudiantes de grado (Sí, Parcialmente)” queremos responder a la pregunta de investigación PI1. La respuesta puede ser Sí o parcialmente, una respuesta negativa indica que el estudio no tiene información relacionada a la pregunta de investigación y se descarta. En el punto “Métodos, técnicas y principios de diseño de software utilizados para enseñar” queremos responder a la pregunta PI2. Para esto preguntamos sobre los artefactos que se encuentran en el estudio, si utilizan principio y métodos de diseño, y si alguna herramienta de diseño es nombrada en el estudio. En el punto “Problemas o dificultades detectados” queremos responder a la pregunta PI3. Buscamos información sobre la falta de material en diferentes puntos, problemas en los cursos de diseño de software o los problemas que los estudiantes tienen con el diseño de software. En el punto “Evaluación de calidad del diseño realizado” queremos detectar si en los estudios se nombra la evaluación de la calidad de los diseños realizados por los estudiantes.

En el proceso de extracción cada revisor completa el formulario de extracción de datos con la información obtenida de cada uno de los estudios devueltos en el proceso de selección. Se compara la extracción realizada por cada revisor y se analizan las diferencias. Las dudas sin resolver entre ambos revisores se discuten con el revisor experto.

3.3. Evaluación y ajustes al protocolo inicial

La validación del protocolo inicial la realizamos mediante la ejecución de un piloto al que llamamos “piloto 1”. La ejecución de un piloto es lo que permite validar que todo lo definido en el protocolo se lleve a cabo y se analice qué problemas puede tener el protocolo y qué pasos o etapas no se cumplen según lo definido. Nos ayuda a reducir la complejidad del proceso del protocolo, reformular las preguntas de investigación o refinar el formulario de extracción. El piloto permite lograr una mejor versión del protocolo basado en sucesivos refinamientos.

Verificamos que todo lo especificado en el protocolo se lleve a cabo y que

cada revisor comprenda claramente las tareas de las que es responsable según el protocolo definido.

El alcance del piloto 1 fue aplicar el proceso de selección del protocolo inicial a los primeros 100 estudios de los 1.328 estudios que devolvió la cadena de búsqueda el 13/03/2018. De los 100 estudios, seleccionamos 19 para realizar su lectura completa. De estos 19, luego de la lectura completa, se aceptaron 5 para realizar la extracción. Estos estudios cumplen con las condiciones de inclusión y exclusión especificadas.

Luego de ejecutar el piloto 1 detectamos varias modificaciones que debíamos realizar al protocolo inicial. La ejecución del piloto 1 se presenta en el anexo 4.

3.3.1. Cambios en las preguntas de investigación

Al realizar la lectura completa de los cinco artículos seleccionados en el piloto 1 encontramos que varios de ellos no solo presentan los problemas que tienen los estudiantes al realizar diseño sino también los problemas que tienen al estudiar o aprender diseño. Esto nos llevó a agregar una sub-pregunta de investigación que nos ayudara a identificar y clasificar este tipo de problemas. La pregunta de investigación 3 (PI3): ¿Con que tipo de problemas se encuentran los estudiantes de grado al realizar un diseño de software? está enfocada a los problemas que tienen los estudiantes al realizar un diseño de software, pero no al estudiar diseño. Por este motivo decidimos agregar la pregunta de investigación PI3.1:¿Con qué tipo de problemas o dificultades se encuentran los estudiantes de grado al estudiar o aprender diseño de software?

3.3.2. Cambios en la cadena de búsqueda

Al ejecutar la cadena de búsqueda (ver 3.5) definida en el protocolo inicial encontramos que devuelve estudios que no están relacionados con el diseño de software. Estos son sobre una población diferente a la que nosotros estamos investigando (estudiantes de grado) o están basados en la industria. Por otra parte, la cantidad de estudios que devuelve esta cadena es elevada para ser una revisión realizada en su mayoría por un único revisor (el autor de la tesis) y supervisada por sus tutores. Por estos motivos, decidimos modificar la cadena de búsqueda. La tabla 3.8 presenta la nueva cadena.

La cadena 3.8 la derivamos de la cadena definida en el protocolo inicial para SCOPUS pero teniendo en cuenta la cadena general 3.4. Esta cadena reduce la

Tabla 3.8: Cadena de búsqueda ajustada

TITLE (design* AND (student* OR undergrad* OR teach* OR learn* OR train* OR impart* OR comprehen*)) AND (ABS (“software design*”) OR ABS (design of software) OR ABS (design software) OR AUTHKEY (“software design*”) OR AUTHKEY (design of software) OR AUTHKEY (design software)) AND ABS (student* OR undergrad*) AND (LIMIT-TO (LANGUAGE , “English”))

cantidad de estudios que devuelve tratando de perder la menor cantidad de estudios valiosos para la investigación.

Analizando los 5 estudios que devuelve la cadena del protocolo inicial y los 6 estudios de la súper alarma de reloj, decidimos que el título y el resumen de la cadena de búsqueda los debíamos considerar de forma separada. El título debe tener la palabra diseño y la población del estudio (estudiantes) o algo de lo que se investiga (enseñar, aprender, entrenar, etc.). El resumen o las palabras de autor deben contener el contexto sobre el cual estamos trabajando que es el diseño de software. Además, el resumen debe referirse a los estudiantes o sus sinónimos, y solo consideramos estudios publicados en inglés.

Esta cadena devuelve una cantidad manejable de estudios y podemos suponer un número menor de estudios a ser descartados. Corremos el riesgo de no considerar estudios valiosos para la investigación por no devolverlos la cadena. Tomamos esta decisión para tratar de tener la menor cantidad de estudios que sean descartados en la selección.

Al ejecutar la cadena comprobamos que los artículos que devuelve incluyen los 5 estudios seleccionados en la ejecución del piloto 1 y los estudios relacionados con los experimentos de diseñar una súper alarma de reloj que son devueltos por SCOPUS a la fecha en que se realiza la búsqueda.

3.3.3. Cambios al formulario de extracción de datos

Al completar el formulario de extracción durante la ejecución del piloto inicial con los datos obtenidos de los estudios, encontramos que cierta información relevante obtenida no sabíamos en qué parte del formulario agregarla. El formulario contenía ciertos puntos ambiguos o poco explicados lo que hacia difícil comprender lo que se quería extraer. Además, no tenía una codificación que facilitara la síntesis de datos o la comparación de lo extraído entre diferentes revisores. Esto nos llevó a reestructurar el formulario, agregando los puntos que

faltaban y haciendo que el formulario fuera más amigable para los revisores. En el Anexo 2 se encuentra el formulario de extracción modificado.

La modificación realizada a la Parte 1 del formulario es el agregado de los siguientes puntos:

- Nivel de dificultad de la extracción
- Universidad u Organización de los autores
- País de los autores
- Universidad/Organización donde se realizó el estudio, investigación o experimento
- País donde se realizó el estudio, investigación o experimento

En la parte 2 las modificaciones que realizamos fueron las de agregar o quitar puntos. A continuación se especifica la nueva definición del formulario de extracción (los puntos nuevos se identifican con la palabra NUEVO).

- 1 - Objetivo del estudio
  - 1.1 - Identificar el tipo de estudio (por ejemplo: experimento, estudio de caso, encuesta, entrevista, etc.). (NUEVO)
- 2 - El objetivo principal del estudio es conocer como diseñan software los estudiantes de grado (Si, Parcialmente)
  - 2.1 - Resumen de lo que trata el artículo. (NUEVO)
- 3 - Métodos, técnicas y principios de diseño de software utilizados
  - 3.1 - Artefactos
    - 3.1.1. Seudocódigo
      - 3.1.1.1. En diseño realizado desde cero por estudiante.(NUEVO)
      - 3.1.1.2. En diseño realizado por tercero y analizado por estudiante.(NUEVO)
    - 3.1.2. Diagrama de clases
      - 3.1.2.1. En diseño realizado desde cero por estudiante.(NUEVO)
      - 3.1.2.2. En diseño realizado por tercero y analizado por estudiante.(NUEVO)
    - 3.1.3. Diagrama de secuencia

        - 3.1.3.1. En diseño realizado desde cero por estudiante.(NUEVO)
        - 3.1.3.2. En diseño realizado por tercero y analizado por estudiante.(NUEVO)
      - 3.1.4. Patrones de diseño
        - 3.1.4.1. En diseño realizado desde cero por estudiante.(NUEVO)
        - 3.1.4.2. En diseño realizado por tercero y analizado por estudiante.(NUEVO)
      - 3.1.5. Diagrama de objetos.(NUEVO)
        - 3.1.5.1. En diseño realizado desde cero por estudiante.(NUEVO)
        - 3.1.5.2. En diseño realizado por tercero y analizado por estudiante.(NUEVO)
    - 3.2 - Principios y métodos de diseño
      - 3.2.1. Principios de diseño
        - 3.2.1.1. En diseño realizado desde cero por estudiante.(NUEVO)
        - 3.2.1.2. En diseño realizado por tercero.(NUEVO)
      - 3.2.2. Metodos de diseño
        - 3.2.2.1. En diseño realizado desde cero por estudiante.(NUEVO)
        - 3.2.2.2. En diseño realizado por tercero.(NUEVO)
- 4 - Problemas o dificultades detectados
  - 4.1 - Falta de información.(NUEVO)
    - 4.1.1. Por Requisitos.(NUEVO)
      - 4.1.1.1. Mal formulados o inentendibles.(NUEVO)
      - 4.1.1.2. Escasa definición, información o incompletos.(NUEVO)
  - 4.2 - Problemas en los cursos de software.(NUEVO)
    - 4.2.1. Mal dictado el curso por los docentes.(NUEVO)
    - 4.2.2. Mal formulado el curso.(NUEVO)
    - 4.2.3. Falta de Material en el curso.(NUEVO)

- 4.3 - Problemas de los estudiantes con DS.(NUEVO)
  - 4.3.1. Falta de motivación.(NUEVO)
  - 4.3.2. Falta de experiencia.(NUEVO)

- 5 - Evaluación de calidad realizada
  - 5.1 - Por análisis del diseño realizado por los estudiantes

Codificamos cada uno de los puntos de la parte 2 del formulario para simplificar el entendimiento entre revisores. El punto 1.1 se agrega para identificar el tipo del estudio, que puede ser un experimento, un estudio de caso, una encuesta, una entrevista, etc., y así conocer sobre qué se basa el artículo a analizar. El punto 2.1 es un resumen del estudio explicando brevemente el objetivo del estudio, de que trata y el resultado del mismo. Al punto 3 se le agregan dos subdivisiones que indican si el diseño fue realizado desde cero por los estudiantes o son diseños realizados por terceros y analizados por los estudiantes. Esta subdivisión se hizo para analizar si esto mejora el aprendizaje de diseño de software cuando los estudiantes evalúan diseños realizados por otros (expertos, graduados o estudiantes). El punto 4 se estructura de forma diferente al formulario de extracción original porque ciertos puntos están contenidos en otros. Se lo re-clasifica en tres tipos de problemas o dificultades. El primer problema analiza la falta de información en los requisitos. El segundo investiga los problemas que pueden tener los cursos de diseño de software. El tercero analiza las dificultades a las que se puede enfrentar un estudiante de grado al realizar o aprender diseño de software. El punto 5 se modifica porque en la ejecución del piloto 1 encontramos que solo nos interesa evaluar la calidad de los diseños realizados por los estudiantes.

3.3.4. Cambios en los procesos de selección y extracción de datos

Al ejecutar el protocolo inicial vimos que las etapas definidas para los procesos de selección y de extracción eran diferentes a las realizadas. Esto nos llevó a modificar ambos procesos. El proceso de selección presentado en la figura 3.3 tiene modificaciones dentro de cada etapa porque la selección de estudios es realizada por el revisor 1 y supervisada por el revisor 2. Durante la primera etapa denominada *Selección por lectura de título y resumen* en lugar de participar dos revisores participa un revisor. El revisor 1 lee dos veces el

título y resumen de los estudios seleccionados con una semana de diferencia entre cada lectura y registra ambos resultados en una planilla diferente. En la segunda etapa denominada *Clasificación de estudios según criterio* se realiza la unificación de los criterios de ambas planillas registradas por el revisor 1. Todos los estudios en estado a discutir son evaluados en la reunión de supervisión con el revisor 2. En esta reunión se discuten dudas y el revisor 2 puede leer título y resumen de un numero aleatorio de estudios. La tercera etapa, denominada *Selección por lectura completa* el revisor 1 realiza la lectura completa de todos los estudios en estado aceptado que son el resultado de la segunda etapa. La lectura completa es realizada dos veces por el revisor 1 y registra en diferentes planillas su criterio de selección, espera una semana entre cada lectura completa. El revisor 1 realiza la unificación de las planillas obtenidas de cada lectura completa según los criterios planteados en la tabla 3.7. Se realiza una reunión de supervisión con el revisor 2 donde se evalúan los estudios con estado a discutir. El revisor 2 puede elegir de forma aleatoria estudios para realizar la lectura completa de los mismos. La cuarta etapa no tiene modificaciones.

El proceso de extracción lo ejecutamos de forma diferente a como esta definido en el protocolo inicial. Las modificaciones son porque es realizada por el revisor 1 y supervisada por el revisor 2. El revisor 1 realiza la extracción y completa el formulario de extracción. Espera una semana entre la primera extracción y realiza nuevamente la extracción para los estudios seleccionados (realizando el proceso de test-retest). Además, realiza la comparación y unificación de ambos formularios para cada estudio. El revisor 2 analiza el formulario de extracción unificado y las diferencia que encuentra son discutidas entre ambos revisores. El revisor 2 puede seleccionar al azar un conjunto de estudios y realizar la extracción para luego comparar con lo extraído por el revisor 1. Las dudas sin resolver entre ambos revisores se discuten con un tercer revisor.

En el anexo 3 se presenta la versión final del protocolo de la SLR.

Capítulo 4

Ejecución de la Revisión Sistemática de la Literatura

En este capítulo presentamos la ejecución del protocolo de la revisión sistemática de la literatura. La condición de parada de la ejecución es la definida en el capitulo 3. Al conjunto de artículos que devuelve la cadena de búsqueda solo consideramos los que fueron publicados desde el año 2010 en adelante.

La cadena de búsqueda es la definida en la tabla 3.8, que ejecutamos el 12/03/2019 y devolvió 200 artículos.

La ejecución del protocolo fue realizada por dos revisores, el revisor 1 que corresponde a la autora de esta tesis y el revisor 2 que corresponde a Silvana Moreno co-tutora de la tesis.

4.1. Proceso de selección

Las etapas del proceso de selección de la SLR se presentan en la figura 4.1. Las tres etapas son "Seleccionar estudios por lectura de título y resumen", "Clasificar estudios según criterios de selección" y "Seleccionar estudios por lectura completa".

Previo a comenzar con el proceso de selección construimos una planilla de selección donde registramos los resultados, ver anexo 1.

El proceso de selección lo realizamos para los primeros 119 artículos de los 200 artículos devueltos por la cadena de búsqueda, ordenados por año de publicación en forma descendente. Estos 119 artículos son los publicados desde

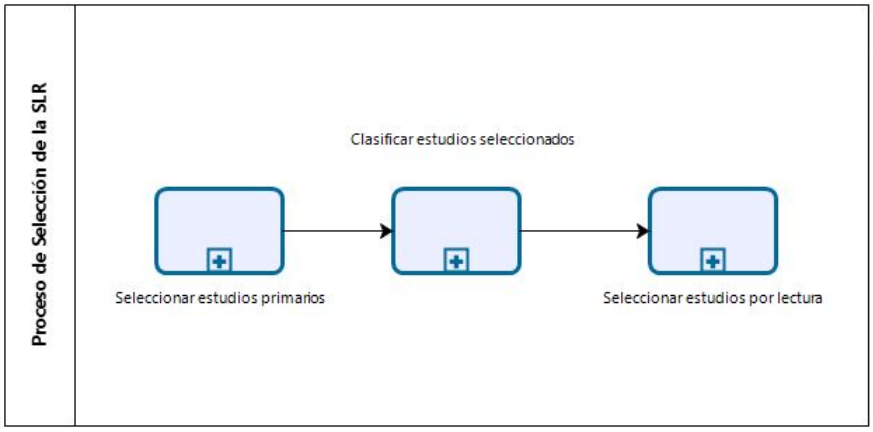

Figura 4.1: Etapas del proceso de selección de la SLR.

el año 2010 en adelante. La decisión de tomar el año de publicación 2010 como año de corte, es una decisión arbitraria basada en los resultados del "*Analyzer search result*" de SCOPUS. Estos resultados muestran que desde el año 2010 el aumento de artículos publicados que devuelve la cadena de búsqueda es continuo. En la etapa 1, como se muestra en la figura 4.2 el revisor 1 realiza la lectura de título y resumen; y aplica los criterios de inclusión y exclusión.

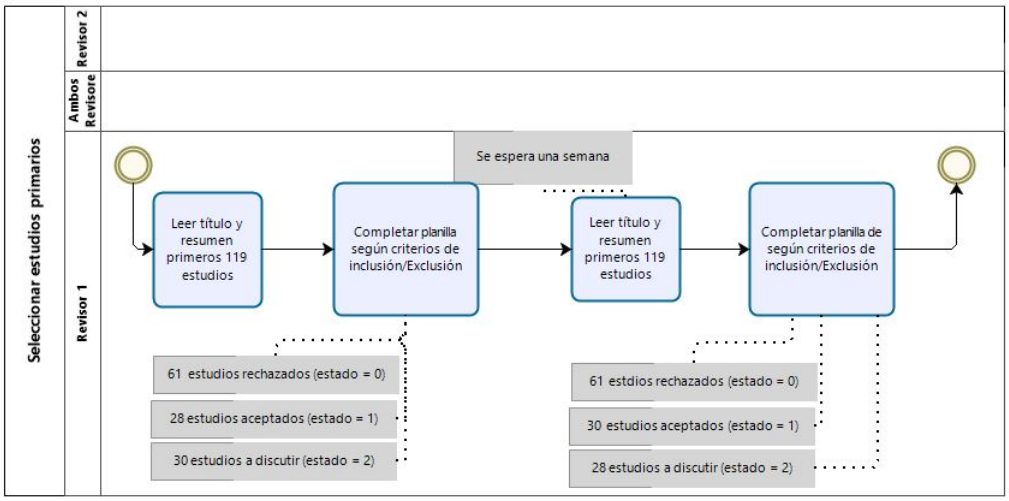

Figura 4.2: Etapa 1 Proceso de selección de la SLR.

La decisión (0=rechazado; 1=aceptado; 2=a discutir) para cada estudio es registrada en la planilla de selección. Con una espera de una semana el revisor 1 vuelve a realizar el proceso de selección sobre los mismos 119 artículos y

completa nuevamente la planilla de selección. En la primera selección el revisor 1, clasifica de los 119 artículos, 61 rechazados, 28 aceptados y 30 a discutir. En la segunda selección, el revisor 1 clasifica 61 artículos como rechazados, 30 como aceptados y 28 como a discutir. Finalizada la etapa 1, se da comienzo a la etapa 2 que se muestra en la figura 4.3, donde el revisor 1 realiza la unificación de las planillas de ambas selecciones.

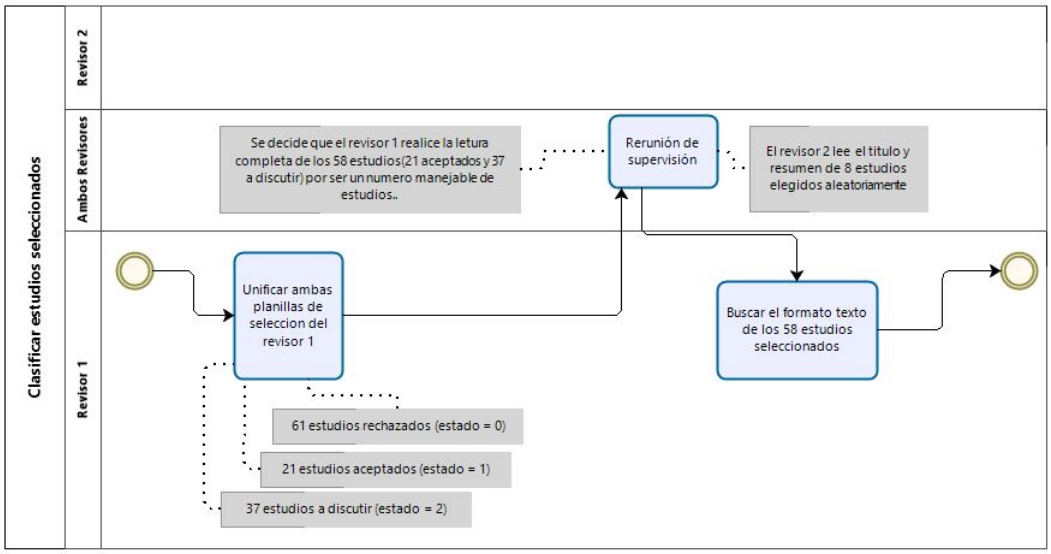

Figura 4.3: Etapa 2 Proceso de Clasificación de la SLR.

El resultado de la unificación son 61 artículos rechazados, 21 artículos aceptados y 37 artículos a discutir. El criterio para la unificación lo mostramos en la tabla 4.1, donde rechazamos el artículo si ambas selecciones están en estado rechazado, lo aceptamos si ambas selecciones están en estado aceptado y lo dejamos en estado a discutir en los otros casos. Los artículos en estado a discutir son discutidos con el revisor 2 en la reunión de supervisión.

En la reunión de supervisión, se reúnen el revisor 1 y 2 a revisar las dudas que tiene el revisor 1 en los artículos con estado a discutir. El revisor 2 lee titulo y resumen de un conjunto de artículos de los cuales el revisor 1 tuvo dudas. Como la cantidad de artículos en estado aceptado y a discutir no es elevado, se toma la decisión de que el revisor 1 realice la lectura completa de todos los artículos. El resultado de la etapa 2 son 58 artículos para realizar lectura completa, todos los artículos son buscados y encontrados en la web en formato de texto completo.

En la etapa 3, que se muestra en la 4.4 el revisor 1 lee el texto completo de

Tabla 4.1: Unificación de criterios de selección.

Revisión1	Revisión2	Unif.	Observación
0	0	0	Se rechaza
1	1	1	Se acepta
2	2	2	A discutir en reunión de consenso
1,2	0	2	A discutir en reunión de consenso
0,2	1	2	A discutir en reunión de consenso
0,1	2	2	A discutir en reunión de consenso

los 58 artículos y completa la planilla de selección con su decisión de clasificación según los criterios de inclusión/exclusión. Con una espera de una semana el revisor 1 realiza nuevamente la lectura completa de los mismo 58 artículos. En la primera lectura completa el revisor 1 clasificó 27 artículos rechazados, 20 aceptados y 11 a discutir. En la segunda lectura completa el revisor 1 clasificó 32 artículos rechazados, 23 aceptados y 3 a discutir. El resultado de la unificación de ambas planillas son 27 artículos rechazados, 18 aceptados y 13 a discutir. En la primera reunión de supervisión el revisor 2 decide leer todos los artículos con estado a discutir y los clasifica con estado rechazado. Con el resultado de la clasificación de estos artículos, el revisor 1 unifica la planilla de clasificación, con 13 artículos a discutir y 18 artículos aceptos. En la segunda reunión de supervisión, de los 13 artículos en estado a discutir, se llega por consenso que todos pasen a estado rechazado. El resultado de la etapa 3 por lectura completa, son 18 artículos para realizar la extracción.

Los detalles de los 18 artículos se presentan en la tabla 4.2.

Tabla 4.2: Estudios incluidos luego del proceso de selección de la SLR.

ID	Anio	Titulo	Autores
16	2017	Raspberry Pi creativity: A student-driven approach to teaching software design patterns (Williams and Kurkovsky, 2017)	Williams C., Kurkovsky S.

ID	Anio	Titulo	Autores
17	2017	Improving the Teaching of Software Design with Automated Modelling of Syntactic Dependencies (Steppe et al., 2017)	Steppe K., Chin S., Tuck W.W.
19	2017	Do UML object diagrams affect design comprehensibility? Results from a family of four controlled experiments (Torchiano et al., 2017)	Torchiano M., Scanniello G., Ricca F., Reggio G., Leotta M.
20	2017	Student software designs at the undergraduate midpoint (Thomas et al., 2017)	Thomas L., Zander C., Loftus C., Eckerdal A.
24	2017	A study of the use of a reflective activity to improve students' software design capabilities (Coffey, 2017)	Coffey J.W.
25	2017	Imparting software engineering design skills (Thevathayan and Hamilton, 2017)	Thevathayan C., Hamilton M.
26	2017	Conceptions of the students around object-oriented design: A case study (Flores and Medinilla, 2017)	Flores P., Medinilla N.
30	2017	Teaching software modeling and design (Gomaa, 2017)	Gomaa H.
35	2016	Using examples for teaching software design: An experiment using a repository of UML class diagram (Karasneh et al., 2015)	Karasneh B., Jolak R., Chaudron M.R.V.
38	2016	Can students design software? the answer is more complex than you think (Hu, 2016)	Hu C.

ID	Anio	Titulo	Autores
52	2015	Uncovering students' common dificulties and strategies during a class diagram design process: An online experiment (Stikkolorum et al., 2015)	Stikkolorum D.R., Ho-Quang T., Karashneh B., Chaudron M.R.V.
57	2014	Teaching analysis of software designs using dependency graphs (Steppe, 2014)	Steppe K.
62	2014	A course for teaching integrated system design to computer engineering students (Zualkernan, 2014)	Zualkernan I.A.
68	2014	Graduating students' designs - Through a phenomenographic lens (Thomas et al., 2014)	Thomas L., Eckerdal A., McCartney R., Moström J.E., Sanders K., Zander C.
98	2014	Teaching object-oriented software design within the context of software frameworks (Ali et al., 2011)	Ali Z., Bolinger J., Herold M., Lynch T., Ramanathan J., Ramnath R.
102	2011	An environment for project-based collaborative learning of software design patterns (Jeremic et al., 2011)	Jeremic Z., Jovanovic J., Gasevic D.
104	2011	Can graduating students design: Revisited (Loftus et al., 2011)	Loftus C., Thomas L., Zander C.
112	2010	Helping students learn expert software design strategies (Wright, 2010)	Wright D.R.

Los artículos que tratan el mismo tema son unificados en un mismo estudio como mostramos en la tabla 4.3.

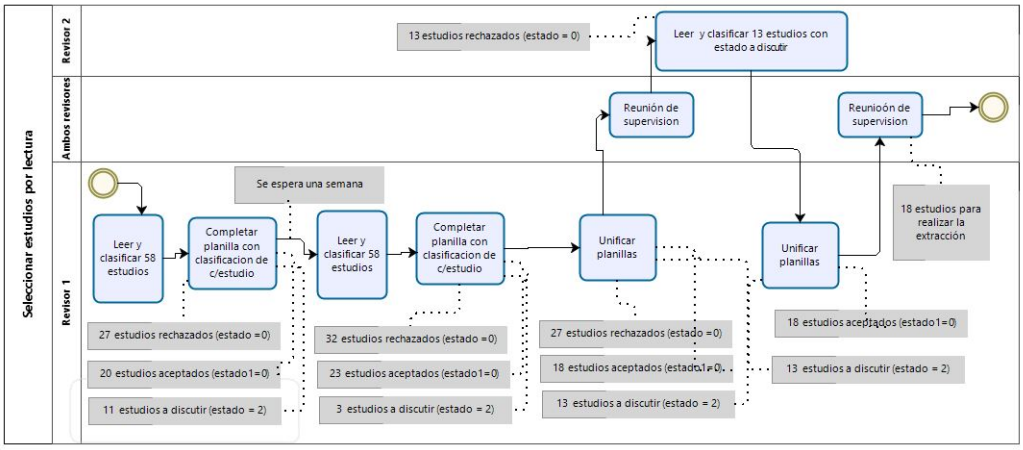

Figura 4.4: Etapa 3 Proceso de selección por lectura completa de la SLR.

Tabla 4.3: Unificación de ID con Estudios.

Estudio	ID
S1	16
S2	17, 57
S3	19
S4	20, 38, 68, 104
S5	24
S6	25
S7	26
S8	30
S9	35
S10	52
S11	62
S12	98
S13	102
S14	112

4.2. Proceso de extracción

En la figura 4.5 detallamos las actividades del proceso de extracción de la SLR.

El revisor 1 realiza la extracción de datos de los 18 estudios seleccionados en el proceso de selección y completa el formulario de extracción con la información obtenida de cada uno de los estudios. Se espera una semana antes de que el revisor 1 realice una nueva extracción a los mismos 18 estudios (realizando el proceso de test-retest). El revisor 1 unifica ambos formularios de extracción en un único formulario controlando las diferencias. El revisor 2 revisa el formulario

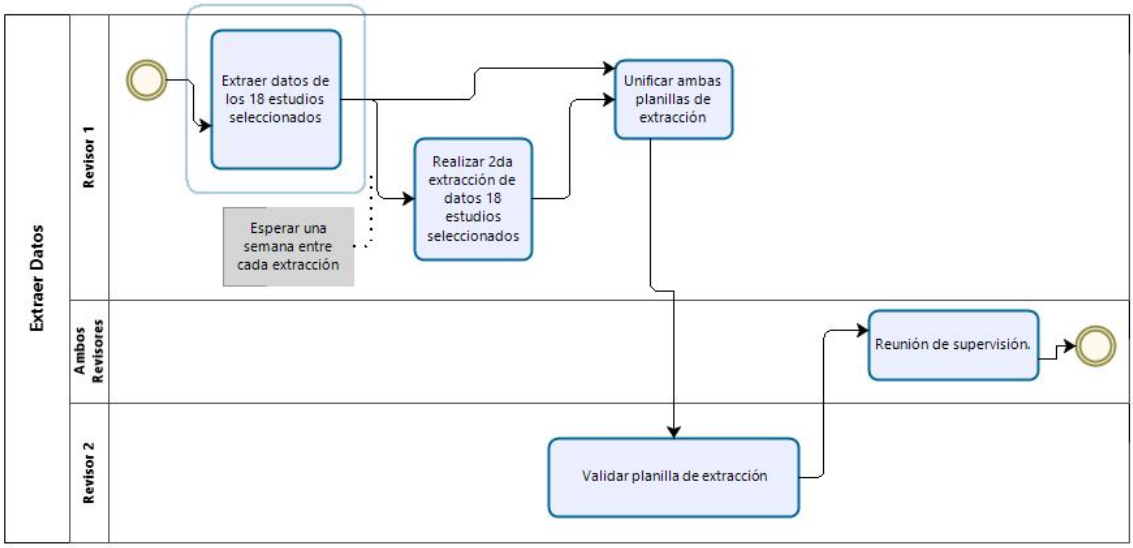

Figura 4.5: Proceso de extracción de la SLR.

unificado para un conjunto de estudios, y en una reunión de supervisión entre ambos revisores se discuten las correcciones y las dudas que le surgieron al revisor 1 durante la extracción.

4.3. Proceso de Síntesis de datos

En esta sección se presenta el punto "2.5.Sintetizar los datos requeridos" de la figura 2.1 basado en Kitchenham et al. (2015).

En el proceso de síntesis de datos se presentan los resultados basados en la información obtenida durante el proceso de extracción de los 18 artículos unificados en 14 estudios. La gráfica de la figura 4.6 muestra la cantidad de artículos que se obtuvieron en cada etapa de la SLR.

Los objetivos de los estudios se escriben en su idioma original porque se realiza una extracción textual de los mismos. En la tabla 4.4 detallamos los objetivos de cada uno de los 18 artículos alineados con el objetivo de la investigación. Estos artículos investigan de forma directa o indirecta como diseñan software los estudiantes de grado o las dificultades con las que se encuentran al diseñar o aprender diseño de software.

Los estudios S1 y S2 presentan técnicas y herramientas que influyen en la enseñanza y aprendizaje de diseño de software. El estudio S1 hace foco en la creatividad aplicada a un curso de patrones de diseño. El estudio S2 son

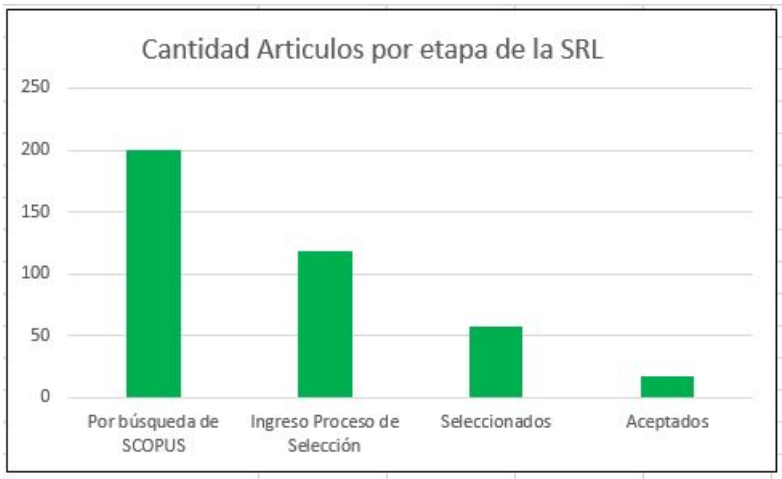

Figura 4.6: Proceso de extracción de la SLR.

técnicas y una herramienta que automatiza la creación de dependencias de grafos.

El estudio S3 es una investigación que evalúan los beneficios que se pueden obtener al utilizar un diagrama de clases y de objetos para comprender un diseño de software.

Los estudios S4 y S5 estudian como los estudiantes realizan diseños a partir de los requisitos especificados, que dificultades encuentran y como las pueden mejorar. Además, en el estudio S4 se investiga como evoluciona el diseño de software en los estudiantes al avanzar en la carrera y que resultado se obtiene al diseñar en grupo, si son capaces de evaluar diseños de otros estudiantes y diferenciar la calidad de los diseños. Por otra parte, el estudio S5 investiga como mejora el diseño al mejorar los requisitos.

El estudio S6 presenta como el utilizar metodología de investigación de acción en un curso de Diseño de Software mejora las habilidades de diseño en los estudiantes.

El estudio S7 es una investigación del tipo caso de estudio cualitativo sobre los conceptos y conocimiento que tienen los estudiantes sobre diseño de software orientado a objetos previo al comienzo de un curso sobre este tema.

Los estudios S8 y S11 describen las experiencias de dictar cursos de diseño a estudiantes de grado y las lecciones aprendidas. El estudio S8 es la experiencia del autor al dictar cursos de modelado y diseño. El estudio S11 describe la experiencia, los resultados y las lecciones aprendidas de enseñar diseño durante 5 semestres en los primeros años de la carrera.

El estudio S9 es un experimento controlado donde se analiza el impacto que tiene que estudiantes de grado utilicen un conjunto de ejemplos de diseño de software mientras están creando sus diseños.

El estudio S10 es un experimento online para descubrir las dificultades y estrategias de modelados que presentan los estudiantes al realizar diseños de diagrama de clases.

Los estudios S12 y S14 estudian la utilización de framework y como estos pueden ayudar a mejorar los diseños de los estudiantes.

El estudio S13 evalúa cuan efectivo es para aprender patrones de diseño tener un entorno de aprendizaje colaborativo.

Tabla 4.4: Objetivo de los estudios en su idioma original para la SLR.

Est.	**ID**	**Objetivo**
S1	16	This paper presents the experience of using student creativity with the Raspberry Pi to drive their learning in an upper level software design patterns course.
S2	17	In this paper, we present our findings teaching workshops for second-year undergraduates using the tool. The students were able to use the tool to analyze and compare designs.
S2	57	In this paper we present findings from a study with thirty third- and fourth-year undergraduates indicating that most were able to use the technique to analyze and compare designs after a single short workshop and indicate that they are likely to continue use the technique in the future.
S3	19	The main objective of our study is to assess whether the use of UML (Unified Modeling Lan- guage) object diagrams improves comprehensibility of software design when this kind of diagrams is added to UML class diagrams
S4	20	We were looking for characteristics of the development of skill at software design as students progress through the curriculum

Est.	ID	Objetivo
S4	38	As an ABET program evaluator, the author noticed that computer science programs take different approaches to teaching software design. While some programs have a dedicated course about design, most seem to teach design in a software engineering or software development course. We do offer a dedicated design course in our own computer science program. Given the aforementioned studies, we asked this question to ourselves: In what way can our students design software after they received some significant formal education on design? We will present our findings in the next section. But a viewpoint we try to make in this article is that designing software is an inherently complex, and undefined process. Thus, assessing students' software design competencies requires substantially different measures from design artifacts students produce
S4	68	Our main focus in this paper is a phenomenographic analysis of the phenomenon "produce a design."
S4	104	In this paper we revisit research that started in 2003-2004 as part of the 'Scaffolding' experiment.[...] In Semester one of 2009-2010, we reconsidered the experiment and extended it at Aberystwyth University. [...] The overarching research question was slightly different from the original: "Can graduating students design software systems in groups?".
S5	24	One of the goals of the current work is to analyze the reflections of students to get a sense of their nature, depth, and contribution to understanding software design. //The goal of the current study is to explore ways to foster better reflection in students.Specifically, the goal is to study how reflection on deficiencies in a preliminary design might help students better anticipate what is needed in a design in subsequent projects.
S6	25	This paper presents the result of our action research to improve student design skills. Our approach combines project-based learning with weekly quizzes, tests and active learning tasks.

Est.	ID	Objetivo
S7	26	The aim of this research is to know the ideas that the students have about designing with objects before they started a module corresponding to this topic.
S8	30	This paper describes my experience with teaching courses on software modeling and design to undergraduate and graduate (Masters and PhD) students.
S9	35	The goal of this research is to study the effects of using a collection of examples for creating a software design.
S10	52	This paper describes an online experiment with the aim to reveal common diculties and modelling strategies of students during a class diagram design task.
S11	62	This paper describes the development and experiences of teaching five semesters of a specialized design course for a computer engineering program taught in the junior year.
S12	98	This paper presents and discusses a methodology developed for designing software in the context of frameworks to overcome these issues. We show how design patterns can serve as the bridge between the paradigms imposed by the framework and the ideal, unconstrained design of the system.
S13	102	In this paper, we propose a project-based collaborative learning environment for learning software design patterns that integrates several existing educational systems and tools based on the common ontological foundation.
S14	112	This paper discusses the results of a pilot study conducted to make an initial appraisal of the tool's efectiveness.

En la tabla 4.5 identificamos los diferentes tipos de estudios de los artículos a los cuales se les hizo la extracción. Los principales tipos de estudios son sobre "casos de estudio" o "experimento".

En la tabla 4.6 presentamos los 14 estudios seleccionados especificando lugar y tipo de la publicación.

Los artículos seleccionados fueron publicados entre los años 2010 y 2017. En la gráfica de la figura 4.7 vemos la cantidad de artículos publicados por año,

Tabla 4.5: Tipos de estudios.

Tipo de Estudio	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Experimento			X	X	X		X		X				X	X
Action Research						X								
Caso de estudio	X	X		X				X		X	X	X		
Cuestionario										X				
Entrevista		X												

Tabla 4.6: Principales características de los estudios seleccionados de la SLR.

Lugar	**Conferencia/ Revista**	**Estudio**
IEEE Frontiers in Education Conference (FIE)	Conferencia	S1, S12
IEEE Conference on Software Engineering Education and Training	Conferencia	S2
Journal of Visual Languages & Computing	Revista	S3
ACM Conference on Innovation and Technology in Computer Science Education	Conferencia	S4
ACM SIGCSE Technical Symposium on Computer Science Education	Simposio	S4, S5
Conference on International Computing Education Research (ICER)	Conferencia	S4
Australasian Computing Education Conference (ACE)	Conferencia	S6
Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (JIISIC)	Conferencia	S7
Central Europe Workshop Proceedings (CEUR)	Workshop	S8
Asia-Pacific Software Engineering Conference (APSEC)	Conferencia	S9
Educators' Symposium @ MODELS	Simposio	S10
IEEE Global Engineering Education Conference (EDUCON)	Conferencia	S11
International Journal of Engineering Education	Revista	S13
International Conference on Software Engineering and Data Engineering (SEDE)	Conferencia	S14

mostrando que el año 2017 es el año con mas publicaciones.

En la tabla 4.7 agrupamos los artículos clasificados según el tipo de publicación, siendo los artículos de conferencia donde se realizaron más publicaciones.

Tabla 4.7: Artículos según tipo de publicación.

Tipo Publicación	**Estudio**
Conferencia	S1, S2, S4, S6, S7, S9, S11, S12, S14
Revista Científica	S3, S13
Simposio	S4, S5, S10
Workshop	S8

En la tabla 4.8 mostramos la distribución de los artículos según país de todos los autores que participaron en el estudio, siendo Estados Unidos el país

Figura 4.7: Cantidad de artículos publicados por año.

que mas publicaciones presentó sobre este tema durante los años 2010 y 2017.

Tabla 4.8: Artículos según país del autor.

Est.	U.S.A.	Italia	Gales	Suecia	Australia	Ecuador	España	Singapur	Países Bajos	Emiratos Árabes	Serbia	Canadá
S1	X											
S2								X				
S3		X										
S4	X		X	X								
S5	X											
S6					X							
S7						X	X					
S8	X											
S9				X					X			
S10									X			
S11										X		
S12	X											
S13											X	X
S14	X											

A continuación se realiza un resumen de las evidencias relacionadas a las preguntas de investigación.

Pregunta Investigación 1 (PI1): ¿Existen reportes de como diseñan software los estudiantes de grado?

Como ya fue mencionado, encontramos 14 estudios que reportan cómo

diseñan software los estudiantes de grado. A continuación se presenta un resumen de cada uno de esos estudios.

El estudio S1 es en un curso avanzado de patrones de diseño donde se busca explorar la creatividad de los estudiantes de 3er y 4to año de la carrera de ciencias de la computación de la Universidad Estatal de Connecticut Central. Se emplea una combinación de pensamiento divergente y convergente que ayuda a los estudiantes a evaluar un rango de ideas desde varias perspectivas. Además de hacer observaciones, evaluar posibles diseños y obtener conclusiones. El proceso de pensamiento divergente se repite en varias iteraciones siguiendo un camino que converge a una solución deseable. En el curso los estudiantes realizan un proyecto con tema libre, la evaluación está basada en la apropiada aplicación de patrones de diseño y en el testeo para lograr el objetivo que se plantean los estudiantes. En las primeras 4 semanas de curso los estudiantes realizan grupos de 5 o 6 personas. Una clase del curso está dedicada a Git workflow, su uso y en cómo conectar con Raspberry Pis vía VNC. En el resto de las clases se enseña patrones de diseño del libro de Gamma et al. (1994) y el pensamiento creativo enfocado en los problemas de diseño de software. Los grupos de estudiantes realizan cuatro proyectos diferentes llamados "Emotional robot", "Time lapse", "Alarm clock" y "Alarm system". Los diseños resultantes son evaluados como buenos diseños. Durante el diseño los estudiantes realizan discusiones entre los integrantes del grupo (mediante lluvia de ideas) y utilizan el pizarrón para visualizar las ideas y qué patrones de diseño utilizar en cada caso. En este estudio no utilizan diseños de terceros o herramientas de diseño para elaborar el diseño. Como lecciones aprendidas los estudiantes concuerdan que este curso les permitió aplicar los conceptos que aprendieron en clase y pensar en el diseño de todo el software que estaban creando. Además, se observó que hubo pocos cambios en el entendimiento de los principios de diseño como abstracción, encapsulamiento, pero una gran mejoría en la habilidad de identificar los patrones apropiados, y una pequeña des-mejoría en el reconocimiento y re-factorización de patrones dados en UML por parte de los estudiantes.

El estudio S2 es sobre dos *workshops* basados en técnicas de dependencia de grafos que permiten analizar el impacto sobre el diseño de software. En el primer *workshop* los estudiantes analizan y comparan diseños de software realizados por terceros basados en una técnica sobre dependencia de grafos. Y de esta forma poder brindar una nueva solución para un problema de diseño particular. El objetivo de los diseñadores es encontrar una estructura que

trabaje maximizando la modificabilidad del sistema implementado. Se estudia la habilidad de los estudiantes de aplicar la técnica de analizar el impacto de cambios de diseño dados y elegir entre opciones de diseño alternativos. Se encuentra que la mayoría de los estudiantes son capaces de hacer buenas elecciones de diseño luego de un pequeño *workshop*. Esta técnica ayuda a obtener un mejor diseño basado en modificaciones. En el segundo *workshop* se presenta una herramienta para automatizar la creación de grafos de dependencia y analizar el impacto de la misma en la enseñanza de diseño de software. Los estudiantes son capaces de aprender y aplicar el método de dependencia de grafos. El Método de Dependencia de Grafos (DGM) fue desarrollado para habilitar a los estudiantes a analizar diseño de software, construido para ser consistente con las prácticas comunes de patrones de diseño, permitir análisis comparativo de diferentes diseños y ser lo suficientemente objetivo para ser aplicado correctamente por novatos. D - Grapher es un plugin para IntelliJ que produce grafos de dependencia a partir del código, permitiendo que los usuarios seleccionen los módulos de interés y resalten las dependencias relevantes. Esta herramienta se utilizó para enseñar análisis de diseño a 60 estudiantes en un taller, donde se discutió el método, se dio entrenamiento en D - Grapher además de ejemplos y ejercicios, finalizando con una breve prueba con acceso a la herramienta D - Grapher. En este *workshop* se presentaron ejemplos de cómo crear un grafo basado en código o diagramas UML y de conjuntos de dependencia para un grafo simple de objeto de dominio. A los estudiantes se les entregó un conjunto de ejercicios en escenarios basados en prácticas de clase y errores comunes en el diseño de software que pueden realizar los novatos. Los escenarios dan práctica a los estudiantes realizando diseños con D - Grapher, comparando diseños y realizando cambios en los mismo. La utilización de la herramienta eliminó completamente los errores en las tareas que la herramienta realiza de forma automatizada y se observó una mejora en las tareas de decisión de diseño. D - Grapher soporta la mayoría de las estructuras de programación Java. En este estudio los estudiantes logran mejorar sus diseños utilizando una herramienta para realizar un diseño o una técnica sobre dependencia de grafos. Se utilizan diseños realizados por terceros.

El estudio S3 presenta una familia de cuatro experimentos controlados que estudian si el uso de diagrama de objetos UML mejora la comprensión del diseño de software al agregarlo a un diagrama de clase UML. Los experimentos en que nos centramos son PoliTo2 y UniGes1 por ser realizados por estudiante

de grado que son el foco de esta tesis. Los participantes de PoliTo2 fueron 17 estudiantes de grado. El experimento fue un ejercicio en una clase de un curso de programación orientada a objetos, que incluía una introducción a UML. Los estudiantes tenían una limitada familiaridad con UML y experiencia en desarrollo de software. Solo habían realizado cursos de introducción a la programación y estructura de datos. Los participantes de UniGel fueron 66 estudiantes de grado de un curso de Ingeniería de Software de tercer año, donde aprendían UML. Una actividad obligatoria de este curso, fue que los estudiantes armaran equipos para desarrollar un proyecto de desarrollo de software usando especificaciones basados en UML. El experimento se realizó como parte de los ejercicios de laboratorio realizados dentro del curso en el que se llevó a cabo el mismo. En ambos experimentos a los participantes se les entregaron diagramas de clase solos y diagramas de clase con diagramas de objetos. Los resultados que obtuvieron fue que el uso de diagramas de clase y objetos mejora la comprensión del diseño sólo cuando el estudiante tiene experiencia. Si el estudiante tiene poco conocimiento de UML no se recomienda agregar diagramas de objetos UML.

El estudio S4 está formado por cuatro investigaciones realizadas a estudiantes que están cursando la mitad de la carrera de computación. En el primer artículo a un grupo de estudiantes con conocimientos de programación JAVA, patrones de diseño, diagramas de clase y de secuencia, se les pide que en 50 minutos realicen un diseño de software desde cero sobre un problema de dificultad similar al problema de desarrollar una alarma de reloj. Los diseños obtenidos son analizados por dos investigadores que los clasifican utilizando la categoría definida en Eckerdal et al. (2006a) donde la categorización está basada en cuan similar es la semántica del diseño. Además se comparan los diseños con otros diseños realizados años anteriores por estudiantes de postgrado o ya recibidos. En la tabla 4.9 se resume la clasificación de la categoría definida por Eckerdal et al. (2006a).

Los resultados que obtienen son que estudiantes próximos a recibirse no son buenos en la realización de un diseño de software. La mayoría de los diseños realizados quedan clasificados dentro de la categoría de “primer paso”, sin obtenerse casi diseños en las categorías “diseño parcial” o “diseño completo”. diseños completos o parcialmente completos. A nivel de enseñanza surgen implicaciones sobre a qué temas se les debe poner más foco para que los estudiantes puedan entender y realizar mejores diseños de software. En el

Tabla 4.9: Resumen de la clasificación de las categorías (Eckerdal et al., 2006a).

Categoría	**Descripción**
Nada	Diseño con poco contenido o sin contenido comprensible.
Reafirmación	Simplemente reformula los requisitos de la descripción de la tarea.
Difuso (o Skumtomte)	Reformula los requisitos con algo de información en texto, dibujos de un GUI (del inglés Graphical User Interface) o detalles poco importantes de implementación sin descripción del diseño.
Primer paso	No solo se trabaja sobre la descripción sino sobre la visión parcial del sistema, identificando las partes pero no como se relacionan entre sí o se trabaja sobre el diseño de uno de los componentes del sistema que puede ser la GUI o la interfaz a la base de datos.
Diseño parcial	Se trabaja en la descripción de cada una de las partes y en dar una visión del sistema con las relaciones entre sus partes, donde la descripción es incompleta y la comunicación no está completamente descripta.
Diseño completo	Se logra una solución bien implementada, se da una visión del sistema entendible, describe las partes que incluye responsabilidades y explicita comunicación entre las mismas. Se usa notación formal como ser UML (del inglés Unified Modeling Language) y casos de uso.

segundo artículo se evalúan las habilidades de diseño de un grupo de estudiantes de un curso de diseño según la calidad de los diseños que realizan. El diseño que deben realizar es de una "Alarma de reloj". Los diseños son evaluados por el mismo instructor según un conjunto de criterios definidos. Al final los estudiantes deben completar una encuesta describiendo su proceso de diseño típico y cuál fue la parte más difícil de la experiencia de aprendizaje. La conclusión a la que llegan es que enseñar diseño en muchos cursos es un esfuerzo aislado donde no se cuenta con esfuerzos coordinados, estrategias pedagógicas o apoyo instructivo básico. Los estudiantes diseñarían de forma inteligente si se trabajara en conjunto para lograr una enseñanza sobre diseño de calidad. El tercer artículo es sobre un análisis fenomenográfico del fenómeno como "producir un diseño". A un grupo de 35 estudiantes de un curso obligatorio de proyecto, con conocimientos de programación JAVA, diseño de software, base de datos y estructura de datos, se les pide "producir un diseño" sobre el requisito de una "Alarma de reloj". Al conjunto de diseños realizados por los estudiantes se agrega un diseño realizado por un diseñador experto, utilizado para compararlo con los diseños realizados por los estudiantes. Los diseños según su calidad son clasificados según la categoría Skumtomte, la mayoría de los diseños se clasifican en "primer paso", "diseño parcial", "diseño completo". Los resultados del análisis fenomenografico se clasifican en seis categorías que son: (0) Diseño informal, (1) Análisis: diseño que utiliza alguna notación formal, (2) Estructura estática: diseño que usa notación formal para expresar las relaciones estáticas entre las partes, (3) Comportamiento dinámico: diseño que usa notación formal

para expresar información secuencial pero no la relaciona con la parte estática del sistema, (4) Múltiples artefactos relacionados: diseño que incluye múltiples artefactos, tanto dinámicos como estáticos, (5) Diseño de Experto: diseño que utiliza múltiples artefactos, mejora las notaciones donde las componentes son claramente presentadas. La mayoría de los diseños se clasificaron como categoría 2 porque utilizan artefactos, pero no logran relacionarlos entre sí. Además encuentran que comprender y aprender cómo hacer un buen diseño es difícil y requiere mucho esfuerzo. Es necesaria mucha experiencia y conocimiento teórico. Muchos de los cursos dictados brindan habilidades en la resolución de problemas de diseño pero resultados individuales muestran que no siempre se tiene éxito. Una forma de mejorar los resultados al producir un diseño sería contar con un plan de estudios cuidadosamente implementado donde los estudiantes se vean obligados a mejorar sus habilidades de diseño. Por otro lado, los estudiantes son conscientes de los mismo enfoques que los diseñadores profesionales y conocen las técnicas, pero también reconocen que necesitan más practica antes de producir soluciones de calidad. El cuarto artículos es sobre una investigación de como diseñan software los estudiantes de grado en grupo. Los datos se obtienen de estudiantes cursando el último semestre obligatorio de un curso titulado "metodologías ágiles". Los estudiantes en su plan de estudios incluían cursos de introducción a UML, patrones de diseño, programación Java, diseño de software, base de datos y estructuras de datos. Lo primero fue organizar 6 grupos de estudiantes. Cada grupo debía realizar en 50 minutos un diseño de software sobre el tema de la "Alarma de reloj". Estos diseños eran evaluado por otro grupo de estudiantes (sin saber a qué grupo pertenecía cada diseño) que debían determinar cuál era el mejor diseño y justificar el porqué. Dentro de los diseños había un diseño producido por el autor del artículo. Los diseños se clasificaron según la categoría de "Reafirmación", por ser diseños que solo reafirman los requisitos de la descripción de una tarea. Se confirma que estudiantes graduados o próximos a graduarse no pueden diseñar software, solo un grupo logró hacer un diseño razonable. Por otra parte, el trabajo en grupo no obtuvo mejores soluciones como se esperaba en un principio. En este estudio los estudiantes realizan diseños desde cero en papel y en algunos casos comparan con diseños realizados por terceros. Los diseños obtenidos son clasificados en diferentes categorías y tienen una calidad baja. En este estudio los estudiantes diseñan utilizando papel o el pizarrón. Utilizan diagramas UML y patrones de diseño. Como resumen podemos decir que los estudiantes diseñan

de forma “parcial”, algunos diseños se clasifican en la categoría de “primer paso”, otros en la categoría de “parcial o completa” y otros de “reafirmación”. No todos los diseños tienen la misma calidad, en la gran mayoría la calidad de los diseños es baja. En algunos de los cursos se analizan diseños realizados por terceros. Además, el tema de como enseñar diseño en los diferentes cursos es tratado en varios de los artículos.

El estudio S5 pretende mostrar como un conjunto de estudiantes de un curso de tercer año de programación JAVA mediante actividades reflexivas mejoran sus habilidades de diseñar software. Se quiere estudiar como el reflexionar sobre deficiencias en un diseño preliminar ayuda a los estudiantes a anticiparse en lo que se necesita en un diseño en proyectos futuros. Los estudiantes realizan un diseño inicial como un diagrama UML y tienen cinco oportunidades durante el semestre para mejorar su trabajo. Se espera que los estudiantes realicen un documento de una página donde se tengan los cambios entre cada diseño, explicando por qué realizaron dichos cambios y cuáles fueron las lecciones aprendidas sobre diseño como resultado del trabajo realizado. Cada uno de los diseños obtenidos en cada etapa fue evaluado por los instructores, encontrando que cada diseño mejoraba con cada modificación. Los estudiantes realizan diseños desde cero, diseñan parcialmente bien y los mejores diseños son los que tienen las últimas modificaciones. Utilizan diagramas UML y la calidad de los diseños es media.

El estudio S6 es una investigación donde se utiliza la metodología de investigación-acción (*action research*) realizada durante el dictado de un curso de fundamentos de ingeniería de software durante tres semestres. Se investiga como los estudiantes identifican conceptos equivocados de diseños de software y cómo ayudarlos a superar esos conceptos equivocados. La metodología de investigación-acción permitió mejorar los métodos de enseñanza durante los tres semestres. Se realizó la triangulación de la información obtenida a través de evaluaciones realizadas en la clase, exámenes y respuestas a cuestionarios realizados durante el curso. Cada curso está estructurado en dos partes, una con aprendizaje pasivo y otra con activo. Se cuenta con una herramienta ITS (Intelligent Tutoring System) para responden preguntas (aprendizaje pasivo) o realizar diagramas de clases con la ayuda de un asistente (aprendizaje activo). En base a las respuestas de los estudiantes se infiere en que tienen problemas y se los puede ayudar rápidamente. Esta herramienta fue diseñada para dar enseñanza personalizada a cada estudiante, está basada en responder o enseñar

temas relacionados al diseño de software y que los diagramas de clase sean cada vez más complejos. Varios estudiantes tienen problemas identificando el escenario especificado y no entienden el diseño distribuido que reduce el acoplamiento entre clases. La conclusión a la que se llega es que la herramienta de ITS es importante tanto para los estudiantes como para los docentes. Los docentes pueden analizar el progreso de cada estudiante, viendo rápidamente donde tienen problemas. Los estudiantes pueden analizar su progreso y aprender conceptos de diseño. Los estudiantes usan diagramas de UML y revisan diseños realizados por tercero. Además se tiene una herramienta con tutoriales para aprender diseño y se evalúa como se enseña diseño.

El estudio S7 es una investigación realizada a estudiantes que cursan el módulo de Diseño de Software del sexto semestre de la carrera Ingeniería Informática de la Universidad Politécnica de Madrid. Se quieren conocer las ideas primarias que tienen los estudiantes sobre diseño orientado a objeto. Esto se hizo previo al comienzo de un curso sobre este tema a los 18 estudiantes que asistieron el primer día. Como aprender diseño de software es un fenómeno complejo de estudiar, para descubrir lo que los estudiantes entendían de los conceptos de diseño se aplicó un enfoque cualitativo basado en encuestas y entrevistas. Las conclusiones obtenidas son que los estudiantes tienen ideas equivocadas sobre el diseño orientado a objeto, que el diseño no es tan importante como la codificación y que varios de los principios de diseño son ignorados por los estudiantes. A partir de este caso de estudio cualitativo los investigadores apreciaron que el diseño de software ha sido subestimado en cursos previos. Al terminar el módulo los estudiantes lograron tener mejor comprensión de los conceptos y el hacer un diseño tomó más valor. Se estudia el tema de enseñar diseño en profundidad.

El estudio S8 describe la experiencia del autor al dictar cursos de modelado y diseño para estudiantes de grado, postgrado y en la industria. Se describe brevemente el contenido de cada curso, cómo ha evolucionado la enseñanza y el enfoque de la enseñanza de "aprender haciendo", además de las lecciones aprendidas en los diferentes grupos de estudiantes. Nosotros nos centramos en la parte referente a estudiantes de grado por ser la de interés para la tesis. En el curso dictado para estudiantes de grado se sigue un modelo iterativo del ciclo de vida, de forma que los conceptos introductorios de modelado sean cubiertos en los requisitos y en la parte de diseño. Los estudiantes trabajan en grupo en las diferentes fases, resolviendo problemas del mundo real (por ejemplo,

realizar un sistema de Reserva de Hotel), desarrollando modelos de casos de uso, diagramas de clase y de secuencia. Este curso según el autor es el más desafiante dado que los estudiantes solo tienen conocimientos de programación orientada a objetos y estructura de datos, lo que hace que algunos estudiantes presenten dificultades en el entendimiento de conceptos de modelado más abstracto. Algunas de las lecciones aprendidas son qué aunque es útil dictar diferentes métodos y diseño, el dictar un método en detalle se hace necesario para que los estudiantes entiendan la complejidad del diseño de software. Que el tema de estudio sea de un proyecto real refuerza lo que los estudiantes aprenden. Los temas dictados en los cursos previos deben sincronizarse con lo que se necesita saber previo a comenzar el proyecto. El autor encuentra que los estudiantes de grado tienen más dificultades para entender los conceptos de modelado y están menos maduros, lo que hace necesario enfatizar en la explicación de los conceptos y reforzar con ejemplos. Los estudiantes realizan diseños desde cero, utilizan diagramas UML y patrones de diseño. El tema de como enseñar diseño de software es tratado en el estudio.

El estudio S9 es sobre un experimento controlado realizado a 46 estudiante de ingeniería de software donde se investiga cómo afecta el uso de ejemplos al crear diseños de software. A los estudiantes se les da una tarea para diseñar su modelo utilizando una herramienta UML, teniendo estos menos de un año de experiencia en modelado con UML. Se los dividen en dos grupos, el grupo RG de 22 estudiantes que utiliza el conjunto de ejemplos a través de un repositorio de búsqueda desde el comienzo. El grupo CG de 24 estudiantes que no utiliza el conjunto de ejemplos hasta luego de realizado el diseño y evaluar si pueden realizar mejoras en el diseño. Ambos grupos tuvieron 2 horas para realizar su modelo y luego dos días para mejorarlo utilizando el repositorio de ejemplos. Cinco expertos evaluaron los 82 modelos realizados por los estudiantes, desconociendo que modelo pertenecía a que grupo y que modelos fueron mejorados luego de utilizados los ejemplos. Los expertos evaluaron la calidad basados en los siguientes atributos de calidad: understandability, layout, extensibility, modifiability, completeness y correctness. El resultado a que llegaron es que los modelos que utilizan el conjunto de ejemplos son un 18 % mejores que los que no lo hicieron y que los diseños que no utilizan el conjunto de ejemplos desde el comienzo, pero lo hicieron para actualizar el modelo, mejoraron un 19 %. Solo 36 (20 de CG y 16 de RG) de los 46 estudiantes mejoraron o actualizaron sus modelos cuando pensaron que no eran buenos

diseños. Esto da que 27 % de los estudiantes de RG y 16 % de los estudiantes de CG estén confiados del diseño que hicieron. En una encuesta realizada a los estudiantes, la mayoría respondió que el tener un conjunto variado de ejemplos en el repositorio los ayudó en la creación de sus diseños. Los ejemplos en general les brindó a los estudiantes un punto de partida para comenzar a desarrollar sus diseños, les dio ideas desde diferentes perspectivas, los ayudó a corregir errores y les recordó revisar detalles en sus diseños. A un grupo reducido de estudiantes el tener una amplia variedad de ejemplos les confundió porque no lograron distinguir si los ejemplos eran buenos para su diseño. La conclusión final es que la utilización de ejemplos en la creación de modelos mejora la calidad de los diseños de los estudiantes. Los estudiantes realizan diseños desde cero y utilizan diseños realizados por terceros, que son los ejemplos dados. Utilizan una herramienta al diseñar diagramas UML y la calidad de los diseños en general es media. Además se considera el tema de como enseñar diseño.

El estudio S10 es sobre un experimento online donde un conjunto de estudiantes sube sus diagramas de clase, con el objetivo de revelar dificultades comunes y estrategias de modelado durante esta tarea de diseño. A 120 estudiantes de tercer año de ingeniería de software con conocimientos de diseño, UML y programación se les pide realizar un diseño de clase de un juego con el editor de diagramas WebUML. La evaluación es realizada por tres expertos con más de seis años de experiencia en diseño de software y educación. Para calificar los diseños los expertos tienen en cuenta si el diagrama refleja la realidad y como está organizado. Al observar los diseños realizados se encuentra que los estudiantes hacen mal uso de los elementos de UML, ubican de forma errónea operaciones y atributos, tienen dificultades en elegir clases y atributos, y no crean las relaciones entre clases. La conclusión a la que llegan es que los estudiantes introducen ruido al agregar elementos innecesarios y tienen dificultades para elegir entre atributos y clases como representaciones de ciertos conceptos del texto asignado. Los estudiantes utilizan una herramienta para realizar los diseños desde cero y utilizan diagramas UML. La calidad de estos diseños es baja y los estudiantes no diseñan bien.

El estudio S11 describe el desarrollo y la experiencia de enseñar en 5 semestres un curso de diseño especializado para los primeros años de la carrera de ingeniería en computación. El curso incluye temas de patrones de diseño, aplicaciones multi-thread, diseño de interfases de usuario rudimentarias y programación embebida. Los estudiantes son formados en equipos de 4 o 5 personas.

Los grupos están integrados por estudiantes con diferente conocimiento. La evaluación está basada en medidas directas e indirectas. Con las medidas directas se evalúa si los estudiantes pueden aplicar e implementar patrones de diseño de software en el sistema. Los resultados obtenidos en los 5 semestres indican que los estudiantes creen entender cómo aplicar los patrones de diseño correctamente. Sin embargo, los resultados obtenidos en el examen final muestran que el promedio de calificación para los 5 años es de 70 % y fue peor en los últimos dos años que el curso se midió. A los estudiantes con menos conocimiento no les fue bien en el examen final o en los cuestionarios, pero si en el proyecto general. Esto se debe a que los equipos están formados por estudiantes con diferente conocimiento, donde los estudiantes con mejor formación ayudan a los con menos conocimiento y el resultado del proyecto es bueno. La mayoría de los proyectos combinan de forma exitosa los patrones de diseño con los diseños orientados a objetos. Los estudiantes realizan diseños desde cero utilizando diagramas UML y patrones de diseño. La calidad de los diseños es media y el tema de como enseñar diseño esta considerado en este estudio.

El estudio S12 presenta y discute una metodología para diseño de software en el contexto de frameworks. Se explora como enseñarle a los estudiantes de software a escribir buen software. A un desarrollador se le pide que desarrolle el juego de Tic Tac Toe usando el framework Android. Se observó que siendo la primera vez que usaba este framework, lo encontró difícil de utilizar y el código generado fue complejo y difícil de entender. Para solucionar esto se explicó el framework en termino de patrones de diseño. Las instrucciones se convirtieron en un proceso de tres pasos. Primero los estudiantes deben diseñar su aplicación mediante el uso de diseño orientado a objetos. Segundo deben rediseñar la aplicación utilizando patrones de diseño. Tercero deben ajustar los patrones de diseño para que coincidan con el framework. Como es diferente desarrollar una aplicación con un framework Android que con una aplicación genérica, la metodología a ser utilizada por los estudiantes para obtener ventajas en la implementación del diseño cuenta de cinco pasos. El primero paso es extraer del problema todos los nombre y verbos, dado que los nombres son las candidatas para objetos, clases y atributos, y los verbos son los candidatos para las responsabilidades. El segundo es combinar los sustantivos en clases, esto puede requerir descartar nombres irrelevantes o nombres representando la misma cosa. El tercero es combinar los verbos extraídos en clases, instancias y responsabilidades. El cuarto es asignar responsabilidades identificando los

métodos requeridos para completar esas responsabilidades. El quinto es recorrer el escenario para asegurarse que cada uno es compatible con el método e identificar las colaboraciones entre ellos. Para realizar la evaluación de la utilidad de este método, se quiere enseñar a una clase de 30 a 35 estudiantes, donde el desarrollo de la aplicación se tratará como una tarea individual. Para explicar la metodología se va a utilizar el juego de Tic Toc Toe, y luego utilizar una aplicación más complicada. Este artículo pretende mostrar como la adaptación de un nuevo framework puede ser un desafío para los desarrolladores novatos y como la implementación de diseños estándares orientados a objetos en un nuevo marco pueden llevar a un mal diseño y mal uso del framework de referencia. Por esto se desarrolló una metodología que guíe a los estudiantes sobre cómo adaptar de buena manera las pautas genéricas que ofrece el marco a las funcionalidades específicas que se requieren en su aplicación. Esta metodología quiere ayudar a los estudiantes a usar patrones de diseño como se describe en su diseño de aplicación orientada a objetos y comparar sus diseños con el framework. Esto no solo mejoró el diseño de aplicación, sino que dio a los estudiantes una nueva visión de cómo abordar un nuevo marco y amalgamarlo en él. Esto permite que un desarrollador elabore un diseño de aplicación final que sea fácil de entender, de depurar y que el código sea más mantenible y reutilizable. En este estudio se trata principalmente el tema de como enseñar diseño de software a los estudiantes.

En el estudio S13 se propone un entorno de aprendizaje colaborativo basado en proyectos para aprender patrones de diseño de software que integran varios sistemas educativos y herramientas basadas en una base ontológica común. El paradigma en que se basa es fomentar la comprensión profunda de un tema al involucrar a los estudiantes en actividades de aprendizaje. No se les permite ser solo receptores pasivos de conocimientos sino participantes activos en las iteraciones sociales dirigidas a la construcción de conocimiento. Basados en este paradigma, técnicas de aprendizaje activo, aprendizaje basado en proyectos y colaborativo se desarrolla un ambiente integrado de aprendizaje para software DPs llamado DEPTHS (Design Patterns Teaching Help System). DEPTHS integra un Sistema de Gestión de Aprendizaje (LMS) existente, una herramienta de modelado de software y diversas herramientas de colaboración y repositorios en línea relevantes de Patrones de Diseño de software. LMS permite a los estudiantes aprender a su ritmo, proporcionándoles al mismo tiempo una variedad de actividades y recursos de aprendizaje. La herramienta de modelado

de software permite a los estudiantes experimentar el desarrollo de software basado en patrones en el contexto de problemas del mundo real. Los repositorios en línea de Patrones de Diseño de software brindan a los estudiantes una gran cantidad de recursos importantes sobre Patrones de Diseño que contienen valiosos ejemplos de Patrones de Diseño e instrucciones sobre cómo deben usarse. DEPTHS fue evaluado en un curso de 13 estudiantes de quinto año de la carrera de ingeniería en computación. Se quería determinar cuán bueno era DEPTHS en la enseñanza de patrones de diseño y si los estudiantes involucrados en un problema del mundo real podían asegurar una manera más efectiva de impartir conocimiento en el dominio de desarrollo de software. El aprendizaje colaborativo a través de proyectos ayuda a los estudiantes a aprender de sus propias experiencias de una manera que promueve la abstracción desde la experiencia, explicación de resultados y entiende las condiciones de la aplicación de los patrones de diseños en situaciones del mundo real. Además promueve la experiencia de trabajar en equipo porque se realizan grupos de 3 o 4 estudiantes para resolver el problema. La primera implementación de DEPTHS tiene una evaluación positiva con resultados convincentes que este tipo de ambientes puede significativamente contribuir en la enseñanza y aprendizaje de patrones de diseño. Los estudiantes realizan diseños desde cero y evalúan diseños realizados por terceros. Usan diagramas UML y patrones de diseño. Se trata el tema de la enseñanza de diseño de software.

En el estudio S14 se evalúan los principios, patrones y marco de proceso (P3F) en una herramienta de aprendizaje desarrollada para ayudar a los estudiantes de diseño de software a aprender y aplicar estrategias de diseño utilizadas por expertos. El principal objetivo es comprender las estrategias de los diseñadores expertos para construir modelos que apoyen el trabajo de diseñadores no expertos en campos específicos. En este estudio surgieron tres estrategias comunes. En la primera estrategia los diseñadores expertos son capaces de mantener una vista de "panorama general" y comprender el problema de diseño dentro del espacio de aplicación donde el problema es una habilidad crítica exhibida por el diseñador experto. En la segunda estrategia la habilidad del diseñador experto navega efectivamente a través del espacio del problema y el desarrollo de la solución. En la tercera estrategia empleada por diseñadores expertos la habilidad de confidencialidad para hacer decisiones de diseño aún cuando se tiene información incompleta. Los principios, patrones y marco de proceso son compuestos por cuatro elementos relacionados. Los elementos son

un conjunto fundamental de principios de diseño, estructuras de patrones de diseño, un proceso para aplicar los principios de diseño y navegar a través del diseño en desarrollo y un template para hacer decisiones de diseño informadas y confiables. El estudio piloto se llevó acabo con estudiantes de dos cursos diferentes relacionados al diseño, el primero fue con estudiantes de grado que estaban avanzados en su carrera y el segundo con estudiantes de primer año. A los estudiantes se les entregó la especificación de un problema y se les pidió que desarrollaran un diseño conceptual para el sistema deseado. La primera asignación fue realizada al comienzo del semestre para establecer una línea base de las habilidades de los estudiantes. Para estudiar como trabajan y piensan los diseñadores, los investigadores usaron una variedad de técnicas como análisis de protocolo, entrevistas y observación directa. Sin embargo, ninguna de estas técnicas fue apropiada para el estudio piloto, desde que los 46 estudiantes de las dos clases no estaban trabajando sobre el problema en un momento y lugar particular. Como la mayoría de las investigaciones, éste estudio tiene limitaciones. Los estudiantes están inscriptos en cursos para estudiantes de grado y de postgrado, la extrapolación con otros ambientes de aprendizaje no es aplicable. Por otro lado el estudio está limitado en alcance y duración, solo se resolvió un problema usando P3F. Por este motivo no se pueden sacar conclusiones sobre que tan bien podría utilizarse en una amplia gama de otro tipo de problemas de diseño. Además no se considera la calidad de los artefactos de diseño. Los resultados del estudio sugieren que el uso del framework promueve el uso de estrategias de diseño experto para resolver problemas complejos. Luego de introducir P3F se notó que la tendencia de los estudiantes de tener comportamiento como estudiantes de diseño descendió y aumentó la tendencia de exhibir comportamiento de diseñadores expertos. Los estudiantes utilizan una herramienta de diseño para aprender y diseñar. La calidad de los diseños mejora en los estudiantes que utilizan la herramienta. Además se considera el tema de como enseñar diseño.

En la tabla 4.10 se presentan las características asociadas a la pregunta de investigación “¿cómo diseñan software los estudiantes de grado?”. La ausencia de valor indica que ese punto no fue tratado en el estudio. La característica “Diseñan: Mal (M), Parcialmente Bien (P) o Bien (B)” pretende indicar como diseñan los estudiantes basados en si utilizan patrones y/o principios de diseño, diagramas UML, cuales son los conceptos de diseño que manejan, si logran plasmar los requisitos en el diseño. La característica “Calidad de los diseños:

Alto (A), Medio (M), Bajo (B)" busca encontrar la calidad de los diseños finales que realizan los estudiantes. Esta pregunta esta directamente relacionada con el punto anterior.

Tabla 4.10: Características asociadas a la preg. "cómo diseñan software los estudiantes de grado"

Carácterísticas	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Diseño realizado en papel o pizarron.	Si	Si		Si			Si	Si						
Uso de diagrama UML.	Si	Si	Si	Si	Si	Si			Si	Si	Si			
Diseñan utilizando patrones.	Si			Si							Si		Si	Si
Diseñan: Mal (M), Parcialmente Bien (P) o Bien (B)	B	P		M	P					M	P			P
Revisan diseños realizados por terceros.		Si	Si	Si		Si	Si		Si				Si	
Realizan diseños desde cero.				Si	Si			Si	Si	Si	Si		Si	
Usan herramientas al diseñar.		Si				Si			Si	Si			Si	Si
Se clasifican los diseños realizados por los estudiantes. Ej. por categorías: parcial, completa				Si										
Se trata el tema de cómo enseñar diseño y cómo afecta la forma de diseñar de los estudiantes.				Si		Si	Si	Si	Si		Si	Si	Si	Si
Calidad de los diseños: Alto (A), Medio (M), Bajo (B)	M			B	M				M	B	M			
Tutoriales para enseñar y/o aprender diseño de software.						Si								Si

De la tabla 4.10 podemos decir que la mayoría de los estudiantes de grado en los diferentes estudios utilizaron diagramas UML para modelar o realizar los diseños de software, siendo solo algunos los estudiantes que diseñaron en papel o en el pizarrón, ya sea previo a realizar el diseño en UML o como diseño final. Solo algunos estudiantes que pertenecen al universo de estudios utilizaron patrones de diseño al realizar sus diseños. Las características sobre cómo diseñan los estudiantes o la calidad de los diseños realizados por estos, están relacionadas. Los estudiantes de estos estudios diseñaron mayormente mal o parcialmente bien porque no utilizaron muchos patrones ni principios de diseños o no lograron plasmar los requisitos en el diseño. La calidad del diseño final fue medio o bajo porque no lograron manejar conceptos de diseño o llegar a un diseño final que pudiera ser enviado a un desarrollador. En estos estudios la misma proporción de estudiantes realizó diseños desde cero o revisó diseños realizados por terceros. Los diseños finales realizados por los estudiantes no fueron clasificados en diferente categorías según la calidad, los patrones o principios de diseño que utilizaron o si lograron plasmar los requisitos en el diseño. No tuvieron una clasificación similar a la definida por Eckerdal et al. (2006a). Un número no muy alto de estudiantes utilizó herramientas para diseñar y no se contó con muchos tutoriales para enseñar o aprender diseño de

software. El tema de como enseñar a diseñar y como ésta enseñanza afecta la forma de diseñar de los estudiantes se identificó como una preocupación entre los diferentes investigadores.

En la tabla 4.11 vemos la proporción de estudios donde encontramos cada una de esas características.

Tabla 4.11: Proporción de estudios que consideran cada característica

Característica	**Porcentaje**
Diseño realizado en papel o pizarron.	35,7 %
Uso de diagrama UML.	78,6 %
Diseñan utilizando patrones.	50,0 %
Diseñan: Mal (M), Parcialmente Bien (P) o Bien (B)	57,1 %
Revisan diseños realizados por terceros.	50,0 %
Realizan diseños desde cero.	50,0 %
Usan herramientas al diseñar.	42,9 %
Se clasifican los diseños realizados por los estudiantes. Ej. por categorías: parcial, completa.	7,1 %
Se trata el tema de cómo enseñar diseño y cómo afecta la forma de diseñar de los estudiantes.	64,3 %
Calidad de los diseños: Alto (A), Medio (M), Bajo (B)	57,1 %
Tutoriales para enseñar y/o aprender diseño de software	14,3 %

Pregunta de investigación 2 (PI2): ¿Cuáles son las técnicas/métodos/principios de diseño de software que utilizan los estudiantes de grado en los diseños de software realizados?

Las técnicas utilizadas por los estudiantes al realizar un diseño de software en estos 14 estudios son patrones de diseño, diagrama de clase, de secuencia y de objeto.

En el estudio S1 los estudiantes utilizan UML para plantear y resolver el problema de diseño. Identifican acertadamente patrones de diseño al realizar sus diseños desde cero y explican los beneficios del uso de patrones en cada caso. Por ejemplo incorporan combinaciones de patrones como “Factory y Strategy” o “Composite y Factory” aplicado en una variedad de contextos, o utilizan el patrón Facade. Sin embargo el reconocer patrones en diseños UML dados no es tan bien realizado. Los principios como abstracción, encapsulamiento, modularidad y polimorfismo no son muy bien comprendidos por los estudiantes.

En el estudio S2 los estudiantes tiene conocimiento de UML, se les pide que traduzcan los problemas que son diagramas de secuencia basados en código o notación UML a grafos de dependencia. Esto lo hacen utilizando la herramienta

D-Grapher o la técnica de dependencia enseñada en el workshop.

En el estudio S3 los estudiantes tienen una breve introducción a UML y realizan diagramas UML de clase y de objeto.

En el estudio S4 los estudiantes utilizan prioritariamente diagramas de clase y de secuencia al realizar sus diseños de software. Algunos estudiantes utilizan patrones de diseño como ser "Observer". En otro conjunto de diseños los estudiantes utilizan notación como seudo-código, diagramas de actividad o diagramas de estado.

En el estudio S5 los estudiantes realizan un diseño preliminar y uno final con diagramas de clase UML.

En el estudio S6 los estudiantes mediante una herramienta de tutorial, evalúan y/o crean diagramas de clase o de secuencia UML. Además conectan código con el diagrama que expresa el diseño.

En el estudio S7 del análisis de las respuestas que dieron el grupo de estudiantes a las preguntas realizadas sobre conceptos de diseño de software se encontró que los principios que los estudiantes relacionan con diseño de software son adaptabilidad y modularidad.

En el estudio S8 los estudiantes diseñan con diagramas de clase y de secuencia para la interacción de los objetos.

En el estudio S9 los estudiantes utilizan una herramienta UML, realizan diagramas de clase y evalúan si los diseños tienen los atributos de calidad modificabilidad, extensibilidad y comprensibilidad.

En el estudio S10 los estudiantes crean diagramas de clase UML.

En el estudio S11 los estudiantes utilizan patrones de diseño en el diseño de aplicaciones de software y la documentación se realiza con notación UML. Algunos de los patrones de diseño utilizados son "Adapter" y "Observer".

En el estudio S12 a través de un framework se quiere que los estudiantes utilicen patrones de diseño correctamente. Uno de los patrones utilizados es el "Model-View Controller".

En el estudio S13 se evalúa como los estudiantes aprenden patrones de diseño como "Facade", "Adapter", "Strategy", "Composite", "Visitor" y "Decorator" en el framework propuesto.

En el estudio S14 se cuenta con una herramienta de aprendizaje desarrollada para ayudar a los estudiantes a diseñar software basado en la estrategia de los diseñadores expertos. El framework "P3F" evalúa los principios y patrones de diseño para entender cómo trabajan y piensan los estudiantes de grado al

diseñar.

En la tabla 4.12 se indica que estudio presenta que técnicas, métodos y/o principios de diseño de software. El lenguaje de modelo UML es muy utilizado para realizar diseño, siendo el diagrama de clase el más utilizado, seguido por el diseño de secuencia. En el conjunto de estudios observamos que los diagramas de objeto y actividad no son muy considerados al momento de realizar un diseño de software por los estudiantes. Dentro del conjunto de principios de diseño solo son nombrados brevemente los principios de "modularidad" y "adaptabilidad" como que fueran utilizados por los estudiantes. Los patrones de diseño también son tenidos en cuenta al momento de diseñar por los estudiantes. Siendo los patrones de diseño como "Facade", "Adapter", "Strategy", "Composite", "Visitor", "Decorator", "Model-View Controller" y "Factory" los utilizados por los estudiantes de los 14 estudios.

Tabla 4.12: Técnicas/Métodos/Principios de Diseño de Software por estudio en la SLR.

Técnicas/Métodos/ Principios	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**
Diseño UML	X	X	X	X	X	X			X	X	X			
Patrones de diseño	X			X							X		X	X
Principio Modularidad							X							
Principio Adaptabilidad							X							
Seudo-código				X										
Diagrama clase			X	X	X	X		X	X	X				
Diagrama objeto			X											
Diagrama secuencia		X		X		X		X						
Diagrama actividad				X										

Pregunta de investigación 3 (PI3): ¿Con que tipo de problemas se encuentran los estudiantes de grado al realizar un diseño de software?

Son varios los problemas encontrados al realizar un diseño de software en los 14 estudios. Estos problemas los clasificamos en tres grupos que son "falta de información", "problemas con los cursos de diseño de software" y "problemas de los estudiantes con el diseño de software". En el primer grupo se consideran los estudios con los problemas que tienen los estudiantes debido a la falta de información en los requisitos. Esto puede deberse a que los requisitos estén mal formulados, tengan escasa definición o estén incompletos. Esto afecta el entendimiento del caso de estudio planteado y la resolución del

mismo. En el segundo grupo están los estudios con los problemas relacionados a los estudiantes de grado al realizar un curso de diseño de software. Estos problemas se refieren al conocimiento o forma de dictar los cursos de diseño de software por los docentes; o a no tener estrategias pedagógicas consistentes, unificadas y sincronizadas entre los diferentes cursos donde se dicta diseño de software. Además la falta de coordinación en el temario de los cursos, de material otorgando a los estudiantes o que la dinámica de los cursos no sea la mas adecuada para enseñar diseño son otros de los problemas considerados en este grupo. En el tercer grupo se consideran los problemas relacionados a la falta de formación en diseño de software que tienen los estudiantes de grado. En este grupo se tiene en cuenta la falta de experiencia para realizar diseños de software y de motivación para aprender diseño de software. Esto último debido a que el diseño de software les parece difícil de aprender. Los estudiantes tienen poca experiencia y familiaridad lo que dificulta la realización de diseños de calidad.

En la tabla 4.13 presentamos la vinculación de los estudios con los diferentes problemas con los que se encuentran los estudiantes al realizar, estudiar o aprender diseño de software que fueron detectados en los diferentes estudios. Analizando la tabla encontramos que los principales problemas detectados en los diferentes estudios son la falta de formación en diseño de software y de experiencia para realizar un diseño que tienen los estudiantes de grado. Seguido del problema relativo a la mala formulación de los cursos de diseño de software. El no dictarse oportunamente los conceptos necesarios sobre diseño de software para los proyectos que estén realizando los estudiantes es uno de los casos a considerar. Otros problemas encontrados en menor proporción en los estudios son la falta de información en los requisitos, de motivación para aprender o estudiar diseño y lo mal dictados que están algunos cursos de diseño.

Algunos estudios describen ciertas particularidades de los problemas y resulta interesante mencionarlas a continuación. Solo mencionamos los problemas y estudios que profundizan en alguno de los problemas que vimos anteriormente.

En el estudio S1 detectaron desmejoras en la forma en que los estudiantes reconocían y refactorizaban ciertos patrones en UML debido a la forma en que se abordó el tema de patrones y la falta de formación en diseño de los estudiantes.

En el estudio S3 la poca experiencia y familiaridad con los diagramas UML que tienen algunos estudiantes los pone en desventaja frente a estudiantes más

Tabla 4.13: Vinculación de problemas encontrados en los estudios.

PROBLEMAS O DIFICULTADES DETECTADOS	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
FALTA DE INFORMACION														
Por Requisitos				x	x		x							
Escasa definición, información o incompletos					x									
PROBLEMAS EN LOS CURSOS DE DISEÑO DE SOFTWARE														
Mal dictado el curso por los docentes				x										
Mal formulado el curso	x					x	x	x			x			
Falta de Material en el curso				x					x					
PROBLEMAS DE LOS ESTUDIANTES CON DS														
Falta de motivación				x	x	x								
Falta de experiencia		x	x	x	x	x		x	x					x
Falta de formación en Diseño de Software	x	x	x	x		x	x	x	x	x	x			

avanzados, y la utilización de diagramas de clase y objeto juntos no les es de utilidad para comprender mejor el diseño presentado.

En el estudio S4 la falta de análisis en los requisitos entregados a los estudiantes los dejaba en una difícil situación para comenzar a diseñar debido a la falta de experiencia que tenían los estudiantes para resolver problemas de diseño similares a estos.

En el estudio S6 el aprender diagramas UML y las relaciones entre estos diagramas les resultaba difícil.

En el estudio S8 la falta de experiencia y de formación que tienen los estudiantes en diseño dificultaba la comprensión de los conceptos de modelado más abstractos. Sin embargo, el trabajar en proyectos de software los ayudaba a entender estos conceptos en la práctica. Se indicaba que a los estudiantes de grado les era necesario explicarles los conceptos claramente y reforzarlos con ejemplos.

En el estudio S9 los ejemplos entregados a los estudiantes no estaban correctamente planteados o contenían errores, lo que dificultaba a los estudiantes encontrar material de calidad para tener como guía para realizar sus diseños.

En el estudio S11 se puso en duda que el trabajo en equipo fuese la mejor forma de aprender diseño, especialmente cuando se formaban equipos con estudiantes con diferentes niveles de conocimiento y formación en diseño de software.

Pregunta de investigación 3.1 (PI3.1): ¿Con que tipo de proble-

mas o dificultades se encuentran los estudiantes de grado al estudiar o aprender diseño de software?

Esta pregunta está muy relacionada con la pregunta PI3 pero queremos hacer incapié en los problemas que tienen o encuentran los estudiantes al estudiar o aprender diseño de software.

Estos problemas los podemos clasificar en dos grupos, el primero relacionado directamente con los problemas de la formulación o dictado de los cursos y el segundo grupo relacionado a la motivación referente al diseño de software que tienen los estudiantes de grado.

En el estudio S1 la forma en que se dictaron las clases de diseño de software mostró una pequeña falla en el reconocimiento y refactorización de patrones UML por parte de los estudiantes.

En el estudio S5 el presentar un caso de estudio con requisitos mal formulados o incompletos les dificultó el entendimiento y resolución del mismo.

En el estudio S6 los estudiantes encontraron que algunas preguntas planteadas o explicaciones dadas por los docentes no estaban bien formuladas. Las clases grandes dificultaron una adecuada retroalimentación a las dudas de los estudiantes en etapas formativas.

En el estudio S7 ciertas actividades realizadas durante el curso como ser corregir el trabajo de otros estudiantes o que sus trabajos fueran corregidos por otros compañeros, no fueron bienvenidas por los estudiantes.

En el estudio S8 se encontraron con el tema de cursos mal formulados por no enseñar un método en detalle, no tener un proyecto de la vida real para aprender haciendo y lograr sincronizar los temas de las clases con el proyecto.

En el estudio S9 se encontraron con que los ejemplos presentados en clase contenían errores.

En el estudio S11 se encontraron con el problema de cursos muy difíciles de seguir para un conjunto de estudiantes y se puso en duda que el trabajar en equipo fuese la mejor forma de aprender diseño especialmente para los estudiantes con menos conocimiento o formación en diseño.

Como resumen encontramos que uno de los problemas con los que se encuentran los estudiantes de grado al estudiar o aprender diseño de software es la forma en que se dictan los cursos de diseño de software por ser un esfuerzo aislado en lugar de esfuerzos coordinados que cuenten con estrategias pedagógicas consistentes y apoyo educativo básico. Por otra parte, los cursos de diseño de software son difíciles y no se cuenta con diferentes niveles que se

ajusten a los diferentes niveles educativos de los estudiantes, lo que hace que no todos pueden aprovechar el curso del mismo modo. No siempre se cuenta con un proyecto del mundo real para reforzar lo que los estudiantes aprenden en los cursos. El material y las clases no siempre están sincronizadas con el proyecto del curso, lo que hace que los conceptos necesarios para el proyecto no estén disponibles a tiempo. Algunos ejemplos presentados en los cursos contienen errores que dificultan el entendimiento de los estudiantes. No todos los docentes tienen dominio o experiencia en los temas que dictan relacionados al diseño. La falta de motivación de los estudiantes para aprender diseño se debe principalmente a que no es fácil de entender ni aprender.

Pregunta de investigación 4 (PI4): ¿Es evaluada la calidad de los diseños de software realizados por los estudiantes de grado?

La calidad de los diseños de software es evaluada o considerada en 11 de los 14 estudios seleccionados. Los estudios en los que se evalúa la calidad de los diseños realizados por los estudiantes de alguna forma son S1, S2, S3, S4, S5, S6, S9, S10, S11, S12 y S13.

Pregunta de investigación 4.1 (PI4.1): ¿Cómo se realiza la evaluación de la calidad del diseño de software realizado por los estudiantes de grado?

A continuación se presenta de qué forma se aborda el tema de cómo se evalúa la calidad del diseño de software realizado por los estudiante en los diferentes estudios.

En el estudio S1 se realizaron evaluaciones orales y escritas a los estudiantes. Basados en las respuestas que estos dieron se notó una mejora en el entendimiento de los diferentes patrones a utilizar y los beneficios prácticos de su uso. Esto hizo que los estudiantes mejoraran la habilidad de identificar patrones de diseño apropiados.

En el estudio S2 la evaluación se realizó mediante cuestionarios realizados a los estudiantes. Mediante la aplicación de la técnica del método de dependencia de grafos los estudiantes lograron mejorar la elección de un buen diseño o convertir un diseño malo en uno mejor.

En el estudio S3 se utilizó el número de defectos originados desde la comprensión del modelo para determinar si el uso de diagrama de objetos mejora la calidad del diseño.

En el estudio S4 la evaluación de la calidad se realizó comparando los diseños realizados por los estudiantes con otros diseños. Estos diseños de

comparación fueron realizados en otros experimentos por estudiantes de grado o ya recibidos o realizados por otro grupo de estudiantes inscriptos al mismo curso sin especificar a que grupo pertenecía cada diseño. La clasificación utilizada fueron las categorías definidas por Eckerdal et al. (2006a).

En el estudio S5 la calidad es evaluada mediante los cambios realizados entre el primer y el quinto proyecto. Los estudiantes tienen un inventario con los cambios que ocurrieron de forma de guiar las actividades reflexivas que determinan porqué los cambios son necesarios y cuáles son las lecciones aprendidas.

El estudio S6 basado en cuestionarios diseñados para marcar errores comunes de diseño, permitió que los estudiantes corrigieran las ideas equivocadas tempranamente. Esto logró mejorar el compromiso, la satisfacción y el desempeño de los estudiantes sustancialmente.

En el estudio S9 la evaluación de los diseños realizados por los estudiantes se hizo basado en seis atributos de calidad que tenían una escala del 1 al 8. Los atributos evaluados fueron "Comprensibilidad","Layour","Extensibilidad", "Modificabilidad", "Completitud" y "Correctitud". El uso de ejemplos ayudó a los estudiantes a mejorar la calidad de sus diseños y en general hizo que los estudiantes se sintieran más seguros de las decisiones tomadas sobre diseño.

En el estudio S10 la evaluación de la calidad de los diseños la realizaron 3 expertos en diseño de software y educación. Lo que evaluaron fue que tan bien el diagrama reflejaba el problema y que tan bien estaba organizado.

En el estudio S11 la evaluación de la calidad del diseño se realizó en la calificación del proyecto final.

En el estudio S12 se evaluó la calidad de los proyectos realizados, evaluando si los estudiantes utilizaban la metodología planteada y cuanto mejoraban el diseño. Para evaluar como mejoraban el diseño se los alentaba a que usaran los patrones de diseño como estaban descriptos en la aplicación de diseño orientada a objetos y lo comparaban con el framework.

En el estudio S13 la calidad de los diseños la realizan los propios estudiantes, evaluando sus diseños y los diseños de los compañeros. Esto hizo que los estudiantes reflexionaran críticamente sobre sus propias contribuciones y la de los demás, adquiriendo conocimiento de otras posibles soluciones.

En la tabla 4.10 hay dos características que están relacionadas a la calidad de los diseños, una se refiere a como diseñan los estudiantes y la otra a la calidad de los diseños realizados por los estudiantes. No todos los estudios que

evalúan la calidad publican los resultados de como es la calidad de los diseños realizados.

En el estudio S1 basado en las evaluaciones orales y escritas se dice que los estudiantes diseñan bien y la calidad de los diseños es media.

En el estudio S2 donde la evaluación de la calidad se hizo mediante cuestionarios se dice que los estudiante diseñan parcialmente bien pero no se tiene información de la calidad de los diseño realizados por estos.

En el estudio S4 mediante la comparación de los diseños con otros diseños ya realizados, se tiene que la calidad de los diseños es baja y que los estudiantes diseñan mal.

En el estudio S10 donde los diseños fueron evaluados por tres expertos, encontramos que la calidad de los diseños es baja y los estudiantes diseñan mal.

En el estudio S11 la calidad del diseño fue evaluada en el proyecto final y vemos que la calidad es media y los estudiantes diseñan parcialmente bien.

4.4. Discusiones y otros aspectos

En esta sección se realiza un análisis transversal de las diferentes preguntas de investigación y la relación entre las mismas. Se plantean los diferentes principios utilizados en los diferente estudios, cómo se evalúa la calidad de los diseños realizados por los estudiantes, qué habilidades debe tener un buen diseñador y cuáles son los problemas más comunes a los que se enfrenta un estudiante de grado al realizar un diseño.

Para lograr un buen diseño de software es importante considerar durante el desarrollo del mismo un conjunto de principios, métodos y técnicas. A continuación nombramos los principios, técnicas y métodos que encontramos en los 14 estudios analizados.

En el artículo de Steppe et al. (2017) (ID 17) el principio de modularización se presenta como el fundamental para cualquier sistema de software, que permite facilitar el desarrollo inicial y los cambios futuros. La modularización efectiva y el minimizar los efectos dominó son reconocidos como aspectos claves para desarrollar un diseño modificable. Otro de los principios nombrado en el artículo de Flores and Medinilla (2017) (ID 26) es el de ocultamiento de información, para que detalles internos de una abstracción no estén accesibles para entidades externas.

Cualidades del software como son la comprensibilidad, correctitud, extensibilidad, entendibilidad, modificabilidad (Karasneh et al., 2015) (ID 35), mantenibilidad y reusabilidad (Thevathayan and Hamilton, 2017) (ID 25) deben ser tenidas en cuenta al momento de diseñar.

Otro aspecto importante del diseño a considerar es la seguridad (Thevathayan and Hamilton, 2017) (ID 25) para así preservar la confidencialidad, integridad y disponibilidad de la información. La seguridad no debe agregarse luego que todo el sistema está construido sino que es más efectivo y eficiente analizar las alternativas y realizarlas desde la etapa de diseño.

Al realizar un diseño hay diferentes formas de documentarlo o anotarlo. La utilización de diferentes diagramas UML, como ser diagramas de clase, de secuencia y la vinculación entre los diferentes objetos (Thomas et al., 2017) (ID 104) es una de estas formas.

Los patrones de diseño de software orientado a objeto son soluciones probadas a problemas comunes, donde se puede explicar un diseño al describir los patrones utilizados. Los patrones son una forma de reutilizar el conocimiento y la experiencia de otros diseñadores (Sommerville, 2011). Los patrones no solo ayudan a entender y construir aplicaciones de la vida real, sino que agregan reusabilidad en el diseño y la implementación (Ali et al., 2011).

En los 14 estudios encontramos que la calidad de los diseños realizados por los estudiantes es evaluada de cinco formas diferentes.

Una primera forma es evaluar si el diagrama realizado utiliza diagramas de secuencia para indicar el comportamiento del sistema, y si cuenta con las clases que deben ser diseñadas. El diseño final debe poder ser tomado por otro profesional y ser implementado (Thomas et al., 2014) (ID 68).

Una segunda forma es definir categorías según sus artefactos (por ejemplo diagramas de casos de uso, GUI, diagramas de clase, de secuencia, etc.) y cómo estos artefactos se relacionan entre sí. Cada diseño es etiquetado con artefactos reconocibles de las categorías y según el entendimiento exhibido del proceso de diseño. Si un diseño es realizado por un experto incluye múltiples artefactos relacionados, los componentes son claros y están elegantemente presentados (Thomas et al., 2014) (ID 68).

Una tercera forma es que el estudiante arme un inventario con todos los cambios realizados desde la primera versión del diseño de software hasta la versión final, explicando por qué cada uno de estos cambios son necesarios de una versión a otra (Coffey, 2017) (ID 24).

Una cuarta forma es que el estudiante utilice ejemplos de diseño de diagramas UML que están dentro de un repositorio. De esta forma el estudiante puede realizar una auto-evaluación de su diseño comparándolo con los ejemplos, y realizar mejoras cuando lo consideren pertinente. El contar con ejemplos ayuda a que los estudiantes sean más confiados y seguros, a razonar por qué y cuándo el modelo es correcto, tener ideas desde diferentes perspectivas, corregir errores en sus diseños y crear modelos más simples y realizarlo más rápidamente. Además, los diseños pueden ser evaluados por un grupo de expertos que verifican que ciertos atributos de calidad como son la entendibilidad, extensibilidad, modificabilidad, completitud y correctitud estén considerados en el diseño (Karasneh et al., 2015) (ID 35).

Una quinta forma es la evaluación de los diseños por expertos qué observan que tan bien los diagramas reflejan la realidad del problema y cómo están organizados (Stikkolorum et al., 2015) (ID 52).

Por otra parte hay ciertas habilidades con las que un buen diseñador debería contar.

Un buen diseñador debe contar con ciertas "habilidades para programar", esto no se refiere a conocer la sintaxis del lenguaje de programación con el que vamos a implementar la solución, sino con la habilidad de desarrollar una solución usando la abstracción que el lenguaje nos da (Bézivin et al., 2009).

El tener un pensamiento creativo y reflexionar sobre la naturaleza de los diseños, con la noción de realizar un examen retrospectivo del trabajo favorece la realización de buenos diseños. La reflexión efectiva requiere buscar una comprensión más profunda de las fortalezas y debilidades en el enfoque del trabajo que se realiza. La reflexión es, en esencia, una actividad de aprendizaje significativo (Coffey, 2017) (ID 24).

Un diseñador junior se diferencia de un experto en que no tiene experiencia en como diseñar una construcción particular al comienzo de un nuevo proyecto, y debe aprender los mecanismos básicos utilizados para resolver el problema. Esto disminuye su conocimiento y capacidad para concebir un buen diseño preliminar (Coffey, 2017) (ID 24). El hacer un mal diseño preliminar puede hacer subestimar el tiempo de finalización del proyecto e impactar negativamente en la implementación.

Tres estrategias usadas por los diseñadores expertos deberían ser imitadas por los diseñadores con menos experiencia. En primer lugar, los diseñadores expertos son capaces de una visión sistémica o "global" de la comprensión

del problema de diseño dentro del espacio de aplicación en el que problema ocurre, ésta es una habilidad crítica que exhiben los diseñadores expertos. Para tener una imagen global del sistema, los expertos hacen un uso extensivo de la abstracción, modelos mentales y simulaciones. En segundo lugar, los expertos tienen la capacidad de navegar de manera efectiva a través del espacio del problema de la aplicación y la solución en desarrollo. Para lograr esto los diseñadores expertos tienen alta tolerancia a la ambigüedad y la incertidumbre, la capacidad de "fragmentar" elementos de un sistema y la resolución de problemas. En tercer lugar los diseñadores expertos tienen la capacidad de tomar decisiones de diseños con confianza, incluso cuando se enfrentan a información incompleta. Los estilos de toma de decisiones de los diseñadores expertos se caracterizan por una mentalidad abierta, alta tolerancia a la ambigüedad, orientación al propósito y una preferencia por la información subjetiva. Los diseñadores expertos confían en una variedad de fuentes de conocimiento cuando toman decisiones de diseño, siendo una de las más importantes los "primeros principios", no se aferran a una única metodología, son flexibles y se adaptan, usan la información de forma oportuna para redefinir la solución (Wright, 2010) (ID 112).

Un buen diseñador debe contar con la habilidad mental para tolerar ambigüedad, mantener la vista del panorama general, manejar la incertidumbre, la complejidad, tomar decisiones, pensar como parte de un equipo y en diferentes lenguajes (Hu, 2016) (ID 38). Además debe contar con conocimientos de programación, diseño de software, base de datos, estructura de datos e introducción a UML (Thomas et al., 2014) (ID 68) porque esto lo puede ayudar a realizar un buen diseño.

A continuación planteamos las dificultades encontradas cuando los estudiantes de grado realizan un diseño de software.

La comprensión de conceptos fundamentales como abstracción, encapsulación, modularidad y polimorfismo no les son fáciles (Williams and Kurkovsky, 2017) (ID 16). Los estudiantes introducen ruido por malentendidos en el texto de la tarea y tienen dificultadas para elegir las abstracciones correctas (Stikkolorum et al., 2015) (ID 52). Muchas de las dificultadas que encuentran al diseñar recaen en la faltan de entendimiento en los elementos esenciales del diseño de software orientado a objetos como abstracción, herencia, encapsulación, clases, métodos, entre otros (Flores and Medinilla, 2017) (ID 26).

Los estudiantes encuentran dificultades en cómo comenzar un diseño de

manera efectiva, en cómo considerar mentalmente las opciones de diseño y en usar los principios de diseño orientados a objeto para refactorizar un diseño (Hu, 2016) (ID 38). La falta de práctica en los estudiantes hace que se generen soluciones con una calidad no muy alta (Thomas et al., 2014) (ID 104). El aprender UML y la relación entre los diferentes diagramas UML no les es fácil (Thevathayan and Hamilton, 2017) (ID 25). Esto les presenta dificultades en el uso adecuado de los elementos UML, en la ubicación de operaciones y atributos, en el no poner todos los elementos que aparecen en la tarea y no saber diferenciar entre clases y atributos (problema de abstracción) (Stikkolorum et al., 2015) (ID 52). Además, los estudiantes presentan los artefactos como diagrama de clase o de interacción de forma independiente uno de otros, y no los logran vincular entre si (Thomas et al., 2014) (ID 104).

Los estudiantes generalmente son incapaces de identificar oportunidades para crear elementos útiles de diseño estructural, a menudo muchos elementos de diseño creados son basados en la intuición y no en análisis (Hu, 2016) (ID 38). Por otra parte, como el comportamiento del sistema es uno de los puntos más difíciles en el diseño de software para la mayoría de los estudiantes, ellos entienden que el software hace algo, pero tienen dificultades para mostrarlo formalmente (Thomas et al., 2017) (ID 104).

La falta de suficiente experiencia con el proceso de diseño no les ayuda para pensar el problema en su totalidad (Coffey, 2017) (ID 24). Por otra parte no son conscientes que el problema de software que deben resolver, puede ya existir y ser descripto con un patrón de diseño de software (Jeremic et al., 2011) (ID 102).

El principal problema que tienen los estudiantes es al entender y aprender los principios, técnicas y métodos de diseño. Esto sumado a la falta de práctica y experiencia, hace que les sea muy difícil lograr un panorama general del problema, obtener la abstracción necesaria, aplicar la metodología adecuada para cada caso y finalmente llegar a una solución de calidad. La falta de experiencia generalmente no les da la confianza y seguridad para aplicar la misma solución que ya utilizaron en otro problema si el caso lo permitiera, o realizar el proceso de diseño que la solución necesita. Las dificultades para aprender y aplicar UML genera un mal uso de los elementos de UML como son los diagramas de clase, de secuencia y sus vinculaciones. Estos problemas se mitigan si se logra generar en los estudiantes al menos algunas de las habilidades nombradas anteriormente. Una de estas habilidades puede ser fortalecer en

los estudiantes el "conocimiento de diferentes técnicas, principios y prácticas de diseño" y que sepan cuándo es posible aplicar cada una, potenciando el pensamiento creativo.

4.5. Limitaciones del estudio y amenazas a su validez

En esta sección se presentan las limitaciones y amenazas que puede tener esta investigación a su validez.

Una SRL es un método exhaustivo, sin embargo en esta revisión algunas etapas fueron llevadas a cabo por un único revisor. En la etapa de selección de estudios primarios durante la ejecución de la revisión, las selecciones las realizó el revisor 1, realizando dos selecciones diferentes con una semana de diferencia entre ellas. Por este motivo no se realizaron comprobaciones cruzadas de integridad de las búsquedas de todos los artículos ni validación de idoneidad de cada artículo. Los estudios en estado a discutir fueron revisados por dos revisores para tomar la decisión de aceptar o rechazar el estudio. En la etapa de extracción de datos de los estudios primarios el revisor 1 fue el que completo la planilla de extracción para todos los estudios, y sus dudas fueron discutidas con el revisor 2. Aunque un protocolo de revisión avala y soporta los resultados obtenidos, al ser ejecutado mayormente por un solo revisor tiene la amenaza de un sesgo alto en los resultados.

La evaluación de la calidad de los estudios primarios no fue realizada. Esta decisión fue tomada para incluir la mayor cantidad de estudios en los resultados. Si al momento de realizar la extracción de datos, se hubiera chequeado la calidad de cada estudio agregando un puntaje ponderando a la misma, esto nos permitiría identificar la calidad de los estudios utilizados en los resultados.

La búsqueda de estudios primarios fue realizada solamente en el motor de búsqueda SCOPUS, no se consideraron búsquedas manuales, ni en otros motores, ni en las referencias de los estudios primarios. Esto hace que ciertos estudios valiosos para nuestra investigación puedan no haber sido considerados.

Capítulo 5

Conclusiones y trabajos a futuro

En este capítulo se presentan las conclusiones de la tesis y los posibles trabajos a futuro.

5.1. Conclusiones

El diseño de software no es una tarea simple y no hay una única solución a un problema de diseño.

El objetivo general de esta tesis es conocer como diseñan software los estudiantes de grado y los problemas o dificultades con los que se encuentran tanto a nivel de formación como de concepto al estudiar o realizar un diseño de software. Para lograr este objetivo se realizó una reseña del estado del arte de investigaciones realizadas sobre el tema. Se planificó y realizó una SLR según la metodología definida por Kitchenham and Charters (2007) para SLR en ingeniería de software.

La SLR devolvió 18 artículos sobre el tema de análisis unificados en 14 estudios, y 5 de estos artículos fueron seleccionados durante la ejecución del piloto. Del análisis de estos artículos entendemos que el diseño de software es un tema que preocupa a los investigadores. En algunos artículos se plantea que las habilidades que debe tener un buen diseñador de software no son fáciles de obtener. Se requiere de mucho tiempo, preparación, conocimiento teórico y experiencia para adquirirlos. Para lograr formar buenos diseñadores a nivel de grado, es necesario entender cómo los estudiantes diseñan o entienden el diseño, para luego desarrollar cursos de formación en diseño que los ayude a lograr mejores resultados en los diseños que realicen. En algunos estudios se

concluye que el unificar los criterios de como enseñar diseño puede ser una forma de mejorar la formación de los estudiantes de grado. Es necesario contar con esfuerzos sincronizados para lograr estrategias pedagógica consistentes.

No todas las preguntas de investigación son respondidas en los 14 estudios. La pregunta sobre los problemas con los que se encuentran los estudiantes de grado es respondida en la mayoría de los estudios. Sin embargo, las preguntas sobre tipos de técnicas, métodos o principios o la pregunta sobre la calidad sólo tienen respuesta en algunos artículos.

En los 14 estudios analizados se identificaron un número de problemas comunes que tienen los estudiantes de grado al aprender o realizar diseños, y presentan algunas posibles soluciones o propuestas que pueden ayudar a mejorar estos problemas. Entre los problemas al aprender diseño se tiene la mala formulación de los cursos de diseño, la forma en que se dictan los cursos, la falta de material o que este no se entrega oportunamente durante el curso y la complejidad que tiene el enseñar y aprender diseño. La falta de análisis en los requisitos le genera a los estudiantes dificultad en el entendimiento del problema y esto les dificulta el comienzo del proceso de diseño. Además, de la falta de experiencia de los estudiantes y de motivación para aprender diseño. Las dificultades al aprender diseño dificulta la aplicación de los conocimientos al realizar un diseño y los casos que se plantean en general no son casos de la vida real.

Aprender o enseñar diseño de software no son tareas simples, por eso se deben crear cursos de diseño que logren captar las dudas, inquietudes y/o problemas con los que se encuentran los estudiantes al realizar un diseño, que estén formulados de forma de motivar a los estudiantes y generar instancias de diseño con ejemplos de la vida real. En uno de los artículos se plantea que el tener ejemplos de diseños ayuda a los estudiantes a realizar nuevos diseños. Los estudiantes generalmente piensen primero en codificar, lo que dificulta la tarea de pensar primero en un diseño para luego codificar.

La pregunta de investigación sobre qué tipo de técnicas, métodos o principios utilizan los estudiantes de grado fue una de las preguntas menos respondida en los diferentes artículos. Los estudiantes utilizan varias técnicas de diseño, principalmente el diagrama de clases UML. Los principios son nombrados en algunos artículos y los métodos no se tienen en cuenta. Por ser pequeña la cantidad de artículos sobre los cuales se obtuvieron los resultados, no se pueden hacer generalizaciones de que todos los estudiantes no utilizan principios o

métodos de diseño.

Este trabajo presenta algunas limitaciones en el método de investigación, porque aunque la SLR es un método exhaustivo se llevó a cabo en la mayor parte de las etapas por un único revisor (el autor de esta tesis). Por este motivo no se realizaron comprobaciones cruzadas de la integridad de las búsquedas de todos los artículos ni validación de idoneidad de cada estudio. La formalidad con que se llevó a cabo la revisión permitió validar los resultados porque están soportados y avalados por el protocolo de revisión. Sin embargo los resultados obtenidos pueden tener un sesgo alto.

Para mitigar el sesgo se aplicó la metodología de test-retest en las etapas "Proceso de selección de SLR" y "Proceso de selección por lectura completa de la SLR" del proceso de selección de la SLR. En ambas etapas se ejecutaron los pasos dos veces y se unificaron los resultados de cada ejecución. Luego se compararon los resultados obtenidos y se discutieron con otro revisor, uno de los co-tutores de la tesis.

En la ejecución del piloto, los 100 primeros artículos devueltos por la cadena de búsqueda fueron evaluados por dos revisores y se discutieron las diferencias. Sin embargo, en la ejecución de la SLR, la selección por lectura de título y resumen fue realizada mayormente por un revisor. Esto agrega subjetividad a la selección, haciendo que la mayoría de los artículos descartados fueran según la aplicación de los criterios de inclusión/exclusión de un solo revisor. Si la SLR hubiera sido llevada a cabo por más revisores, existe la posibilidad que se tuviera otro universo de artículos finales. La cadena de búsqueda del piloto fue modificada porque devolvía estudios que no estaban relacionados con el diseño de software ni con la población de estudio que son los estudiantes de grado. Somos consientes que algunos artículos valiosos para la investigación pueden no haber sido devueltos por la cadena de búsqueda final (ver tabla 3.8). Tomamos esta decisión para tener la menor cantidad de artículos descartados en la selección debido a que esta era abordada por un solo revisor. Las búsquedas de artículos primarios solo se hicieron en un motor de búsqueda SCOPUS, dejando de lado otros buscadores como ACM que podrían haber devuelto otros artículos para la misma cadena. Consideramos solo artículos publicados entre los años 2010 y 2017, aunque desde el año 1996 existen publicaciones sobre este tema.

La calidad de los 18 artículos a los que se les hizo la extracción no fue evaluada durante la SLR, se priorizó la cantidad ante la calidad, considerando

que en ingeniería de software las SLR son relativamente nuevas y asegurar la calidad de los artículos como lo sugiere Kitchenham et al. (2015) podría dejar muchos artículos fuera de la misma.

Los principales aportes de esta tesis son los siguientes:

- Un protocolo de trabajo que puede ser aplicado a un grupo mayor de revisores y de esta forma obtener nuevos resultados.
- Una planilla de selección que es fácilmente escalable para más participantes en la ejecución de un nuevo protocolo.
- Un formulario de extracción de datos para ser utilizado nuevamente al ejecutar la SLR con otros revisores.
- La realización de un SLR con todas las etapas que esta lleva para lograr el objetivo.
- Brindar una primera idea de cómo diseñan los estudiantes de grado reportando las técnicas, métodos o principios que utilizan y los problemas con los que se encuentran los estudiantes al aprender o los docentes al enseñar.

5.2. Trabajo futuro

En esta sección se presentan las posibles lineas de trabajo futuro.

La cadena de búsqueda de la SRL se ajustó para encontrar los 6 artículos que devolvió la ejecución del prototipo 1. Se podría modificar esta cadena, agregando otra parte a la cadena que filtrara por el resumen según los términos de la pregunta. Esto nos permitiría incluir artículos que pudieran ser valiosos a la investigación y que no fueron devueltos por la cadena de la SRL.

Se podría correr la cadena de búsqueda en diferentes motores de búsqueda diferentes a SCOPUS, para analizar si se incorporan nuevos estudios para ser analizados.

El protocolo definido podría ser ejecutado con un numero mayor de revisores. Esto podría evitar sesgos y enriquecer las discusiones sobre que estudios incluir/excluir, que parte de los estudios extraer y durante la síntesis de datos.

Se podría evaluar la calidad de los estudios devueltos en la selección para identificar la calidad de los estudios utilizados en los resultados.

Durante la realización de un curso de grado relacionado al diseño de software en la facultad de Ingeniería de la UDELAR, se podría implementar un piloto

para evaluar la calidad de los diseños que realizan los estudiantes de grado. Un caso podría ser similar al propuesto en uno de los artículos. En este caso el estudiante hace su diseño, lo valida y luego valida el del resto de los estudiantes del curso, esto genera instancias de discusión y enriquecimiento de todas las partes. Otro caso podría ser que al inicio y final del curso, se proponga que en 50 minutos los estudiantes realicen el diseño planteado. Esto permitiría conocer el nivel de los estudiantes al comienzo y final del curso de diseño, evaluar si se logra mejorar el nivel de los estudiantes sobre como realizar un diseño y obtener sugerencias de mejora para futuras ediciones del mismo.

En los estudios seleccionados encontramos que algunos de los principios de diseño de software especificados en el SWEBOOK (Bourque et al., 2014). son nombrados por los estudiantes. Sin embargo no los utilizan o consideran al realizar un diseño de software. Se podría investigar cuales son las razones por las cuales no los utilizan, por ejemplo si es por falta de comprensión de los principios o por pensar que no son necesarios al momento de realizar un diseño.

Además, se podría estudiar cómo se está desarrollando el curso de diseño de software y si cumple con algunas de las pautas que más se vieron en los artículos estudiados. Las pautas a analizar podrían ser:

- lograr que los principios y patrones de diseño de software sean aprendidos, entendidos y aplicados de forma correcta por los estudiantes de grado al finalizar el curso.
- que los ejemplos y/o ejercicios planteados en los cursos sean de la vida real y no tan guiados, para que cuando los estudiantes deban enfrentarse a un problema real lo puedan realizar y no decidan ir directamente a implementar el código.
- analizar las previas que debe tener este curso para que sea aprovechado de forma correcta por los estudiantes.
- realizar una prueba de nivelación previo a comenzar el curso para conocer cual es el nivel de cada estudiante, y en base a eso tomar acciones para nivelar sus conocimientos.
- intentar fomentar el pensamiento creativo durante el curso para favorecer la realización de buenos diseños.

Referencias bibliográficas

Ali, Z., Bolinger, J., Herold, M., Lynch, T., Ramanathan, J., and Ramnath, R. (2011). Teaching object-oriented software design within the context of software frameworks. In *2011 Frontiers in Education Conference (FIE)*, pages S3G–1–S3G–5.

Bourque, P., Dupuis, R., and Society, I. C. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0 ®. A Project of the IEEE Computer Society.* IEEE Computer Society Press.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583.

Budgen, D. (2003). *Software Design.* Harlow, England: Addison-Wesley.

Bézivin, J., France, R., Gogolla, M., Haugen, O., Taentzer, G., and Varro, D. (2009). Teaching modeling: Why, when, what? volume 6002, pages 55–62.

Cheng, P.-H. and Chen, L.-W. (2018). Peer learning efficacy analysis on undergraduate software design course. *Computer Applications in Engineering Education*, 26:5–16.

Coffey, J. W. (2017). A study of the use of a reflective activity to improve students' software design capabilities. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 129–134. ACM.

Cross, K. P. (2005). What do we know about students' learning and how do we know it?. Technical report, Center for Studies in Higher Education, UC Berkeley, University of California at Berkeley.

Curtis, B., Krasner, H., and Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, 31.

Eckerdal, A., Mccartney, R., Moström, J., Ratcliffe, M., and Zander, C. (2006a). Categorizing student software designs: Methods, results, and implications. *Computer Science Education*, 16:197–209.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., and Zander, C. (2006b). Can graduating students design software systems? *ACM SIGCSE Bulletin*, 38.

Fink, A. (2019). *Conducting Research Literature Reviews: From the Internet to Paper*. SAGE Publications, Inc, 5th edition.

Flores, P. and Medinilla, N. (2017). Conceptions of the students around object-oriented design: A case study. In *XII Jornadas Iberoamericanas de Ingenieria de Software e Ingenieria Del Conocimiento 2017, JIISIC 2017 - Held Jointly with the Ecuadorian Conference on Software Engineering, CEIS 2017 and the Conference on Software Engineering Applied to Control and Automation Systems, ISASCA 2017*, page 243–254.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gomaa, H. (2017). Teaching software modeling and design. In *MODELS*.

Hu, C. (2016). Can students design software?: The answer is more complex than you think. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16*, pages 199–204. ACM Press.

Hussain, S., Keung, J., and Khan, A. A. (2017). A framework for ranking of software design patterns. In *Complex, Intelligent, and Software Intensive Systems - Proceedings of the 11th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2017), Torino, Italy, July 10-12, 2017*, volume 611 of *Advances in Intelligent Systems and Computing*, pages 205–215. Springer.

Jeremic, Z., Jovanovic, J., and Gasevic, D. (2011). An environment for project-based collaborative learning of software design patterns. *International Journal of Engineering Education*, 27(1 PART 1):41–51.

Karasneh, B., Jolak, R., and Chaudron, M. R. V. (2015). Using examples for teaching software design: An experiment using a repository of uml class diagrams. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 261–268. IEEE.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, EBSE Technical Report EBSE-2007-01.

Kitchenham, B. A., Budgen, D., and Brereton, P. (2015). *Evidence-Based Software Engineering and Systematic Reviews*. Chapman and Hall/CRC.

Loftus, C., Thomas, L., and Zander, C. (2011). Can graduating students design? In *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11*, page 105. ACM Press.

McCracken, W. M. (2004). Research on learning to design software. In *Computer Science Education Research*, chapter 4, pages 155–174. Taylor & Francis.

Nandigam, J., Gudivada, V. N., and Hamou-Lhadj, A. (2008). Learning software engineering principles using open source software. In *38th Annual Frontiers in Education Conference*, pages S3H–18–S3H–23. IEEE.

Pizard, S. and Acerenza, F. (2017). Curso de EBSE. Facultad de Ingeniería - Universidad de la Reprública - Uruguay.

Pizard, S., Acerenza, F., Casella, V., Moreno, S., García, R., Lezama, J., and Vallespir, D. (2019). Conceptos de ingenieria de software basado en evidencias. Technical Report 15-08, Instituto de Computación - Facultad de Ingeniería - Universidad de la Reprública - Uruguay.

Sommerville, I. (2011). *INGENIERÍA DE SOFTWARE*. Addison-Wesley.

Stapic, Z., López, E. G., Cabot, A. G., de Marcos Ortega, L., and Strahonja, V. (2012). Performing systematic literature review in software engineering. In *Central European Conference on Information and Intelligent Systems*, page 441. Faculty of Organization and Informatics Varazdin.

Steppe, K. (2014). Teaching analysis of software designs using dependency graphs. In *2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE&T)*, pages 65–73.

Steppe, K., Chin, S., and Tuck, W. W. (2017). Improving the teaching of software design with automated modelling of syntactic dependencies. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, pages 144–151. IEEE.

Stikkolorum, D., Ho-Quang, T., Chaudron, M., and Karasneh, B. (2015). Uncovering students' common difficulties and strategies during a class diagram design process: an online experiment. In *2EduSymp@ MoDELS*, pages 29–42.

Stuurman, S., Passier, H., and Barendsen, E. (2016). Analyzing students' software redesign strategies. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pages 110–119. ACM.

Tenenberg, J. and Fincher, S. (2005). Students designing software: a multinational, multi-institutional study. *Informatics in Education*, 4:143–162.

Thevathayan, C. and Hamilton, M. (2017). Imparting software engineering design skills. In *Proceedings of the Nineteenth Australasian Computing Education Conference on - ACE '17*, pages 95–102. ACM Press.

Thomas, L., Eckerdal, A., McCartney, R., Moström, J. E., Sanders, K., and Zander, C. (2014). Graduating students' designs. In *Proceedings of the tenth annual conference on International computing education research - ICER '14*, pages 91–98. ACM Press.

Thomas, L., Zander, C., Loftus, C., and Eckerdal, A. (2017). Student software designs at the undergraduate midpoint. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 34–39. ACM.

Torchiano, M., Scanniello, G., Ricca, F., Reggio, G., and Leotta, M. (2017). Do uml object diagrams affect design comprehensibility? results from a family of four controlled experiments. *Journal of Visual Languages & Computing*, 41.

Williams, C. and Kurkovsky, S. (2017). Raspberry pi creativity: A student-driven approach to teaching software design patterns. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wessln, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

Wright, D. R. (2010). Helping students learn expert software design strategies. In *ISCA 19th International Conference on Software Engineeringand Data Engineering (SEDE-2010) June 16-18, 2010, Hilton Fisherman's Wharf, San Francisco, CA, USA*, pages 159–164. ISCA.

Zhang, H. and Babar, M. A. (2013). Systematic reviews in software engineering: An empirical investigation. *Information and Software Technology*, 55:1341–1354.

Zualkernan, I. A. (2014). A course for teaching integrated system design to computer engineering students. In *2014 IEEE Global Engineering Education Conference (EDUCON)*, pages 470–474.

ANEXOS

Anexo 1

Planilla Selección de Cadena de búsqueda 2

Esta planilla de selección tiene los 200 artículos que devuelve la cadena de búsqueda 2. En la columna ID orden Cadena 1 especifica que artículos que devolvió esta cadena ya se encontraron en la cadena de búsqueda 1.

SCOPU TITLE (design* AND (student* OR undergrad* OR teach* OR learn* OR train* OR impart* OR comprehen*)) AND (ABS ("software design*") OR ABS ({design of software}) OR ABS ({design software}) OR AUTHKEY ("software design*") OR AUTHKEY ({design of software}) OR AUTHKEY ({design software})) AND ABS (student* OR undergrad*) AND (LIMIT-TO (LANGUAGE , "English"))

Fecha 12/03/2019

							BUSQUEDA CADENA 1							BUSQUEDA CADENA 2							
								Decision Seleccion			Decision Lectura Completa				Decision Seleccion			Decision Lectura Completa				
ID orden cadena 2	Authors	Title	Year	Source tit	Abstract	Language of Original Document	ID orden cadena 1	SILVANA	PATSY	UNIFICADA y CONSENSO	SILVANA	PATSY	UNIFICADA y CONSENSO	ID orden cadena 2	PATSY - SELEC 1	PATSY - SELEC 2	UNIFICACION PLANILLA	PATSY LECTURA 1	PATSY LECTURA 2	UNIFICACION PLANILLA	SILVANA LECTURA	UNIFICACION LUEGO REUNION SUPERVISION 2
1	Luburić N., Sladi	A framework for teaching sec	2019	ACM Trar	With ever-greate	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	1	0	0	0					
2	Reddy D., Iyer S.,	Technology enhanced learnin	2018	Proceedir	Expansionist-Red	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	2	1	2	2	2	0	2	0	2
3	Fernandez-Reyes	The impact of opt-in gamificat	2018	21st ACM	An achievement-	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	3	2	1	2	1	0	2	0	2
4	Cairns D.R., Curti	Taking professional developm	2018	Interdisci	There is currentl	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	4	0	0	0					
5	Zhang H.Z., Xie C	Are their designs iterative or f	2018	Internatic	This paper invest	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	5	0	0	0					
6	Prasad P.	Developing students' cognitiv	2018	ICER 2018	Computer Scienc	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	6	1	2	2	0	0	0		
7	Lakshmi T.G.	Developing students' concept	2018	ICER 2018	Conceptual desig	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	7	1	2	2	0	0	0		
8	Jamal N.A.A., Ali	Comparative critiquing and ex	2018	2017 IEEE	Software architec	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	8	0	0	0					
9	Stikkolorum D.R.,	Evaluating didactic approache	2018	ACM Inte	Teaching assistar	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	9	1	1	1	2	2	2	0	2
10	Cristie V., Joyce S	GHShot: 3D design versioning	2018	Conferen	During the proce	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	10	0	0	0					
11	Magana A.J., Sea	Fostering cooperative learnin	2018	Journal o	Agile methods su	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	11	2	1	2	2	0	2	0	2
12	Cheng P.-H., Che	Peer learning efficacy analysis	2018	Compute	Knowledge sharii	English	3	1	1	1	0	0	0	12	0	0	0					
13	Wu Y.	Research on the design of yog	2018	Journal o	As a major impor	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	13	0	0	0					
14	Kalmpourtzis G.,	The role of adults in giving an	2018	Advances	Recent technolo	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	14	0	0	0					
15	Li Y.	Teaching innovation of enviro	2017	Boletin Te	This study is teac	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	15	2	2	2	0	0	0		
16	Williams C., Kurk	Raspberry Pi creativity: A stud	2017	Proceedir	One of the prima	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	16	2	1	2	1	1	1		1
17	Steppe K., Chin S	Improving the Teaching of Sof	2017	Proceedir	We present the u	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	17	1	1	1	1	1	1		1
18	Lim E.W.C.	A design software to facilitate	2017	Educatior	A design softwar	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	18	0	0	0					
19	Torchiano M., Sc	Do UML object diagrams affec	2017	Journal o	Objective: The m	English	31	1	1	1	1	1	1	19	1	1	1	1	1	1		1
20	Thomas L., Zande	Student software designs at th	2017	Annual Co	We replicate a st	English	37	1	1	1	1	1	1	20	1	1	1	1	1	1		1
21	Kecskemety K.M.	Examining software design pr	2017	ASEE Ann	Courses that tea	English	41	1	1	1	0	0	0	21	0	0	0					
22	Sykamiotis G., Ch	Extraction and presentation o	2017	IEEE Glob	Educational data	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	22	0	0	0					
23	McDaniel R., Fan	Creative content managemen	2017	IEEE Tran	Background: This	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	23	0	0	0					
24	Coffey J.W.	A study of the use of a reflecti	2017	Proceedir	This paper conta	English	53	1	1	1	1	2	1	24	1	1	1	1	1	1		1
25	Thevathayan C., I	Imparting software engineerir	2017	ACM Inte	Teaching softwar	English	62	2	1	1	1	2	1	25	1	1	1	1	1	1		1
26	Flores P., Medini	Conceptions of the students a	2017	XII Jornac	Software design	English	72	1	1	1	no lo tene	1	#VALUE!	26	1	2	2	1	1	1		1
27	Robson D.	Applying pedagogy to the des	2017	Proceedir	In this paper, we	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	27	2	1	2	0	0	0		
28	Orzeszek D., Kop	Beyond participatory design: T	2017	CEUR Wo	Population aging	English	77	0	0	0		#N/A	#N/A	28	0	0	0					
29	Fan C., Ren Z., Yi	Practical research on the teac	2017	Proceedir	Optical design, to	English	76	0	0	0		#N/A	#N/A	29	0	0	0					
30	Gomaa H.	Teaching software modeling a	2017	CEUR Wo	This paper descri	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	30	1	1	1	1	1	1		1
31	Steingrimsson B.	Ecosystem for engineering de	2017	Internatic	Design is a huma	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	31	1	2	2	0	0	0		
32	Roy D.	Task-based EFL language teacl	2017	Cogent Ec	Task-based langu	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	32	0	0	0					
33	Fronza I., El Ioini	Teaching software design eng	2016	SIGITE 20	Over the years a	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	33	2	2	2	0	0	0		
34	Carpenter M.S., \	Improved student engagemer	2016	ASEE Ann	The implementat	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	34	2	1	2	0	0	0		
35	Karasneh B., Jola	Using examples for teaching s	2016	Proceedir	Context: This res	English	121	#N/A	#N/A	#N/A		#N/A	#N/A	35	1	2	2	1	1	1		1

36	Xu L., He W.	The Office Software Learning	2016	MATEC W	Fragmented lear	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	36	0	0	0					
37	Carr J.	What are the benefi ts of expe	2016	Structura	It is apparent tha	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	37	0	0	0					
38	Hu C.	Can students design software	2016	SIGCSE 2(	In this paper, we	English	128	#N/A	#N/A	#N/A		#N/A	#N/A	38	1	1	1	1	1	1		1
39	Takahashi A., Kas	Design of advanced active and	2016	Proceedir	In this paper, we	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	39	0	0	0					
40	Yang C.Y., Hu Y.,	Is Spatial Ability improved? Cr	2016	44th Ann	Spatial ability has	English	148	#N/A	#N/A	#N/A		#N/A	#N/A	40	0	0	0					
41	Markiewicz P.	Smart project, building and cit	2016	World Tra	Current building	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	41	0	0	0					
42	Hilton E., Li W., N	The development and effects	2016	Proceedir	As Computer-Aid	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	42	0	0	0					
43	Goldstein M.H., F	Assessing idea fluency throug	2015	Proceedir	Engineering desi	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	43	2	2	2	0	0	0		
44	Gravino C., Scanr	Source-code comprehension t	2015	Journal o	Objective: The m	English	187	#N/A	#N/A	#N/A		#N/A	#N/A	44	2	1	2	2	1	2	0	2
45	Bertacchini F., Bi	Designing an educational mus	2015	Progress	This work aims a	English	188	#N/A	#N/A	#N/A		#N/A	#N/A	45	0	0	0					
46	Benson B., Mealy	Experience with teaching digit	2015	Proceedir	Advances in tech	English	193	#N/A	#N/A	#N/A		#N/A	#N/A	46	2	1	2	0	0	0		
47	Vazquez M.M., R	Analysis of self-regulated lear	2015	Proceedir	In all the projects	English	190	#N/A	#N/A	#N/A		#N/A	#N/A	47	0	0	0					
48	Tao Y., Liu G., Mc	Just-in-Time-Teaching experie	2015	IEEE Glob	Teaching design	English	200	#N/A	#N/A	#N/A		#N/A	#N/A	48	2	1	2	2	2	2	0	2
49	Campbell J., Tafli	An experience report: Using n	2015	SIGCSE 2(	We report on ou	English	196	#N/A	#N/A	#N/A		#N/A	#N/A	49	2	1	2	2	1	2	0	2
50	Dan L., Erhu H., Y	Teaching and practice innovat	2015	Internatic	This study aims a	English	208	#N/A	#N/A	#N/A		#N/A	#N/A	50	0	0	0					
51	Wendell D., Guar	2.00GoKart - Using electric go	2015	ASEE Ann	This paper introd	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	51	0	0	0					
52	Stikkolorum D.R.,	Uncovering students' commor	2015	CEUR Wo	This paper descri	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	52	1	1	1	1	1	1		1
53	Verner I., Merksa	Digital design and 3D printing	2015	Procedia	This paper report	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	53	0	0	0					
54	Marshall M., Mo	WIMP51 processor: Envisionir	2015	ASEE Ann	The Digital Syste	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	54	0	0	0					
55	Flores P., Medini	What do software design stud	2014	ACM Inte	The information	English	240	#N/A	#N/A	#N/A		#N/A	#N/A	55	1	2	2	0	0	0		
56	Luo Q.	Software design of college stu	2014	Applied M	The hardware str	English	253	#N/A	#N/A	#N/A		#N/A	#N/A	56	0	0	0					
57	Steppe K.	Teaching analysis of software	2014	2014 IEEE	We present the u	English	290	#N/A	#N/A	#N/A		#N/A	#N/A	57	1	1	1	1	1	1		1
58	Danielak B.A., Dc	Studying students' early-stage	2014	Proceedir	This paper descri	English	260	#N/A	#N/A	#N/A		#N/A	#N/A	58	2	2	2	0	0	0		
59	Whittle J., Bull C.	Teaching in a software design	2014	CEUR Wo	This paper reflec	English	263	#N/A	#N/A	#N/A		#N/A	#N/A	59	0	0	0					
60	Shelley T., Lyons	Information-building applicati	2014	Conferen	The propagation	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	60	0	0	0					
61	Koehler N.A., Thc	Designing online software for	2014	Educatior	Our case study is	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	61	0	0	0					
62	Zualkernan I.A.	A course for teaching integrat	2014	IEEE Glob	While cornerstor	English	289	#N/A	#N/A	#N/A		#N/A	#N/A	62	1	1	1	1	1	1		1
63	Seneca E.	Analyzing the effects of conte	2014	Internatic	The adoption of	English	291	#N/A	#N/A	#N/A		#N/A	#N/A	63	0	0	0					
64	Brown R.A., Adar	Teaching parallel design patte	2014	SIGCSE 2(	The industry shif	English	302	#N/A	#N/A	#N/A		#N/A	#N/A	64	2	2	2	0	0	0		
65	Luo Q.	Design of college students per	2014	WIT Tran	Based on the ind	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	65	2	2	2	0	0	0		
66	Lai C.-H., Huang S	Task-based learning in vocatic	2014	Proceedir	Machinery begin	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	66	0	0	0					
67	Wang C.X., Yang	Design and development of st	2014	Applied M	With the develop	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	67	0	0	0					
68	Thomas L., Ecker	Graduating students' designs	2014	ICER 2014	We expand upon	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	68	1	1	1	1	1	1		1
69	Gimenes I.M.S., I	Learning design for software e	2014	CSEDU 2(	This paper prese	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	69	2	2	2	0	0	0		
70	Millen L., Cobb S.	A collaborative virtual environ	2014	Virtual Re	Young students v	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	70	0	0	0					
71	Zifchock R., Gran	Computer aided design: Learn	2014	Compute	In the Computer	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	71	0	0	0					
72	Tamburri D.A., Ri	Teaching software design with	2013	Software	Software designe	English	361	#N/A	#N/A	#N/A		#N/A	#N/A	72	1	1	1	2	1	2	0	2
73	Nesbitt D.	Student evaluation of CALL to	2013	Compute	This article discu	English	370	#N/A	#N/A	#N/A		#N/A	#N/A	73	0	0	0					
74	Crawford G., Bye	Computer aided design: Learn	2013	ASEE Ann	In the Computer	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	74	0	0	0					
75	Buelin-Biesecker	Can pedagogical strategies aff	2013	ASEE Ann	Linear models fo	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	75	2	2	2	0	0	0		
76	Mondragon-Torr	A comprehensive embedded s	2013	2013 IEEE	In this paper we	English	372	#N/A	#N/A	#N/A		#N/A	#N/A	76	1	1	1	0	0	0		
77	Sheikh Abdul Azi	The design and usability testir	2013	Lecture N	Norman claims th	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	77	2	2	2	0	0	0		
78	Enciso M., Soler I	Teaching database design: A r	2013	IEEE Glob	Software design	English	376	#N/A	#N/A	#N/A		#N/A	#N/A	78	0	0	0					
79	Doddannara L.S.,	Exploring the relationships be	2013	Lecture N	Recent research	English	378	#N/A	#N/A	#N/A		#N/A	#N/A	79	0	0	0					

80	Hoi Y., Fong S.	Design of all-in-one distribute	2013	Journal o	The early explore	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	80	0	0	0						
81	Rahman N.A., Hu	Integrating computer applicat	2013	Educatior	The current studi	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	81	0	0	0						
82	Hu C.	The nature of software design	2013	ACM Inro	Design plays a cr	English	394	#N/A	#N/A	#N/A		#N/A	#N/A	82	1	1	1	2	1	2	0	2	
83	López-Alba E., Ló	Teaching methodology based	2013	Mechanis	Engineering educ	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	83	0	0	0						
84	Oner D., Adadan	Design of a customized web-b	2012	Proceedir	This paper descri	English	418	#N/A	#N/A	#N/A		#N/A	#N/A	84	0	0	0						
85	Sun Y., Liu X.	Educating students by "real-w	2012	ICCSE 201	Introductory Soft	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	85	2	2	2	2	0	2	0	2	
86	Arenas-Márquez	Interactive learning in operati	2012	Internatic	Purpose: The pur	English	439	#N/A	#N/A	#N/A		#N/A	#N/A	86	2	2	2	0	0	0			
87	Castro M.	Highway design software as su	2012	Compute	Traditionally, pra	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	87	0	0	0						
88	Mei-Shan J., Chai	The designment of student inl	2012	2012 2nd	The amount of hi	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	88	0	0	0						
89	Nikolaraizi M., Vo	The design of a software to er	2012	Educatior	In this paper we	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	89	0	0	0						
90	Mahmoodi H., M	Hands-on teaching of embedd	2012	Proceedir	Modern embedd	English	461	#N/A	#N/A	#N/A		#N/A	#N/A	90	0	0	0						
91	Ben Youssef B., B	Teaching first-year interdiscip	2012	Proceedir	Many technology	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	91	2	2	2	0	0	0			
92	Hsieh P.-H., Cher	Effects of reflective thinking ir	2012	Turkish O	The purpose of tl	English	462	#N/A	#N/A	#N/A		#N/A	#N/A	92	2	2	2	2	2	2	0	2	
93	Skov M.B., Stage	Training software developers	2012	Behaviou	Many efforts to i	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	93	2	2	2	0	0	0			
94	Chance S.M.	Rubric for assessing epistemo	2012	Proceedir	There is an exten	English	489	#N/A	#N/A	#N/A		#N/A	#N/A	94	1	1	1	0	0	0			
95	Dzan W.-Y., Lou S	A teaching mode on network	2012	Worksho	Network cloud is	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	95	0	0	0						
96	Stolley K.	Teaching sustainable methods	2011	IEEE Inter	This workshop pr	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	96	0	0	0						
97	Balog P., Kramer	Dedicated infrastructure for a	2011	Proceedir	According to labo	English	501	#N/A	#N/A	#N/A		#N/A	#N/A	97	2	2	2	0	0	0			
98	Ali Z., Bolinger J.,	Teaching object-oriented soft	2011	Proceedir	Object-oriented s	English	495	#N/A	#N/A	#N/A		#N/A	#N/A	98	1	1	1	1	1	1		1	
99	Amoussou G.-A.,	Teaching and assessing an inte	2011	Proceedir	The role of desig	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	99	2	2	2	1	0	2	0	2	
100	Hu J.	Exploration of a new approach	2011	ICCSE 201	This paper prese	English	524	#N/A	#N/A	#N/A		#N/A	#N/A	100	0	0	0						
101	Wessa P., de Ryc	Content-based VLE designs im	2011	PLoS ONE	Background: We	English	525	#N/A	#N/A	#N/A		#N/A	#N/A	101	0	0	0						
102	Jeremic Z., Jovan	An environment for project-ba	2011	Internatic	Software engine	English	553	#N/A	#N/A	#N/A		#N/A	#N/A	102	1	1	1	1	1	1		1	
103	Shih C.-C., Hwang	Learning embedded software	2011	IEEE Tran	The need for pro	English	556	#N/A	#N/A	#N/A		#N/A	#N/A	103	1	1	1	0	0	0			
104	Loftus C., Thoma	Can graduating students desig	2011	SIGCSE'1	In this paper we	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	104	1	1	1	1	1	1		1	
105	Niu H.	The evaluation of college stud	2011	2011 Inte	To increase the q	English	558	#N/A	#N/A	#N/A		#N/A	#N/A	105	0	0	0						
106	Hundhausen C., A	The design and experimental e	2011	Journal o	BACKGROUND In	English	570	#N/A	#N/A	#N/A		#N/A	#N/A	106	0	0	0						
107	Triantafyllakos G	Designing educational softwar	2011	Compute	In this paper, we	English	565	#N/A	#N/A	#N/A		#N/A	#N/A	107	0	0	0						
108	Lundh P., Cheng	Using design personas to infor	2010	Learning	Design personas	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	108	0	0	0						
109	Peng T., Sun L., F	Teaching reform research and	2010	Proceedir	This paper prese	English	580	#N/A	#N/A	#N/A		#N/A	#N/A	109	2	2	2	0	0	0			
110	Triantafyllakos G	Designing E-learning applicati	2010	Affective,	In years past, ma	English	595	#N/A	#N/A	#N/A		#N/A	#N/A	110	2	2	2	0	0	0			
111	Abrami P.C., Sava	The learning toolkit: The desig	2010	Design ar	In this chapter th	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	111	0	0	0						
112	Wright D.R.	Helping students learn expert	2010	19th Inte	Computer scienc	English	583	#N/A	#N/A	#N/A		#N/A	#N/A	112	1	1	1	1	1	1		1	
113	Dalamarinis P.	An easy to use well design sof	2010	Proceedir	One of the bigge	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	113	2	2	2	0	0	0			
114	Dou Z., Wang Y.,	Teaching software design of p	2010	ICENT 20	In order to make	English	615	#N/A	#N/A	#N/A		#N/A	#N/A	114	0	0	0						
115	Annetta L.A., Hol	Measuring student perceptior	2010	Internatic	As educational ga	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	115	0	0	0						
116	Dorado R., López	Teaching turbochargers via co	2010	Internatic	A computer 3D m	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	116	0	0	0						
117	Zoltowski C., Oak	Teaching human-centered des	2010	ASEE Ann	Effectively teachi	English	644	#N/A	#N/A	#N/A		#N/A	#N/A	117	2	2	2	0	0	0			
118	Hassan F., Vemu	Introducing hybrid design app	2010	ASEE Ann	Nowadays, embe	English	647	#N/A	#N/A	#N/A		#N/A	#N/A	118	2	2	2	2	1	2	0	2	
119	Seniow K., Nefcy	Representations of student m	2010	ASEE Ann	The undergradua	English	652	#N/A	#N/A	#N/A		#N/A	#N/A	119	0	0	0						
120	Gómez-Martín M	Teaching design patterns usin	2009	Proceedir	Object-oriented s	English	666	#N/A	#N/A	#N/A		#N/A	#N/A	120									
121	Jia Y., Tao Y.	Teaching software design usir	2009	ITNG 200	Modeling is a fur	English	660	#N/A	#N/A	#N/A		#N/A	#N/A	121									
122	Johnson M.C., Vil	Undergraduate dual-core prot	2009	2009 IEEE	Multi-core techn	English	690	#N/A	#N/A	#N/A		#N/A	#N/A	122									
123	Alers S., Hu J.	AdMoVeo: A robotic platform	2009	Lecture N	Designing intellig	English	703	#N/A	#N/A	#N/A		#N/A	#N/A	123									

124	Connolly R.W.	Complecto mutatio: Teaching	2008	Proceedir	Students can bes	English	735	#N/A	#N/A	#N/A		#N/A	#N/A	124								
125	Jalil M.A., Noah S	Assisting students in applying	2008	Proceedir	Design patterns r	English	737	#N/A	#N/A	#N/A		#N/A	#N/A	125								
126	Connolly R.	Complecto mutatio: Teaching	2008	Proceedir	This paper argue	English	735	#N/A	#N/A	#N/A		#N/A	#N/A	126								
127	De Rezende J.L.,	Using pbl focused on knowled	2008	MCCSIS'0	This article prese	English	741	#N/A	#N/A	#N/A		#N/A	#N/A	127								
128	Skov M.B., Stage	Direct integration: Training so	2008	CEUR Wo	Many improvem	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	128								
129	Chen J., Liu J.-H.,	An intensive curriculum on en	2008	Proceedir	This paper prese	English	743	#N/A	#N/A	#N/A		#N/A	#N/A	129								
130	Dietrich J., Kemp	Tool support for teaching desi	2008	Proceedir	Design patterns a	English	782	#N/A	#N/A	#N/A		#N/A	#N/A	130								
131	White T.	Debugging an artifact, instrum	2008	Internatic	This article explo	English	793	#N/A	#N/A	#N/A		#N/A	#N/A	131								
132	Eckerdal A., McC	Can graduating students desig	2007	Proceedir	This paper exami	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	132								
133	Jiménez-Díaz G.,	Pass the ball: Game-based lea	2007	Lecture N	Based on our exp	English	809	#N/A	#N/A	#N/A		#N/A	#N/A	133								
134	Wright P.N.H., Hu	Approaches developed to sup	2007	RINA - Int	The background	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	134								
135	Cowling A.J.	Stages in teaching software de	2007	Software	This paper descri	English	839	#N/A	#N/A	#N/A		#N/A	#N/A	135								
136	Schwartzman L.	Student defensiveness as a th	2007	Informati	Reflective practic	English	854	#N/A	#N/A	#N/A		#N/A	#N/A	136								
137	Jiménez-Díaz G.,	Using role-play virtual environ	2007	Europear	Object-oriented s	English	857	#N/A	#N/A	#N/A		#N/A	#N/A	137								
138	Code J.R., MacAll	Self-regulated learning, motiv	2006	Proceedir	Instructional des	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	138								
139	Brown C.A.	Teaching axiomatic design to	2006	Journal o	Axiomatic design	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	139								
140	Wang S., Yilmaz I	A strategy and tool support to	2006	Internatic	Proper design an	English	912	#N/A	#N/A	#N/A		#N/A	#N/A	140								
141	Cunningham H.C	Using classic problems to teac	2006	Science o	All programmers	English	919	#N/A	#N/A	#N/A		#N/A	#N/A	141								
142	Volk K.	Assessment practices that pro	2006	How Asse	In an informatior	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	142								
143	Blaha K., Monge	Do students recognize ambigu	2005	Proceedir	Successful softwa	English	926	#N/A	#N/A	#N/A		#N/A	#N/A	143								
144	Sims-Knight J.E.,	A simulation task to assess stu	2005	Proceedir	Research has sho	English	948	#N/A	#N/A	#N/A		#N/A	#N/A	144								
145	Navarro E.O., Ho	Scaling up: How thirty-two stu	2005	Proceedir	Virtually all softw	English	927	#N/A	#N/A	#N/A		#N/A	#N/A	145								
146	Hill J.H., Gokhale	Visual OS: Design and implem	2005	Proceedir	An operating syst	English	944	#N/A	#N/A	#N/A		#N/A	#N/A	146								
147	Jarzabek S., Eng I	Teaching an advanced design,	2005	Proceedir	Students learn al	English	934	#N/A	#N/A	#N/A		#N/A	#N/A	147								
148	Jarzabek S., Eng I	Teaching an advanced design,	2005	Software	Students learn al	English	934	#N/A	#N/A	#N/A		#N/A	#N/A	148								
149	Navarro E.O., Vai	Scaling up: How thirty-two stu	2005	Software	Virtually all softw	English	927	#N/A	#N/A	#N/A		#N/A	#N/A	149								
150	Fernandez E.B., F	Using UML and security patte	2005	ASEE Ann	Our introductory	English	969	#N/A	#N/A	#N/A		#N/A	#N/A	150								
151	Dornan T., Lee C.	Rapid application design of an	2005	Compute	The aim was to fi	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	151								
152	Warren I.	Teaching Patterns and Softwa	2005	Conferen	In this paper we	English	979	#N/A	#N/A	#N/A		#N/A	#N/A	152								
153	Cooley W.L.	Individual student assessment	2004	Proceedir	In our senior des	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	153								
154	Wick M.R., Steve	Seven design rules for teachin	2004	Proceedir	Because encapsu	English	1029	#N/A	#N/A	#N/A		#N/A	#N/A	154								
155	Wick M.R., Steve	Seven design rules for teachin	2004	SIGCSE Bu	Because encapsu	English	1029	#N/A	#N/A	#N/A		#N/A	#N/A	155								
156	Penuel W.R., Yar	Designing handheld software	2004	Journal o	Since 2002, Proje	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	156								
157	Huang T., Swente	Teaching Undergraduate Softv	2003	Proceedir	Most large resea	English	1044	#N/A	#N/A	#N/A		#N/A	#N/A	157								
158	Carrington D., Kir	Teaching software design with	2003	Proceedir	When an introdu	English	1043	#N/A	#N/A	#N/A		#N/A	#N/A	158								
159	Lamm E.	Booch's Ada vs. Liskov's Java:	2003	Lecture N	We study two te	English	1045	#N/A	#N/A	#N/A		#N/A	#N/A	159								
160	Sprunt B.	A novel racetrack platform for	2003	ASEE Ann	The changes beir	English	1049	#N/A	#N/A	#N/A		#N/A	#N/A	160								
161	Hankley W.	On teaching software architec	2003	ASEE Ann	This paper descri	English	1057	#N/A	#N/A	#N/A		#N/A	#N/A	161								
162	Huddleston D.H.,	Using commercial software to	2003	Compute	Recent advances	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	162								
163	[No author name	Student designers	2003	Folding C	Winners of the Si	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	163								
164	Chang N., Lee I.	Embedded system hardware c	2003	Proceedir	Qualified softwar	English	1075	#N/A	#N/A	#N/A		#N/A	#N/A	164								
165	Carrington D., Kir	Teaching software design with	2003	Proceedir	When an introdu	English	1043	#N/A	#N/A	#N/A		#N/A	#N/A	165								
166	Frezza S.	Integrating testing and design	2002	Proceedir	Software testing	English	1083	#N/A	#N/A	#N/A		#N/A	#N/A	166								
167	Lessard R.A.	A Lego-based soccer-playing r	2002	ASEE Ann	The competition	English	1085	#N/A	#N/A	#N/A		#N/A	#N/A	167								

168	Weinberg J.B., St	A laboratory experience for te	2002	ASEE Ann	The inclusion of a	English	1089	#N/A	#N/A	#N/A		#N/A	#N/A	168											
169	Cottrell D.S.	Computer aided design for int	2002	Proceedir	This paper descri	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	169											
170	Altman A.	A computational tool to aid in	2002	40th AIAA	Traditionally in a	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	170											
171	Weinberg J.B., St	Participatory design in a huma	2002	SIGCSE B	Empirical eviden	English	1096	#N/A	#N/A	#N/A		#N/A	#N/A	171											
172	Armarego J.	Advanced software design: A	2002	Software	The restructuring	English	1100	#N/A	#N/A	#N/A		#N/A	#N/A	172											
173	Waldo D.J.	Real-time DSP system design	2002	Proceedir	This paper descri	English	1105	#N/A	#N/A	#N/A		#N/A	#N/A	173											
174	Choi S.-H., Cairnc	Using interactive multimedia	2001	Proceedir	Interactive multi	English	1117	#N/A	#N/A	#N/A		#N/A	#N/A	174											
175	Chung G.K.W.K.,	The impact of a simulation-ba	2001	IEEE Tran	This study exami	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	175											
176	Rasala R., Raab J.	Java power tools: Model softv	2001	SIGCSE B	The Java Power T	English	1136	#N/A	#N/A	#N/A		#N/A	#N/A	176											
177	Kafai Y.B., Ching	Affordances of Collaborative	2001	Journal o	Although educati	English	1143	#N/A	#N/A	#N/A		#N/A	#N/A	177											
178	Svane T., Aderklo	Knowledge by user demand ai	2001	Proceedir	This paper report	English	1141	#N/A	#N/A	#N/A		#N/A	#N/A	178											
179	Svane T., Aderklo	Knowledge by user demand ai	2001	Proceedir	This paper report	English	1141	#N/A	#N/A	#N/A		#N/A	#N/A	179											
180	Murtagh J.L., Har	Teaching a real-world softwar	2000	ASEE Ann	In this paper, we	English	1163	#N/A	#N/A	#N/A		#N/A	#N/A	180											
181	Symans M.D.	Introducing middle school stu	2000	Journal o	In an effort to m	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	181											
182	Stern J.	The design of learning softwa	2000	Journal o	This paper descri	English	1162	#N/A	#N/A	#N/A		#N/A	#N/A	182											
183	Naveda J.Fernan	Teaching architectural design	1999	Proceedir	In the fall of 199	English	1186	#N/A	#N/A	#N/A		#N/A	#N/A	183											
184	Carrington David	Teaching software design and	1998	Proceedir	This paper descri	English	1217	#N/A	#N/A	#N/A		#N/A	#N/A	184											
185	Nicaise M.	Cognitive research, learning t	1998	Journal o	The idea that the	English	1220	#N/A	#N/A	#N/A		#N/A	#N/A	185											
186	Jackson Shari L.,	Design of guided learner-adap	1998	Conferen	The learner-cent	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	186											
187	Budgen D.	Is teaching software design a	1995	Lecture N	The creative act	English	1316	#N/A	#N/A	#N/A		#N/A	#N/A	187											
188	Park S.	Implications of learning strate	1995	Journal o	This study preser	English	1314	#N/A	#N/A	#N/A		#N/A	#N/A	188											
189	Meyer Karl F., Re	Tool but not a crutch: using de	1995	Computir	The use of desigr	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	189											
190	Horton T.B.	Using commercial CASE envirc	1994	Lecture N	This paper descri	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	190											
191	Huang Garng M.,	Development of an undergrad	1993	American	An undergraduat	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	191											
192	Sims-Knight J.E.,	Teaching Object-Oriented Des	1993	Compute	This project is de	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	192											
193	Hohmann L., Guz	SODA: A computer-aided desi	1992	Lecture N	SODA (Software I	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	193											
194	Cioch F.A.	Teaching Software Design Usi	1990	Compute	For the past dec	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	194											
195	Lightner S., Hartr	Inventory of computer softwa	1989	Journal o	This article repor	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	195											
196	Hoffman D.	An undergraduate course in s	1988	Lecture N	While there are r	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	196											
197	Byrd-Bredbenner	Computer-analyzed dietary in	1988	Journal o	The purpose of t	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	197											
198	Robertazzi T.G.	Computer Communications St	1988	IEEE Tran	A number of und	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	198											
199	Weiss D.M.	Teaching a Software Design M	1987	IEEE Tran	This paper descri	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	199											
200	Wood D.L.	Designing microcomputer pro	1986	Compute	The microcompu	English	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	200											

Anexo 2

Formulario de Extraccion de Datos

La siguiente planilla es el formulario de extracción del protocolo final. Especifica cada punto que se quiere extraer del estudio con una breve descripción de lo que se quiere obtener en cada uno de los puntos del formulario.

	Generales	
		Definicion de cada uno de los puntos
	Nivel de dificultad de la extraccion	
	ID. Selección	El id de seleccion es el orden en que se encontro el articulo en la salida de la seleccion de la busqueda en SCOPUS.
	Titulo	
	Autores	
	Abstract	
	Fuente	
	Año de publicación	
	Tipo de publicación (revista, conferencia, revista tecnica)	
	Universidad / Organización de los autores	Este punto es para indicar la universidad u organización a la cual pertenecen el o los autores del articulo.
	Pais de los autores	Este punto es para indicar el pais en la cual esta la universidad u organización a la cual pertenecen el o los autores.
	Universidad / Organización donde se realizó el estudio/investigacion/experimento	Este punto es para indicar la universidad u organización en la cual se hizo o hicieron los experimientos o estudios.
	País donde se realizó el estudio/investigacion/experimento	Este punto es para indciar el pais o paises en los cuales se hizo o hicieron los experimientos o estudios.
	Origen (motor, snowballing, etc)	
	Específico	
1	**1 - Objetivo del estudio**	Indica el objetivo principal del articulo que se esta analizando, se debe copiar textual el objetivo.
1.1	1.1 - El articulo es sobre una investigacion, experimiento o ambos.	Este punto es para indicar a que tipo de estudio se le esta realizando la extracción. Los tipos de investigación a seleccionar estan: * basada en entrevistas y/o cuestionario * basada en un caso de estudio o en casos de estudios (casos de estudio cualitativos). * basada en un expermiento * basado en "action research" * etc
2	**2 - El objetivo principal del estudio es conocer como diseñan software los estudiantes de grado (Si, Parcialmente)**	Este punto es para Indicar si el articulo tiene como objetivo principal conocer como diseñan software los estudiantes de grado (esto quiere responder a la pregunta de investigación RQ1). La respuesta es "Sí" o "Parcialmente", la respuesta "No" no es aceptada dado que implica la no realización de la extracción de datos del artículo.
2.1	2.1 - Resmen de lo que trata el articulo	
3	**3 - Métodos, técnicas y principios de diseño de software utilizados**	Este punto es para responder la pregunta de investigación RQ2. Se generan subpreguntas especificas para los diferentes puntos que se quieren responder u obtener informacion del articulo que se esta analizando. No solamente se quiere responder las preguntas de los metodos, tecnicas y principios de DS que utilizan los estudiantes de grado al realizar un diseño de software de cero, sino los que analizan en diseños de software ya realizados por terceros (digase tercero al estudiante, graduado, experto en DS, docente de DS que ha realizado el diseño de software que esta evaluando o analizando el estudiante de Ingenieria de Software que le ayude aprender DS).
3.1	3.1 - ARTEFACTOS	Este punto es para especificar si en el articulo se habla (nombra o especifica) los artefactos de DS que son utilizados por los estudiantes de DS al realizar un diseño desde cero o al analizar un diseño realizado por terceros. Si es articulo habla a nivel general sobre este punto, es acá donde se debe agregar esta información.
3.1.1	3.1.1. Seudocódigo	
3.1.1.1	3.1.1.1. En diseño realizado desde cero por estudiante	
3.1.1.2	3.1.1.2. En diseño realizado por tercero y analizado por estudiante	
3.1.2	3.1.2. Diagrama de clases	
3.1.2.1	3.1.2.1. En diseño realizado desde cero por estudiante	
3.1.2.2.	3.1.2.2. En diseño realizado por tercero y analizado por estudiante.	
3.1.3	3.1.3. Diagrama de secuencia	
3.1.3.1	3.1.3.1. En diseño realizado desde cero por estudiante	
3.1.3.2	3.1.3.2. En diseño realizado por tercero y analizado por estudiante	
3.1.4	3.1.4. Patrones de diseño	
3.1.4.1	3.1.4.1. En diseño realizado desde cero por estudiante	

3.1.4.2	3.1.4.2. En diseño realizado por tercero y analizado por estudiante	
3.1.5	3.1.5. Diagrama de objetos	
3.1.5.1	3.1.5.1. En diseño realizado desde cero por estudiante	
3.1.5.2	3.1.5.2. En diseño realizado por tercero y analizado por estudiante	
3.2	3.2 - PRINCIPIOS Y METODOS DE DISEÑO	Este punto es para agrupar los principios y/o metodos de diseño de software utilizados por los estudiantes de software al realizar un diseño de cero o al analizar un diseño de software realizado por tercero. Si el articulo habla a nivel general sobre este punto, es acá donde se debe agregar esta información. (este punto es para agregar solamente los principios o metodos utilizados al realizar un DS por los estudiates, otros puntos relacionados a esto se deben de agregar en la tab PReguntas EXtras)
3.2.1	3.2.1. Principios de diseño	Este punto es para Indicar si se habla de principios de diseños de software (Si/No) y si se especifica cuales en el artículo, nombrar en este punto. Es para diseños realizados por los estudiantes o si analizan diseños de software realizados por terceros.
3.2.1.1	3.2.1.1. En diseño realizado desde cero por estudiante	
3.2.1.2	3.2.1.2. En diseño realizado por tercero.	
3.2.2	3.2.2. Metodos de diseño	Este punto es para Indicar si se habla de métodos de diseños de software (Si/No) y si se especifica cuales, nombrar aca. (por ejemlo si nombra programación top down, bottom up, modular, estructurados). Es para diseños realizados por los estudiantes o si analizan diseños de software realizados por terceros.
3.2.2.1	3.2.2.1. En diseño realizado desde cero por estudiante	
3.2.2.2	3.2.2.2. En diseño realizado por tercero.	
4	**4 - Problemas o dificultades detectados**	Este punto es para responder a la pregunta de investigación RQ3 y RQ3.1. Esta pregunta es para responder si en el articulo se habla, investiga o expone de posibles problemas o dificultades detectadas por los estudiantes de grado al realizar un diseño, o al estudiar o aprender Diseño de Software.
4.1	4.1 - FALTA DE INFORMACION	Este punto es para especificar si en el articulo se habla, investiga o expone problemas con relación a la falta de material o informacion con los cuale pueden encontrarse los estudiantes de grado para poder realizar un buen diseño de software.
4.1.1	4.1.1. Por Requisitos	Este punto es para indicar si la falta de informacion para poder realizar un buen diseño es por falta de informacion en los requisitos.
4.1.1.1	4.1.1.1. Mal formulados o inentendibles.	Este punto es para indicar que se tiene todas las definiciones de los requisitos que se deben realizar pero no estan bien formulado el problema, tiene mucha definición pero no se entiende lo que se quiere.
4.1.1.2	4.1.1.2. Escasa definición, informacion o incompletos	Este punto es para indicar si se cuenta con toda la informacion de los requisitos que se necesita para poder realizar un buen diseño.
4.2	4.2 - PROBLEMAS EN LOS CURSOS DE DISEÑO DE SOFTWARE	Este punto es para especificar sobre los problemas con los cuales se encuentran los estudiantes al realizar un curso de diseño de software.
4.2.1	4.2.1. Mal dictado el curso por los docentes	Este punto es para saber si los docentes no son claros, se contradicen entre si en los conceptos o con lo dado en cursos anteriores, mala preparacion de los docentes.
4.2.2	4.2.2. Mal formulado el curso	Este punto es para saber si todo los temas que son importante o fundamentales para poder realizar un buen diseño de software se desarrollan en los cursos.
4.2.3	4.2.3. Falta de Material en el curso	Este punto es para indicar si el curso brinda todo el material necesario a los estudiantes que les permita realizar un bueno diseño.
4.3	4.3 - PROBLEMAS DE LOS ESTUDIANTES CON DS	Este punto es para especificar los problemas con los cuales se encuentra o tiene un estudiantes de grado relacionados con el aprendizaje o realización de diseño de software.
4.3.1	4.3.1. Falta de motivación	Este punto es para ver si el articulo nombra o habla de la falta de motivación de los estudiantes al aprender diseño o al querer entender el diseño de software. (Se puede especificar si es por: la complejidad de aprender diseño, porque cada docente tiene diferentes puntos de vista de un mismo tema, porque muchas veces los cursos son muy guiados y al enfrentarse los estudiantes a realizar un diseño SOLOS no saben como resolverlo, etc). Si en el articulo se habla de la motivicación y de como se pueden o estan motivados los estudiantes, se debe especificar en este punto tambien). Tambien se quiere ver si la falta de motivación es un problema para que los estudiantes puedan realizar un buen diseño o aprender diseño de software.

4.3.2	4.3.2. Falta de experiencia	Este punto es para especificar si en el articulo se haba de la falta de experiencia que tienen los estudiantes al realizar un diseño, y como esto puede repercutir al realizar un mal diseño de software. Por ejemplo: falta de experiencia en las herramientas, el lenguaje de programación, falta de experiencia laboral (que diferencia si se habla o se compara en el articulo entre el diseño realizado por un senior o con experiencia laboral y un estudiante), o un estudiante con y sin experiencia laboral al realizar un DS, etc.
4.3.3	4.3.3. Falta de formacion en Diseño de Software	Este punto es para especificar si el articulo expone informacion sobre la mala formacion de los estudiantes por tener conceptos y/o definiciones erroneas sobre diseño de software que haga que el diseño quede mal o de una calidad no deseada, o falta de conocimientos previos de diseño que son necesarios para realizar un buen diseño de software o comprender un diseño de software realizado por terceros. TAMBIEN PODRIA SER LAS DIFICULTADES QUE TIENE LOS ESTUDIANTES CON EL DISEÑO, NO SE SI ESTE NOMBRE DE FORMACION EN DISEÑO LO CUBRE.
5	**5 - Evaluación de calidad realizada**	Este punto es para detectar si el artículo habla (investiga o expone) sobre la evaluación de la calidad de los diseños realizados o modificados por los estudiantes de grado, o sobre la calidad en la formacion del estudiante con respecto al aprendizaje y la influencia que esta puede tener al realizar un diseño de software. Es para responder las preguntas RQ4 y RQ41.
5.1	5.1 - POR ANALISIS DE DISEÑO REALIZADO POR LOS ESTUDIANTES	Este punto es para detectar si en el articulo se habla sobre la calidad del diseño que han realizado los estudiantes desde cero. Sea tanto la calidad del ultimo diseño de software realizado como si se realizaron DS previos a realizar el diseño final.

Anexo 3

Protocolo de la SLR de la tesis

A continuación presentamos la versión final del protocolo de la SLR de la tesis. Describimos cada una de las etapas de protocolo de la SLR.

3.1. Especificación de las preguntas de investigación

Las preguntas de investigación que debemos responder las especificamos en la tabla 3.1.

Tabla 3.1: Preguntas de investigación de la SLR.

PI	**Pregunta**
PI1	¿Existen reportes de cómo diseñan software los estudiantes de grado?
PI2	¿Cuáles son las técnicas/métodos/principios de diseño de software que utilizan los estudiantes de grado en los diseños de software realizados?
PI3	¿Con que tipo de problemas se encuentran los estudiantes de grado al realizar un diseño de software?
PI3.1	¿Con que tipo de problemas o dificultades se encuentran los estudiantes de grado al estudiar o aprender diseño de software?
PI4	¿Es evaluada la calidad de los diseños de software realizados por los estudiantes de grado?
PI4.1	¿Cómo se realiza la evaluación de la calidad del diseño de software realizado por los estudiantes de grado?

3.2. Selección de datos

La cadena de búsqueda general que podemos utilizar en cualquier biblioteca digital la definimos en la tabla 3.2.

Tabla 3.2: Cadena de búsqueda general de la SLR.

((software design) and (student or undergraduate or teach or learn or train) AND (software engineering))

La cadena general que adaptamos para realizar las búsquedas en la librería digital SCOPUS las mostramos en la tabla 3.3.

Tabla 3.3: Cadena de búsqueda del Protocolo SLR

TITLE (design* AND (student* OR undergrad* OR teach* OR learn* OR train* OR impart* OR comprehen*)) AND (ABS (“software design*”) OR ABS (design of software) OR ABS (design software) OR AUTHKEY (“software design*”) OR AUTHKEY (design of software) OR AUTHKEY (design software)) AND ABS (student* OR undergrad*) AND (LIMIT-TO (LANGUAGE , “English”))

Los criterios de inclusión y exclusión los definimos en la tabla 3.4.

Tabla 3.4: Criterios de inclusión/Exclusión del Protocolo de la SLR.

INCLUIR	**EXCLUIR**
El artículo está vinculado al diseño detallado de software en estudiantes de grado.	Escrito en un idioma que no es inglés.
El artículo responde directamente una o más de las preguntas de investigación.	El artículo no está disponible en texto completo.
	Es una *keynote*, *workshop*, presentación, reporte de opinión o de experiencia.
	Es un artículo duplicado.
	Es un artículo corto de 2 o menos paginas.

El proceso de selección de los estudios comienza luego de ejecutar la cadena de búsqueda. Las etapas del proceso de selección son las especificadas en la figura 3.1.

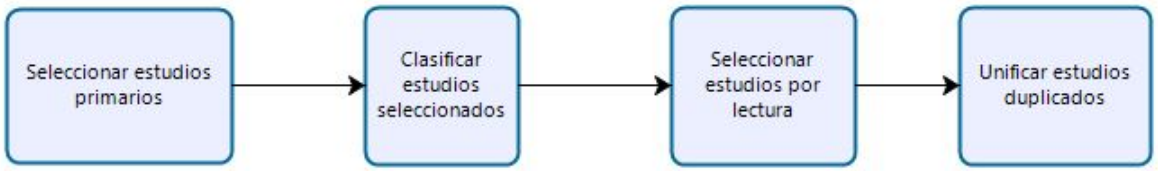

Figura 3.1: Proceso de selección de los estudios del Protocolo de la SLR.

La primera etapa del proceso de selección se denomina *Selección por lectura de título y resumen*. En esta etapa participa un revisor (el revisor 1), realiza dos veces la lectura de título y resumen de todos los estudios devueltos por la cadena de búsqueda y aplicando a cada estudio los criterios de inclusión y exclusión. Registra en una planilla su decisión para ambos resultados (0=Rechazado; 1=Aceptado; 2=A discutir) y las observaciones y comentarios. Esta planilla la definimos en el Anexo 1. Durante la segunda etapa, denominada *Clasificación de estudios según criterio*, se realiza la unificación de ambas planillas registradas por el revisor 1. Según la decisión que el revisor 1 marcó en cada selección, se toma la decisión final para cada estudio de acuerdo al criterio presentado en la tabla 3.5. Todos los estudios en estado "a discutir" son evaluados en la reunión de supervisión con el revisor 2. Esta reunión puede ser presencial, mediante e-mail o reunión por web. Se discuten dudas y el revisor 2 puede leer título y resumen de un número aleatorio de estudios. Si para algún estudio no se llega a consenso se solicita la opinión de un tercer revisor (revisor experto). La idea es incluir la mayor cantidad de estudios, por este motivo para los casos en que el revisor 1 selecciona un estudio como "aceptado" y el la segunda selección lo selecciona "a discutir" se decide aceptar el estudio primario.

Tabla 3.5: Casos según decisión de cada revisor.

			Revisor B	
		Aceptado	**Rechazado**	**A discutir**
	Aceptado	Aceptado	A discutir	Aceptado
Revisor A	**Rechazado**	A discutir	Rechazado	Rechazado
	A discutir	Aceptado	Rechazado	Experto

La tercera etapa, denominada *Selección por lectura completa*, tiene como entrada los estudios que resultaron aceptados de la etapa *Clasificación de estudios según criterio*. En esta etapa, el revisor 1 realiza la lectura completa de todos los estudios en estado aceptado que son el resultado de la segunda etapa. La lectura completa es realizada dos veces por el revisor 1 y registra en diferentes planillas su criterio de selección. Espera una semana entre cada lectura completa. El revisor 1 realiza la unificación de las planillas obtenidas de cada lectura completa según los criterios planteados en la tabla 3.5. Se realiza una reunión de supervisión con el revisor 2 donde se evalúan los estudios con estado a discutir. El revisor 2 puede elegir de forma aleatoria estudios para realizar la lectura completa de los mismo. En la cuarta etapa, denominada *Unificación de documentos duplicados* se tiene como entrada todos los estudios aceptados de la etapa *Selección por lectura completa*. En esta etapa se analiza si hay estudios que son parte de otros de los estudios seleccionados. Se evalúa entre los revisores que acciones tomar para cada caso, si dejar todos los estudios o eliminar los que tienen menos contenido.

El resultado de la cuarta etapa son los estudios que son la entrada al proceso de extracción de datos.

3.3. Extracción de datos

El proceso de extracción es ejecutado por un revisor (revisor 1) y supervisado por un segundo revisor (revisor 2). El revisor 1 realiza la extracción y completa el formulario de extracción. Espera una semana entre la primera extracción y realiza nuevamente la extracción para los estudios seleccionados. Realiza la comparación y unificación de ambos formularios para cada estudio. El revisor 2 analiza el formulario de extracción unificado y las diferencias que encuentra son discutidas entre ambos revisores. El revisor 2 puede seleccionar al azar un conjunto de estudios y realizar la extracción para luego comparar con lo extraído por el revisor 1. Las dudas sin resolver entre ambos revisores se discuten con un tercer revisor. El formulario de extracción esta definido en el anexo 2. Este formulario se completa basado en la información extraída de cada estudio.

Anexo 4

Ejecución del piloto 1

En este anexo presentamos la ejecución del piloto 1. La condición de parada de la ejecución es la definida en el capítulo 3. Al conjunto de artículos que devuelve la cadena de búsqueda solo consideramos los 100 primeros.

La cadena de búsqueda es la definida en la tabla 3.5 que ejecutamos el 13/03/2018 y devolvió 1328 artículos.

La ejecución del piloto 1 fue realizada por dos revisores, el revisor 1 que corresponde al autor de esta tesis y el revisor 2 que corresponde a Silvana Moreno co-tutora de la tesis.

4.1. Proceso de Selección

Las etapas del proceso de selección del piloto 1 se definen en la figura 4.1.

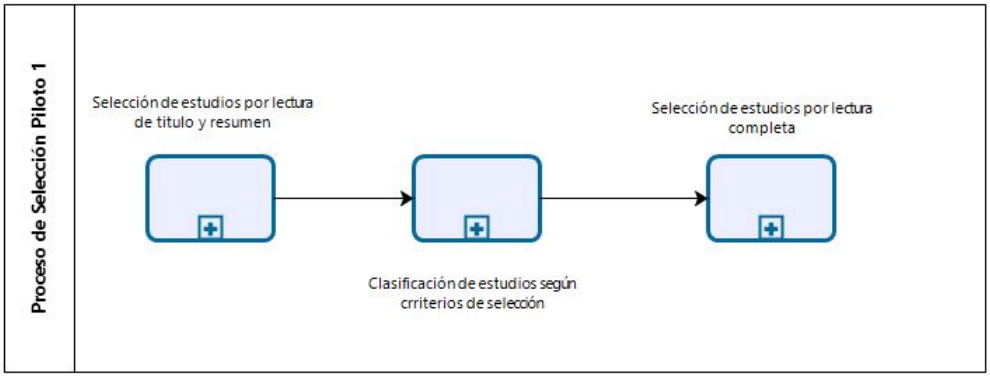

Figura 4.1: Etapas del proceso de selección del Piloto 1

En las figuras 4.2, 4.3 y 4.4 se detallan las actividades de cada una de las

etapas del proceso de selección del piloto 1.

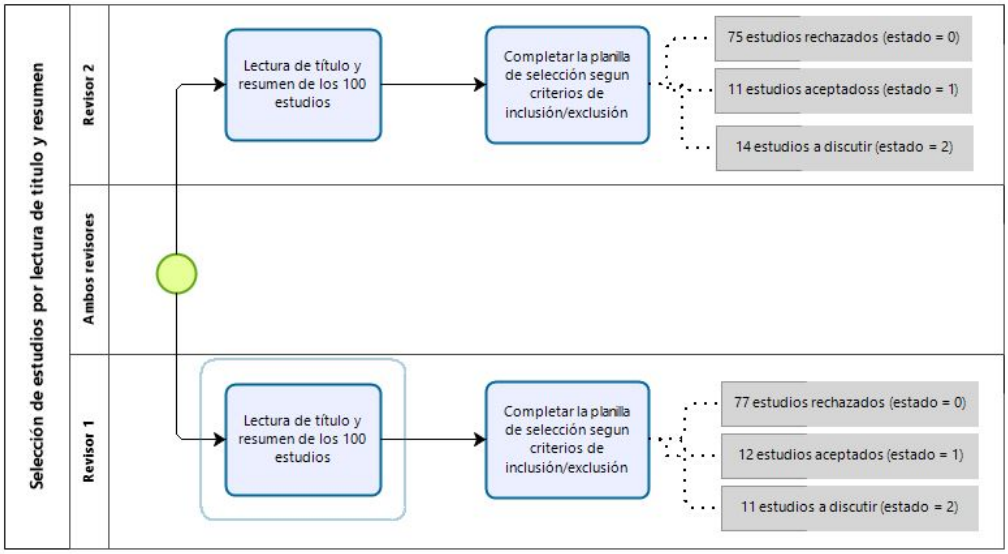

Figura 4.2: Etapa1 Proceso de selección Piloto 1

Previo a comenzar con el proceso de selección se construyó una planilla de selección que es utilizada por cada revisor para registrar los resultados de su proceso de selección.

El proceso de selección se realiza para los primeros 100 estudios, de los 1328 estudios ordenados por año de publicación en forma descendente. En la etapa 1, como se muestra en la figura 4.2 cada revisor de forma independiente realiza la lectura de título y resumen; y aplica los criterios de inclusión y exclusión. La decisión (0=Rechazado; 1=Aceptado; 2=A discutir) para cada estudio es registrada en la planilla de selección.

El revisor 1, clasificó de los 100 estudios, 77 rechazados, 12 aceptados y 11 a discutir. El revisor 2, clasificó 75 rechazados, 11 aceptados y 14 a discutir. Finalizada la etapa 1, se da comienzo a la etapa 2, donde el revisor 1 realiza la unificación de las planillas de ambos revisores. El resultado de la unificación son 69 estudios rechazados, 10 estudios aceptados y 21 estudios a discutir. El criterio para la unificación es si ambos revisores tuvieron la misma decisión de clasificación, ese es el valor que se pone como resultado unificado al estudio, sino el resultado de la unificación es a discutir. Los estudios en estado a discutir se van a volver a clasificar en la reunión de consenso. En la tabla 4.1 se tienen los valores (0=Rechazado; 1=Aceptado; 2=A discutir) con los criterios utilizados al momento de unificar las planillas de selección de ambos revisores.

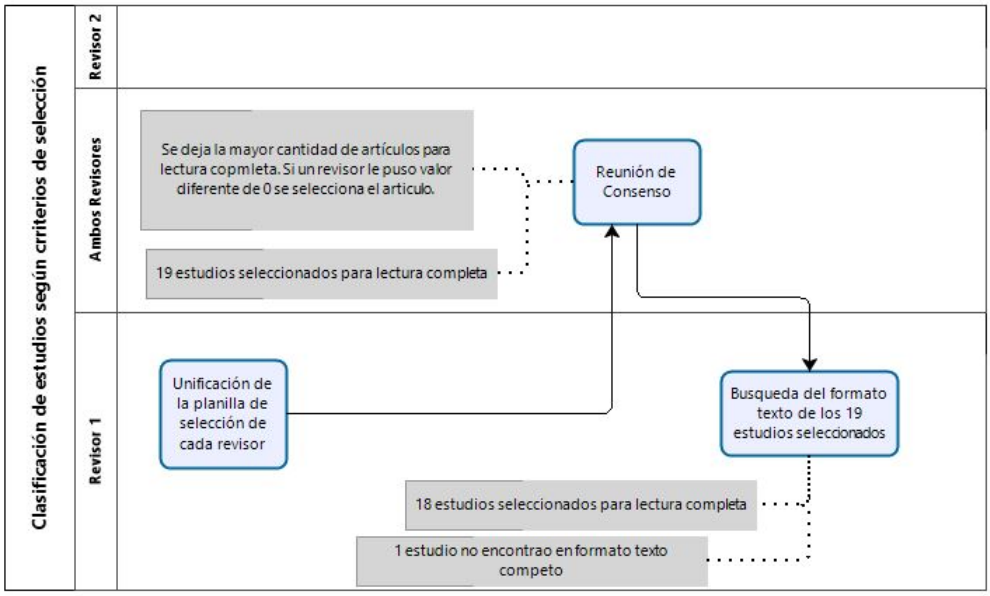

Figura 4.3: Etapa2 Proceso de selección Piloto 1

En la reunión de consenso se evalúa cada uno de los 21 estudios en estado a discutir, dando como resultado 9 estudios en estado aceptado y 12 estudios en estado rechazado. No fue necesario pedir la opinión de un tercer revisor por no tener diferencias de opinión en la clasificación de los estados durante esta reunión. El resultado de la etapa 2, son 19 estudios para realizar lectura completa, solo 18 estudios se encuentran en formato de texto completo en la web o se obtuvo respuesta de los autores al intentar contactarlos. En la etapa 3, que se muestra en la figura 4.4 cada revisor lee de forma independiente el texto completo de los 18 estudios, y completa la planilla de selección con su decisión de clasificación según los criterios de inclusión o exclusión para cada estudio. El revisor 1 clasificó 7 estudios rechazados, 8 estudios aceptados y 3 estudios a discutir. El revisor 2 clasificó 11 estudios rechazados y 7 aceptados. El revisor 1 realiza la unificación de ambas planillas. El resultado de la unificación son 3 estudios aceptados, 8 estudios rechazados y 7 estudios a discutir. En la reunión de consenso, de los 7 estudios clasificados en estado a discutir, 2 estudios fueron aceptados y 5 rechazados. No fue necesario pedir la opinión de un tercer revisor por no tener diferencias de opinión en la clasificación de los estados durante esta reunión. El resultado de la etapa 3 son 5 estudios para realizar la extracción.

Los detalles de los cinco estudios seleccionados luego del proceso de selección se presentan en la tabla 4.2.

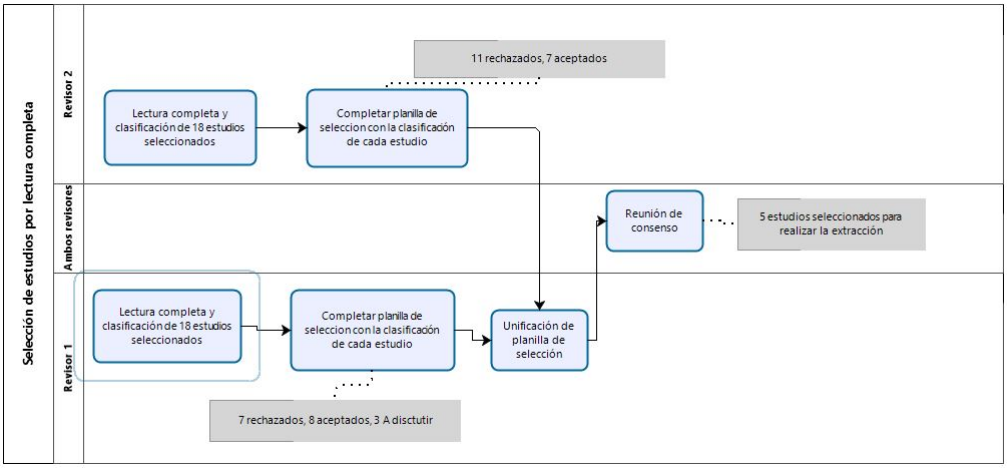

Figura 4.4: Etapa3 Proceso de selección Piloto 1

Tabla 4.1: Unificación de criterios de selección entre revisores.

$Revisor1$	$Revisor2$	$Unificacion$	$Observacion$
0	0	0	Se rechaza
1	1	1	Se acepta
2	2	2	A discutir en reunión de consenso
1,2	0	2	A discutir en reunión de consenso
0,2	1	2	A discutir en reunión de consenso
0,1	2	2	A discutir en reunión de consenso

4.2. Proceso de Extracción

En la figura 4.5 se detallan las actividades de cada una de las actividades del proceso de extracción del piloto 1.

El revisor 1 realizar la extracción de datos de los 5 estudios seleccionados en el proceso de Selección y completa el formulario de extracción con la información obtenida de cada uno de los estudios. Se espera una semana antes de que el revisor 1 realice una nueva extracción a los mismos 5 estudios (realizando el proceso de test-retest). El revisor 1 unifica ambas extracciones controlando las diferencias en un única planilla de extracción. El revisor 2 valida la planilla de extracción unificada y realiza las correcciones que crea necesarias. Se realiza una reunión entre ambos revisores donde se discuten las correcciones y las dudas que le surgieron durante la extracciones al revisor 1.

Tabla 4.2: Estudios incluidos luego del proceso de selección.

ID	*Anio*	*Titulo*	*Autores*
S1	2017	Do UML object diagrams affect design comprehensibility? Results from a family of four controlled experiment	Torchiano M., Scanniello G., Ricca F., Reggio G., Leotta M. s
S2	2017	Student software designs at the undergraduate midpoint	Thomas L., Zander C., Loftus C., Eckerdal A.
S3	2017	A study of the use of a reflective activity to improve students' software design capabilities	T Coffey J.W
S4	2017	Imparting software engineering design skills	Thevathayan C., Hamilton M.
S5	2017	Conceptions of the students around object-oriented design: A case study	Flores P., Medinilla N.

4.3. Proceso de Síntesis de datos

En el proceso de síntesis de datos se presentan los resultados basados en la información obtenida durante el proceso de extracción de los 5 artículos unificados en 5 estudios.

Los objetivos de los estudios se escriben en su idioma original porque se realiza una extracción textual de los mismo. En la tabla 4.3, se detallan los objetivos de cada uno de los 5 estudios alineados con el objetivo de la investigación. Estos estudios investigan de forma directa o indirecta como diseñan software los estudiantes de grado o las dificultades con las que se encuentran al diseñar.

El estudio S1 es una investigación que evalúan los beneficios que se pueden obtener al utilizar un diagrama de clases y de objetos para comprender un diseño de software..

Los estudios S2 y S3 estudian como los estudiantes realizan diseños a partir de los requisitos especificados, que dificultades encuentran y como las pueden mejorar. Además, en el estudio S2 se investiga como evoluciona el diseño de software en los estudiantes al avanzar en la carrera y que resultado se obtiene al diseñar en grupo, si son capaces de evaluar diseños de otros estudiantes y diferenciar la calidad de los diseños. Por otra parte, el estudio S3 investiga como mejora el diseño al mejorar los requisitos

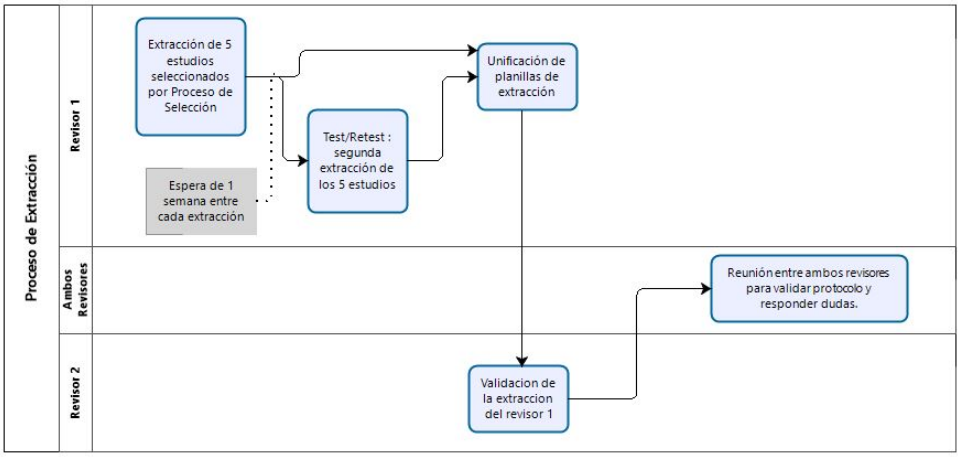

Figura 4.5: Etapa1 Proceso de Extracción del Piloto 1

El estudio S4 presenta como el utilizar metodología de investigación de acción en un curso de Diseño de Software mejora las habilidades de diseño en los estudiantes.

El estudio S5 es una investigación del tipo caso de estudio cualitativo sobre los conceptos y conocimiento que tienen los estudiantes sobre diseño de software orientado a objetos previo al comienzo de un curso sobre este tema.

En la tabla 4.4 presentamos los cinco estudios seleccionados con sus principales características. Todos los estudios son publicados en el año 2017. Cuatro estudios son artículos de conferencia y uno es un estudio publicado en una revista científica. La distribución de estudios por país de autor indica que Europa realiza la mayor cantidad de publicaciones de estudios relacionados a este tema durante el año 2017.

Tabla 4.4: Principales características de los estudios seleccionados.

Est.	Año Pub.	Tipo	Tipo Pub.	Pais Pub.
S1	2017	artículo de revista científica	Journal of Visual Languages and Computing 41 (2017) 10–21. Elsevier.	Italia
S2	2017	artículo de conferencia	ITiCSE '17, July 03-05, 2017, Bologna, Italy	Wales, USA, Sweden

Est.	Año Pub.	Tipo Pub.	Lugar Pub.	Pais Pub.
S3	2017	artículo de conferencia	SIGCSE '17, March 8–11, 2017, Seattle, WA, USA.	USA
S4	2017	artículo de conferencia	ACE '17, January 31-February 03, 2017, Geelong, VIC, Australia	Australia
S5	2017	artículo de conferencia	XII Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento 2017 (JIISIC'17)	Ecuador, España.

A continuación se realiza un resumen de las evidencias relacionadas a las preguntas de investigación.

Pregunta Investigación 1 (PI1): ¿Existen reportes de como diseñan software los estudiantes de grado?

Sí, los cinco estudios seleccionados reportan como diseñan software los estudiantes de grado. A continuación se presenta un resumen de cada uno de los estudios.

El estudio S1 presenta una familia de cuatro experimentos controlados que estudian si el uso de diagrama de objetos UML mejora la comprensión del diseño de software al agregarlo a un diagrama de clase UML. Los experimentos en que nos centramos son PoliTo2 y UniGes1 por ser realizados por estudiante de grado que son el foco de esta tesis. Los participantes de PoliTo2 fueron 17 estudiantes de grado. El experimento fue un ejercicio en una clase de un curso de programación orientada a objetos, que incluía una introducción a UML. Los estudiantes tenían una limitada familiaridad con UML y experiencia en desarrollo de software. Solo habían realizado cursos de introducción a la programación y estructura de datos. Los participantes de UniGel fueron 66 estudiantes de grado de un curso de Ingeniería de Software de tercer a˜no, donde aprendían UML. Una actividad obligatoria de este curso, fue que los estudiantes

Tabla 4.3: Objetivo de los estudios en su idioma original Piloto 1.

Est.	Objetivo
S1	The main objective of our study is to assess whether the use of UML (Unied Modeling Lan- guage) object diagrams improves comprehensibility of software design when this kind of diagrams is added to UML class diagrams.
S2	We were looking for characteristics of the development of skill at software design as students progress through the curriculum.
S3	One of the goals of the current work is to analyze the re ections of students to get a sense of their nature, depth, and contribution to understanding software design. //The goal of the current study is to explore ways to foster better re ection in students. Specically, the goal is to study how re ection on deciencies in a preliminary design might help students better anticipate what is needed in a design in subsequent projects.
S4	This paper presents the result of our action research to improve student design skills. Our approach combines project-based learning with weekly quizzes, tests and active learning tasks
S5	The aim of this research is to know the ideas that the students have about designing with objects before they started a module corresponding to this topic.

armaran equipos para desarrollar un proyecto de desarrollo de software usando especificaciones basados en UML. El experimento se realizó como parte de los ejercicios de laboratorio realizados dentro del curso en el que se llevó a cabo el mismo. En ambos experimentos a los participantes se les entregaron diagramas de clase solos y diagramas de clase con diagramas de objetos. Los resultados que obtuvieron fue que el uso de diagramas de clase y objetos mejora la comprensión del diseño sólo cuando el estudiante tiene experiencia. Si el estudiante tiene poco conocimiento de UML no se recomienda agregar diagramas de objetos UML.

En el estudio S2 a un grupo de estudiantes con conocimientos de programación JAVA, patrones de diseño, diagramas de clase y de secuencia, se les pide que en 50 minutos realicen un diseño de software desde cero sobre un problema de dificultad similar al problema de desarrollar una alarma de reloj. Los diseños obtenidos son analizados por dos investigadores que los clasifican utilizando la categoría definida en Eckerdal et al. (2006a) donde la categorización está basada

en cuan similar es la semántica del diseño. Además se comparan los diseños con otros diseños realizados años anteriores por estudiantes de postgrado o ya recibidos. En la tabla 4.5 se resume la clasificación de la categoría definida por Eckerdal et al. (2006a).

Tabla 4.5: Resumen de la clasificación de las categorías (Eckerdal et al., 2006a).

Categoría	**Descripción**
Nada	Diseño con poco contenido o sin contenido comprensible.
Reafirmación	Simplemente reformula los requisitos de la descripción de la tarea.
Difuso (o Skumtomte)	Reformula los requisitos con algo de información en texto, dibujos de un GUI (del inglés Graphical User Interface) o detalles poco importantes de implementación sin descripción del diseño.
Primer paso	No solo se trabaja sobre la descripción sino sobre la visión parcial del sistema, identificando las partes pero no como se relacionan entre sí o se trabaja sobre el diseño de uno de los componentes del sistema que puede ser la GUI o la interfaz a la base de datos.
Diseño parcial	Se trabaja en la descripción de cada una de las partes y en dar una visión del sistema con las relaciones entre sus partes, donde la descripción es incompleta y la comunicación no está completamente descripta.
Diseño completo	Se logra una solución bien implementada, se da una visión del sistema entendible, describe las partes que incluye responsabilidades y explicita comunicación entre las mismas. Se usa notación formal como ser UML (del inglés Unified Modeling Language) y casos de uso.

Los resultados que obtienen son que estudiantes próximos a recibirse no son buenos en la realización de un diseño de software. La mayoría de los diseños realizados quedan clasificados dentro de la categoría de "primer paso", sin obtenerse casi diseños en las categorías "diseño parcial" o "diseño completo", diseños completos o parcialmente completos. A nivel de enseñanza surgen implicaciones sobre a qué temas se les debe poner más foco para que los estudiantes puedan entender y realizar mejores diseños de software.

El estudio S3 pretende mostrar como un conjunto de estudiantes de un curso de tercer año de programación JAVA mediante actividades reflexivas mejoran sus habilidades de diseñar software. Se quiere estudiar como el reflexionar sobre deficiencias en un diseño preliminar ayuda a los estudiantes a anticiparse en lo que se necesita en un diseño en proyectos futuros. Los estudiantes realizan un diseño inicial como un diagrama UML y tienen cinco oportunidades durante el semestre para mejorar su trabajo. Se espera que los estudiantes realicen un documento de una página donde se tengan los cambios entre cada diseño, explicando por qué realizaron dichos cambios y cuáles fueron las lecciones aprendidas sobre diseño como resultado del trabajo realizado. Cada uno de los diseños obtenidos en cada etapa fue evaluado por los instructores, encontrando que cada diseño mejoraba con cada modificación. Los estudiantes realizan

diseños desde cero, diseñan parcialmente bien y los mejores diseños son los que tienen las últimas modificaciones. Utilizan diagramas UML y la calidad de los diseños es media.

El estudio S4 es una investigación donde se utiliza la metodología de investigación-acción (action research) realizada durante el dictado de un curso de fundamentos de ingeniería de software durante tres semestres. Se investiga como los estudiantes identifican conceptos equivocados de diseños de software y cómo ayudarlos a superar esos conceptos equivocados. La metodología de investigación-acción permitió mejorar los métodos de enseñanza durante los tres semestres. Se realizó la triangulación de la información obtenida a través de evaluaciones realizadas en la clase, exámenes y respuestas a cuestionarios realizados durante el curso. Cada curso está estructurado en dos partes, una con aprendizaje pasivo y otra con activo. Se cuenta con una herramienta ITS (Intelligent Tutoring System) para responden preguntas (aprendizaje pasivo) o realizar diagramas de clases con la ayuda de un asistente (aprendizaje activo). En base a las respuestas de los estudiantes se infiere en que tienen problemas y se los puede ayudar rápidamente. Esta herramienta fue diseñada para dar enseñanza personalizada a cada estudiante, está basada en responder o enseñar temas relacionados al diseño de software y que los diagramas de clase sean cada vez más complejos. Varios estudiantes tienen problemas identificando el escenario especificado y no entienden el diseño distribuido que reduce el acoplamiento entre clases. La conclusión a la que se llega es que la herramienta de ITS es importante tanto para los estudiantes como para los docentes. Los docentes pueden analizar el progreso de cada estudiante, viendo rápidamente donde tienen problemas. Los estudiantes pueden analizar su progreso y aprender conceptos de diseño. Los estudiantes usan diagramas de UML y revisan diseños realizados por tercero. Además se tiene una herramienta con tutoriales para aprender diseño y se evalúa como se enseña diseño.

El estudio S5 es una investigación realizada a estudiantes que cursan el módulo de Diseño de Software del sexto semestre de la carrera Ingeniería Informática de la Universidad Politécnica de Madrid. Se quieren conocer las ideas primarias que tienen los estudiantes sobre diseño orientado a objeto. Esto se hizo previo al comienzo de un curso sobre este tema a los 18 estudiantes que asistieron el primer día. Como aprender diseño de software es un fenómeno complejo de estudiar, para descubrir lo que los estudiantes entendían de los conceptos de diseño se aplicó un enfoque cualitativo basado en encuestas y entrevistas.

Las conclusiones obtenidas son que los estudiantes tienen ideas equivocadas sobre el diseño orientado a objeto, que el diseño no es tan importante como la codificación y que varios de los principios de diseño son ignorados por los estudiantes. A partir de este caso de estudio cualitativo los investigadores apreciaron que el diseño de software ha sido subestimado en cursos previos. Al terminar el módulo los estudiantes lograron tener mejor comprensión de los conceptos y el hacer un diseño tomó más valor. Se estudia el tema de enseñar diseño en profundidad.

Pregunta de investigación 2 (PI2): ¿Cuáles son las técnicas/métodos/principios de diseño de software que utilizan los estudiantes de grado en los diseños de software realizados?

Las técnicas utilizadas por los estudiantes al realizar diseño de software en estos 5 estudios son diagrama de clases, de secuencia y de objetos.

En el estudio S1 los estudiantes tienen una breve introducción a UML y realizan diagramas UML de clase y de objeto.

En el estudio S2 los estudiantes utilizan prioritariamente diagramas de clase y de secuencia al realizar sus diseños de software.

En el estudio S3 los estudiantes realizan un diseño preliminar y uno final con diagramas de clase UML.

En el estudio S4 los estudiantes mediante una herramienta de tutorial, evalúan y/o crean diagramas de clase o de secuencia UML. Además conectan código con el diagrama que expresa el diseño.

En el estudio S5 del análisis de las respuestas que dieron el grupo de estudiantes a las preguntas realizadas sobre conceptos de diseño de software se encontró que los principios que los estudiantes relacionan con diseño de software son adaptabilidad y modularidad.

El lenguaje de modelo UML es muy utilizado para realizar diseño, siendo el diagrama de clase el más utilizado, seguido por el diseño de secuencia. Dentro del conjunto de principios de diseño solo son nombrados brevemente los principios de “adaptabilidad” y “modularidad”.

Pregunta de investigación 3 (PI3): ¿Con que tipo de problemas se encuentran los estudiantes de grado al realizar un diseño de software?

Son varios los problemas encontrados al realizar un diseño de software en los 5 estudios. Estos problemas los clasificamos en tres grupos que son “falta de información”, “problemas con los cursos de diseño de software” y

"problemas de los estudiantes con el diseño de software". En el primer grupo se consideran los estudios con los problemas que tienen los estudiantes debido a la falta de información en los requisitos. Esto puede deberse a que los requisitos estén mal formulados, tengan escasa definición o estén incompletos. Esto afecta el entendimiento del caso de estudio planteado y la resolución del mismo. En el segundo grupo están los estudios con los problemas relacionados a los estudiantes de grado al realizar un curso de diseño de software. Estos problemas se refieren al conocimiento o forma de dictar los cursos de diseño de software por los docentes; o a no tener estrategias pedagógicas consistentes, unificadas y sincronizadas entre los diferentes cursos donde se dicta diseño de software. Además la falta de coordinación en el temario de los cursos, de material otorgando a los estudiantes o que la dinámica de los cursos no sea la más adecuada para enseñar diseño son otros de los problemas considerados en este grupo. En el tercer grupo se consideran los problemas relacionados a la falta de formación en diseño de software que tienen los estudiantes de grado. En este grupo se tiene en cuenta la falta de experiencia para realizar diseños de software y de motivación para aprender diseño de software. Esto último debido a que el diseño de software les parece difícil de aprender. Los estudiantes tienen poca experiencia y familiaridad lo que dificulta la realización de diseños de calidad.

En la tabla 4.6 presentamos la vinculación de los estudios con los diferentes problemas con los que se encuentran los estudiantes al realiza, estudiar o aprender diseño de software que fueron detectados en estos 5 estudios.

Tabla 4.6: Vinculación de problemas encontrados en los estudios.

Problema	$S1$	$S2$	$S3$	$S4$	$S5$
Requisitos mal formulados (o poco entendibles)			X		X
Problemas en la forma de dictar los cursos de diseño de software				X	
Falta de motivación		X	X	X	
Falta de experiencia	X	X	X	X	X
Mala formación de los estudiantes	X	X		X	X
Conceptos de diseño de software aprendidos de forma errónea	X	X		X	X

Pregunta de investigación 4 (PI4): ¿Es evaluada la calidad de los diseños de software realizados por los estudiantes de grado?

La calidad de los diseños de software es evaluada o considerada en 4 de los 5 estudios seleccionados. Los estudios en los que se evalúa la calidad de los diseños por los estudiantes de alguna forma son S1, S2, S3 y S4.

Pregunta de investigación 4.1 (PI4.1): ¿Cómo se realiza la evaluación de la calidad del diseño de software realizado por los estudiantes de grado?

A continuación se presenta de qué forma se aborda el tema de cómo se evalúa la calidad del diseño de software realizado por los estudiante en los diferentes estudios.

En el estudio S1 se utilizó el número de defectos originados desde la comprensión del modelo para determinar si el uso de diagrama de objetos mejora la calidad del diseño.

En el estudio S2 la evaluación de la calidad se realizó comparando los diseños realizados por los estudiantes con otros diseños. Estos diseños de comparación fueron realizados en otros experimentos por estudiantes de grado o ya recibidos.

En el estudio S3 la calidad es evaluada mediante los cambios realizados entre el primer y el quinto proyecto. Los estudiantes tienen un inventario con los cambios que ocurrieron de forma de guiar las actividades reflexivas que determinan porqué los cambios son necesarios y cuáles son las lecciones aprendidas.

El estudio S4 basado en cuestionarios diseñados para marcar errores comunes de diseño, permitió que los estudiantes corrigieran las ideas equivocadas tempranamente. Esto logró mejorar el compromiso, la satisfacción y el desempeño de los estudiantes sustancialmente.