Encuentro Internacional de Educación en Ingeniería ACOFI
2º Congreso Latinoamericano de Ingeniería
2019 10 al 13 de septiembre - Cartagena de Indias, Colombia
RETOS EN LA FORMACIÓN DE INGENIEROS EN LA ERA DIGITAL

ENSEÑANZA DE LA INGENIERÍA DE SOFTWARE BASADA EN COMPETENCIAS FUNDAMENTALES Y AULA INVERTIDA

Óscar Hernán Franco Bedoya, Sandra Victoria Hurtado Gil

Universidad de Caldas
Manizales, Colombia

Resumen

Los avances tecnológicos recientes sumados a las características de las nuevas generaciones, marcadas por un mayor uso y familiaridad con el uso de herramientas relacionadas con las tecnologías de información y comunicación (TIC), plantean nuevos retos en los procesos de enseñanza y aprendizaje, en particular para la enseñanza de la ingeniería de software en estudiantes de pregrado. Para afrontar estos retos se ha definido un proceso de enseñanza y aprendizaje enfocado en una serie de competencias fundamentales que han sido extraídas de la literatura especializada y de la experiencia de los autores, en conjunto con la estrategia colaborativa de enseñanza basada en el aula invertida (en inglés “flipped learning” o “flipped classroom”).

Debido a que el software es un campo cuyas técnicas y prácticas evolucionan continuamente (plataformas y lenguajes de desarrollo, marcos de trabajo, métodos ágiles, etc.), el cuerpo de conocimiento asociado también crece rápidamente. En contraste, las competencias fundamentales (habilidades de comunicación y trabajo en equipo, diseño de interfaces humano-máquina, manejo de versiones de productos, etc.) se conservan relativamente estables. Para esta propuesta se han definido una serie de competencias -relacionadas con los conceptos fundamentales y con las técnicas y prácticas recientes- que deben adquirir los ingenieros de sistemas para desempeñarse en el rol ingenieros de software, especialmente en proyectos de desarrollo.

El uso de estrategias definidas en el aula invertida como: definición de rutas de aprendizaje, videos, aprendizaje basado en problemas "reales" y el uso de redes sociales, entre otras, han permitido soportar la adquisición de las competencias definidas, lo cual se evidencia no solo en el desempeño académico de los estudiantes, sino también en la calidad de los proyectos entregados.

Palabras clave: aula invertida; ingeniería de software; competencias

Abstract

The recent technological advances, plus the new generation's characteristics marked by a bigger use and familiarity with Information and Communication Technologies (ICT) related tools, present new challenges for the teaching and learning processes, and in particular for the teaching of Software Engineering to undergraduate students. To face these challenges a new teaching and learning process has been defined, focused in several fundamental competencies extracted from specialized literature and from the authors' experience, joined with the collaborative teaching strategy based on flipped learning or flipped classroom.

Because software is a field whose practices and techniques continually evolved (development platforms and languages, frameworks, agile methods, etc.), the associated body of knowledge also grows rapidly. By contrast, the fundamental competencies (communication abilities and team work, human-machine interface design, product versions management, etc.) are conserved basically without changes. For this approach, a series of competencies have been defined –related to the fundamental concepts and to the recent practices and techniques- that must be acquired by system engineers to perform in the software engineer role, especially in development projects.

The use of strategies defined in the flipped classroom, such as: learning paths definition, videos, "real problems" based learning, and social networks use, among others, had allowed to support the acquisition of the defined competences, which can be evidenced not only in the students' academic performance, but also in the quality of the delivered products.

Keywords*: flipped classroom; software engineering; competencias*

1. Introducción

La Ingeniería de Software se originó a finales de los años 60 solo como una idea en una conferencia, y más de cincuenta años después ya se considera una disciplina, que cuenta, entre otros, con estándares, prácticas, guías curriculares y un cuerpo de conocimiento (Bourque, P. & Fairley, R. Eds., 2014). Este cuerpo de conocimiento no es algo estático y ya definido completamente, sino que se encuentra en constante evolución, lo cual representa un reto importante para su enseñanza en programas de pregrado.

A pesar de la evolución del cuerpo de conocimiento, existen un conjunto de conceptos y prácticas que se han consolidado con el tiempo, y que se pueden aplicar en proyectos con diferentes tecnologías o procesos, conformando de esta forma los fundamentos de la ingeniería de software. Se cuenta con diferentes iniciativas para consolidar y presentar de manera unificada estos fundamentos, las cuales se convierten en los principales referentes para establecer los contenidos de los cursos relacionados con la disciplina. Dentro de los referentes más importantes se tiene: Guía al cuerpo de conocimiento de la ingeniería de Software – SWEBOK, v3.0. (Bourque, P. &

Fairley, R. Eds., 2014); Guías curriculares para programas de pregrado en Ingeniería de Software (ACM & IEEE, 2015); "Peopleware" (DeMarco T. & Lister T., 2013) y Essence - Núcleo y Lenguaje para métodos de Ingeniería de Software, versión 1.2. (OMG, 2018).

Todos estos referentes, aunque trabajan temas compartidos y, por lo tanto, tienen muchos elementos comunes, tienen diferentes formas de presentar y organizar los elementos, por lo que se hace necesario unificar la información en aras de elaborar un plan curricular para los cursos. Un elemento integrador es el concepto de "competencia", que se puede definir como "*una combinación integrada de conocimientos, habilidades y actitudes conducentes a un desempeño adecuado y oportuno en diversos contextos*" (González-Díaz, C. & Sánchez-Santos, L., 2003). Dentro de las ventajas de usar competencias está el hecho de que algunos de los referentes analizados están ya planteados en esos términos, pero además varios entes certificadores de calidad (tanto en contextos empresariales como académicos) realizan sus evaluaciones basados en competencias, lo que genera un valor agregado a la propuesta.

Por otra parte, es pertinente analizar diferentes estrategias de enseñanza y aprendizaje que permitan presentar las competencias fundamentales a los estudiantes. Se encuentra entonces una estrategia denominada aula invertida (en inglés "flipped learning" o "flipped classroom"), en la cual se invierten los momentos tradicionales de la enseñanza, de manera que los estudiantes revisan los conceptos teóricos extra-clase y durante la clase se pueden realizar diversas actividades para afianzar y poner en práctica estos conceptos (Lagunes-Domínguez, A., Tafur-Jiménez, L., Giraldo-Ocampo, J., 2017).

Esta estrategia tiene varios elementos interesantes, pero se resaltan dos que mostraron su idoneidad para el proyecto: el hecho de que "c*onsidera, como elemento central la identificación de competencias meta que se han de desarrollar en el estudiante*" (Martínez-Olvera, W., Esquivel-Gámez, I., & Martínez-Castillo, J., 2014) y el uso que hace de las tecnologías para las actividades extra-clase (videos, foros, redes sociales, etc.), lo cual es particularmente propicio para las generaciones de estudiantes del milenio o "millennials" (Roehl, A., Reddy, S.L. & Shannon, G.J., 2013) (Akçayır, G., Akçayır, M., 2018).

En las siguientes secciones se presentarán brevemente algunos trabajos relacionados, luego la descripción de la propuesta realizada y finalmente las conclusiones y trabajos futuros a partir de la aplicación que se ha realizado hasta el momento.

2. Trabajos relacionados

La enseñanza de la ingeniería de software va más allá de la programación. Sin embargo, no siempre es fácil trazar la línea que las separa. Una de las mejores definiciones de la función del ingeniero de software es la que proporcionó el profesor Brian Randell[1]. "*La ingeniería de software es un desarrollo multi-personal de multi-versiones de un programa*" (Naur, P. & Randell, B. Eds., 1969). Esta breve definición tiene varias implicaciones importantes como: el software debe ser

[1] Fue uno de los organizadores de las dos primeras conferencias de ingeniería de software en 1969.

desarrollado por equipos, los productos del desarrollo deben ser probados, validados, versionados, etc., y dicho software será usado por personas diferentes a los que las desarrollan. Aunque desempeñarse como desarrollador implica competencias en programación, muchas otras competencias son también necesarias (Landwehr, C. et al., 2017).

La ingeniería de software ha evolucionado en los últimos años y una de las iniciativas más importantes es SEMAT (*Software Engineering Method and Theory*), que establece un núcleo de conceptos básicos que deben tenerse en cuenta al desarrollar software. En esta misma línea, el artículo de (Ibargüengoitia, G. & Oktaba, H., 2014) define un subconjunto de elementos de ESSENCE[2] para trabajar un curso inicial de ingeniería de software. En el artículo, los autores establecen un método de enseñanza con prácticas sociales, de gestión y de desarrollo para crear software. El método es soportado por una serie de guías de trabajo y formatos que ayudan a los estudiantes en la aplicación del método. Los autores concluyen que es posible, en un curso básico de ingeniería de software, desarrollar las competencias básicas de la ingeniería de software como las que se encuentran en el núcleo o *kernel* ESSENCE de SEMAT.

Un trabajo interesante es el de (Landwehr, C. et al., 2017), en donde los autores evalúan varios programas académicos de computación e ingeniería de software para definir una serie de "capacidades generales" que los desarrolladores de software deben alcanzar para construir software. Algunas de estas capacidades hacen referencia a aspectos sociales como la comunicación entre *stakeholders* y el diseño de interfaces hombre máquina; otras se relacionan más con el producto como el diseño de software y la parametrización. Con relación a las competencias del ingeniero de software la IEEE estableció el modelo SWECOM (*Software Engineering Competency Model*) para describir las competencias de los ingenieros de software.

Otro enfoque en la enseñanza de la ingeniería de software es el uso de juegos que soportan áreas del conocimiento específicas de la ingeniería de software. En una revisión sistemática de literatura (Souza, M. et al., 2018) encuentran 156 artículos relacionados con este tema, con un incremento notable en los últimos 10 años. Los autores concluyen que la utilización de juegos, en especial los denominados "juegos serios", desarrolla en los estudiantes las habilidades para la aplicación de conceptos y buenas prácticas relacionadas con la ingeniería de software. Los objetos virtuales de aprendizaje (OVAs) también se han utilizado en el proceso de enseñanza-aprendizaje de conceptos relacionados con la ingeniería de software, por ejemplo, (Romero-Lázaro, I.J., 2017) presenta un OVA para para la enseñanza de conceptos básicos de algoritmia.

3. Descripción de la propuesta

La propuesta que se presenta para modificar el contenido y metodología de los cursos de ingeniería de software de pregrado está basada en las competencias encontradas en la literatura en ingeniería de software, en los principios del aprendizaje invertido (*flipped learning*[3]) y en la experiencia de los autores. La figura 1 presenta las tres fases generales de la propuesta.

[2] Es un estándar de métodos para trabajar con ingeniería de software definido en el SEMAT
[3] https://flippedlearning.org/

Figura 1 – Fases generales de la propuesta
Fuente: Los autores

3.1. Identificar las competencias

El primer paso fue identificar un conjunto de competencias fundamentales de la disciplina que se incluirían en los cursos de pregrado. Para esto, se revisaron diferentes fuentes bibliográficas de la literatura científica y libros que se consideran referentes en este aspecto[4]. Con las referencias identificadas se creó un mapa conceptual de saberes y prácticas, que luego se refinarían en competencias. En la figura 2 puede verse una parte del mapa durante su construcción. Este mapa no solo permitió identificar las competencias fundamentales de la disciplina, sino también clasificarlas.

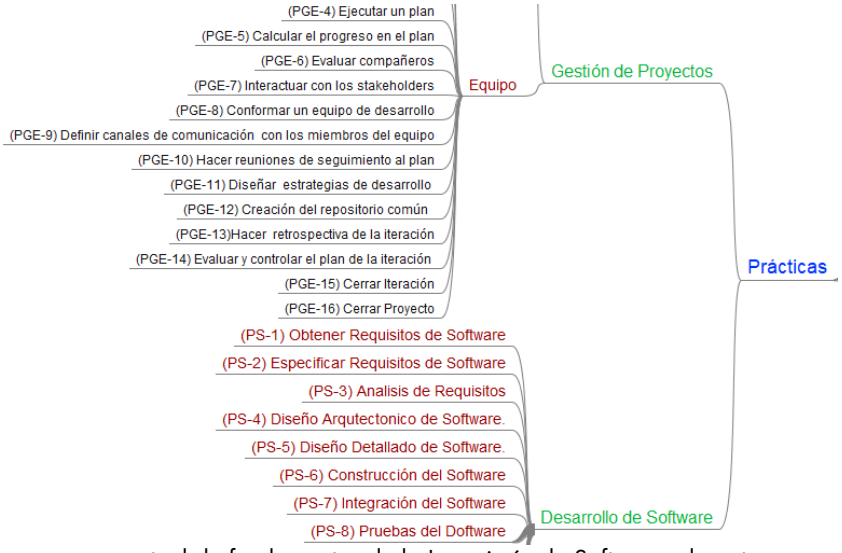


Figura 2 – Parte del mapa conceptual de fundamentos de la Ingeniería de Software, durante su construcción.
Fuente: Los autores

3.2. Clasificar las competencias

El siguiente paso consistió en clasificar las competencias a partir del mapa conceptual, agrupando elementos comunes (como calcular el progreso y evaluar el plan) o dividiendo algunos aspectos que eran demasiado amplios (como el diseño detallado o la construcción). La clasificación se basó en las categorías que (Landwehr, Carl, et al., 2017) establece (i.e., fundamentales y técnicas). Estas competencias se distribuyen entre los tres cursos que conforman la línea de Ingeniería de Software en la institución. Cabe anotar que la distribución no se realiza separando las competencias, sino identificando los elementos (temas y prácticas) de cada competencia que se deben presentar en cada curso. Por ejemplo, para una competencia relacionada con planeación

[4] Ver la sección de referencias

de proyectos, se puede trabajar la planeación individual en el primer curso y la planeación grupal en el segundo y tercer cursos. Además, en el segundo curso se puede usar la técnica del valor ganado, mientras que en el tercer curso se puede dar la técnica de tareas pendientes.

3.3. Estrategia de enseñanza

Teniendo clara la distribución de las competencias en temas y prácticas o técnicas, se definió la estrategia de enseñanza y aprendizaje, la cual se basa en las tácticas para el aula invertida que se aplican en los diferentes momentos del curso, así:

- Por cada curso se establece una guía o guion general en donde por cada semana se identifican las competencias y prácticas que se pretende que los estudiantes adquieran con las actividades programadas. Esto permite a los estudiantes identificar los objetivos de las sesiones y además que puedan preparar por su cuenta los temas que se trabajarán, antes de cada clase. Los contenidos conceptuales son en su mayoría presentados en videos y diapositivas, los cuales se acompañan de otro tipo de materiales adicionales y opcionales que el estudiante puede acceder si lo considera necesario. También se definen los laboratorios o talleres que se realizan en el salón de clase. En la figura 3 se puede ver una parte de la guía general para el primer curso de Ingeniería de Software.

Temas		Comp. Prac.	Videos	Material Apoyo	Lab.
Requisitos de Software & PSP-TSP					
Introducción Conceptos básicos de POO	Conceptos Básicos de La programación Orientada a Objetos	Fundamentales CF-5, Practicas PS-5	Clases Atributos Asociaciones Métodos Objetos Constructores Llamado de métodos Creación de Objetos		LAB01 LAB02
Practica de Programación en Java UML Básico	Taller Práctico en Java	Fundamentales CF-5, CF-6 Practicas PS-3, PS5, PS-6, PS-6.1 CI-5, CI-6 CG-4		Java API UML 2.5	LAB03

Figura 3 – Primera parte de la guía para el curso de Ingeniería de Software I.
Fuente: Los autores

- Como se mencionó en el punto anterior es muy importante la elaboración de materiales educativos para que los estudiantes se apropien, previamente, de los conceptos que se trabajaran en cada clase de manera presencial. Considerando los lineamientos de la estrategia de aula invertida, se ha comenzado a elaborar videos que presentan las diferentes temáticas. Estos videos se publican en el canal de YouTube http://www.youtube.com/c/CatchExceptionCanal, sin embargo, deben ser vistos por los estudiantes en la plataforma Edpuzzle que permite controlar quién ve los videos y también

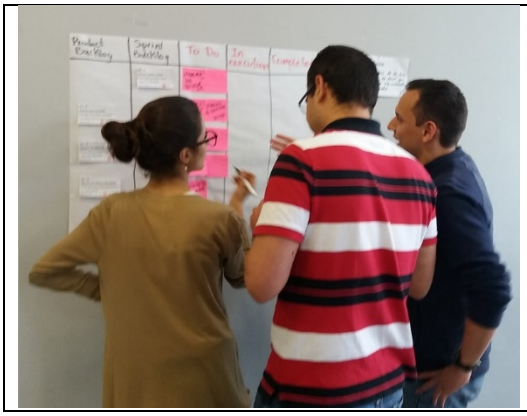
la realización de preguntas inmersas en el video que deben ser respondidas para poder continuar con su visualización.

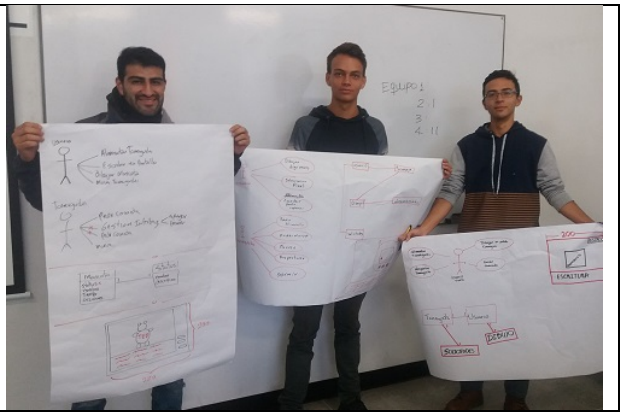
Aunque para algunos temas aún no se cuenta con videos propios, se tienen presentaciones que sirven de material de estudio y enlaces a otros videos relacionados a las diferentes temáticas que, aunque no han sido realizados en el curso, se adaptan al contenido. Tanto los videos como las presentaciones se acompañan de otros enlaces y referencias.

Similar a lo hecho por (Ibargüengoitia, G. & Oktaba, H., 2014), se han creado un conjunto de guías de proceso y formularios que permiten a los estudiantes desarrollar software en un entorno de administración de proyectos siguiendo un proceso de desarrollo basado en PSP (*Personal Software Process*) y TSP (*Team Software Process*).

- Durante las clases se realizan los talleres guiados por el profesor, trabajo colaborativo por parte de los estudiantes y la utilización de diferentes dinámicas para presentar y discutir los resultados. Este aspecto ha tenido una implementación gradual, en la medida en que los estudiantes van manejando mejor la preparación de los temas, y ya no se necesita tanto tiempo de clase para presentar aspectos teóricos, lo cual permite plantear un taller o una serie de preguntas de reflexión para resolver en grupos. Dependiendo de la dinámica seleccionada, los estudiantes pueden elaborar carteleras, realizar un foro, contestar preguntas para ganar puntos, hacer competencias, etc. La variedad en las sesiones hace que los estudiantes estén más motivados para asistir y para preparar los temas. En la tabla 1 pueden verse algunas fotos de actividades realizadas en clase.

Tabla 1 – Fotos de actividades colaborativas realizadas en clase





- Para complementar las tácticas anteriores los estudiantes deben llevar a cabo un desarrollo de software en el cual apliquen los elementos que se van viendo en el curso. Esto no solo permite trabajar alrededor de problemáticas reales (o muy cercanas a la realidad), sino que también permite evaluar la apropiación de las competencias que se definieron.

4. Conclusiones y trabajos futuros

En este artículo se presenta una estrategia de enseñanza-aprendizaje de la ingeniería de software para cursos de pregrado. La estrategia se fundamenta en el desarrollo de competencias en los estudiantes mediante la utilización de tácticas basadas en aula invertida como los videos y los talleres colaborativos. La estrategia ha permitido revisar y actualizar el plan de los cursos, elaborar material pertinente y orientado a las nuevas generaciones de estudiantes y realizar más actividades prácticas y colaborativas, lo cual facilita la apropiación y aplicación de las competencias deseadas.

El trabajo futuro se centra en el desarrollo de contenidos para los tres cursos de ingeniería de software que se basen en las competencias definidas y que permitan la definición de redes de aprendizaje dinámicas que permitan un aprendizaje personalizado por parte de los estudiantes. A corto plazo se busca que los estudiantes empiecen a colaborar con los contenidos de los cursos elaborando videos de temas específicos de ingeniería de software.

5. Referencias

Artículos de revistas

- Akçayır, G., Akçayır, M. (2018). The flipped classroom: A review of its advantages and challenges. Computers & Education. Vol. 126, pp. 334-345.
- González-Díaz, C. & Sánchez-Santos, L. (2003). El diseño curricular por competencias en la educación médica. Educación médica superior, Vol. 17, No. 4.
- Lagunes-Domínguez, A., Tafur-Jiménez, L., Giraldo-Ocampo, J. (2017). Propuesta de Flipped Classroom para el desarrollo de las competencias genéricas en estudiantes de ingeniería. Ingenierías USBMed, Vol. 8, No. 1, pp. 43-48.
- Landwehr, C. et al. (2017). Software Systems Engineering programmes a capability approach. Journal of Systems and Software, Vol 125, pp. 354-364.
- Roehl, A., Reddy, S.L. & Shannon, G.J. (2013). The Flipped Classroom: An Opportunity to Engage Millennial Students through Active Learning Strategies. Journal of Family and Consumer Sciences, Vol. 105, No. 2, pp. 44-49.
- Romero-Lázaro, I.J. (2017). Los objetos de aprendizaje basados en M-learning en la enseñanza de la programación de computadoras en estudiantes universitarios. Encuentro Internacional de Educación en Ingeniería ACOFI 2017.
- Souza, M. et al. (2018). A systematic mapping study on game-related methods for software engineering education. Information and Software Technology, Vol. 95, pp. 201-218.

Libros y capítulos de libro

- ACM & IEEE (2015). Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Joint Task Force for Computing Curricula.
- Bourque, P. & Fairley, R. Eds. (2014). Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press.

- DeMarco, T. & Lister, T. (2013). Peopleware: Productive Projects and Teams, third edition. Addison-Wesley Professional.
- Ibargüengoitia, G. & Oktaba, H. (2014). Identifying the scope of Software Engineering for Beginners course using ESSENCE. En libro: Software Engineering: Methods, Modeling and Teaching, Vol. 3. Universidad Nacional de Colombia, Sede Medellín.
- Martínez-Olvera, W., Esquivel-Gámez, I., & Martínez-Castillo, J. (2014). Aula invertida o modelo invertido de aprendizaje: Origen, sustento e implicaciones. En libro: Los Modelos Tecno-Educativos, revolucionando el aprendizaje del siglo XXI. DSAE-Universidad Veracruzana
- Naur, P. & Randell, B. Eds. (1969), Software Engineering: Report on a Conference sponsored by the NATO Science Committee. Scientific Affairs Division, NATO.
- OMG (2018). Essence – Kernel and Language for Software Engineering Methods, version 1.2. OMG (Object Magement Group).

Sobre los autores

- **Óscar Hernán Franco Bedoya**: Ingeniero de Sistemas, Magíster en Ciencias Computacionales y Doctor en Computación. Profesor Asociado Universidad de Caldas. oscar.franco@ucaldas.edu.co
- **Sandra Victoria Hurtado Gil**: Ingeniera de Sistemas, Magíster en Ingeniería de Sistemas y Computación. Profesor Asistente Universidad de Caldas. sandra.hurtado@ucaldas.edu.co

---