Universidad Internacional de La Rioja
Facultad de Educación

Trabajo fin de máster

Aprendizaje en red a través de GitHub en la era del *post-agile* para el módulo profesional de ciclo superior *Desarrollo web en entorno cliente*

Presentado por: David García Chaves
Tipo de trabajo: Propuesta de intervención
Director/a: Alicia Parras Parras

Ciudad: Santiago de Compostela
Fecha: Junio 2018

Resumen

En el contexto actual de la industria asociada al desarrollo web, el número de tecnologías y librerías que han tenido que aprender y desaprender los programadores en el último lustro para mantener su valía profesional, desafía cualquier análisis racional. Esto sitúa la competencia de aprender a aprender como un elemento clave para el futuro éxito profesional del alumnado del Título LOE SIFC03 *Desarrollo de aplicaciones web*.

En este Trabajo Fin de Máster se articula una propuesta centrada en el módulo profesional MP0612 *Desarrollo web en entorno cliente*, que pretende dar solución a este problema. Para ello, en base a las prácticas y valores del desarrollo de software *open source*, se destilará una metodología que, situando al alumno como protagonista, favorezca el aprendizaje en red.

La propuesta se sustancia alrededor de la plataforma de colaboración entre programadores GitHub, hogar del *open source* moderno. En torno a GitHub será posible construir una nueva metodología de aprendizaje - *Metodología de desarrollo del Open Source* o MOS-, cuya implantación se detallará en esta propuesta de intervención.

Se prestará especial atención a dos corrientes en la industria del software que se ubican bajo el paraguas del *post-agile*: *The Hacker Way* y "*agile is dead*". La primera servirá de filosofía e inspiración con su mantra "*code wins arguments*". De la segunda, se escogerán prácticas contrastadas para disminuir la fricción al trasladar las formas, maneras, principios y métodos del *open source* al aula.

Palabras Clave

Aprender a aprender; The Hacker Way; Agile; Open Source; Aprendizaje en red; GitHub; Aprender haciendo.

Abstract

In the current context of the web development industry, the number of technologies and libraries that programmers have had to learn and unlearn during the last five years to keep their professional value, defies any rational analysis. This fact places the *learning to learn* competence as a key element for the future professional success of the students of "Título LOE SIFC03 *Desarrollo de aplicaciones web"*.

This Master's Dissertation revolves around a proposal centered on the professional module MP0612 *Desarrollo web en entorno cliente*, which aims to solve this problem. In order to accomplish that, a methodology will be distilled, based on the practices and values of open source software development, placing the student at the center of the stage and favoring the establishment of learning webs.

This proposal is based on GitHub, the collaboration platform between programmers and home of modern open source. GitHub will allow us to build a new learning methodology - *Open Source Development Methodology* or MOS-, whose implementation will be detailed in this intervention proposal.

Special attention will be given to two different trends, under the *post-agile* umbrella, in the software industry: *The Hacker Way* and "*agile is dead*". The former will serve as philosophy and inspiration with its mantra "*code wins arguments*". From the latter, contrasted practices will be chosen to reduce friction by transferring the forms, manners, principles and methods of open source to the classroom.

Keywords

Learning to Learn; The Hacker Way; Agile; Open Source; Learning Webs; GitHub; Learn by Doing.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Abreviaturas más usadas

- **ABP**: Aprendizaje basado en problemas.
- **ABPr**: Aprendizaje basado en proyectos.
- **API**: Acrónimo. Ver glosario.
- **BC**: Bloque de contenidos.
- **CE**: Criterios de evaluación.
- **MOS**: *Metodología de desarrollo del Open Source*.
- **NDA**: Contratos de confidencialidad o *Non-Disclosure Agreements*.
- **PR**: *Pull Request*. Ver "*Issues, Pull Requests y Diffs*" en **Anexo A**.
- **RA**: Resultados de aprendizaje.
- **REPL**: Acrónimo. Ver glosario.
- **UT**: Unidad de trabajo.

Glosario

A lo largo de este TFM se utilizan los siguientes términos relacionados con la cultura informática. Algunas de las explicaciones pueden haberse simplificado en exceso buscando una mejor comprensión por parte de los no técnicos:

- **Atom**: Editor de código *open source* impulsado por GitHub (GitHub Inc., 2018c).
- **API** (*Application Programming Interface*): Conjunto de funcionalidades que una web -en este TFM, GitHub- ofrece al resto de programas, aplicaciones o máquinas en Internet para interactuar con ella. Si los humanos acceden a las webs a través de una interfaz gráfica desde un navegador o una app en un smartphone o tablet, los programadores usan las APIs para que sus programas hagan lo mismo. El acrónimo API es más extenso y genérico, pero en este TFM se usa en el sentido señalado.
- **Bash**: Ventana de comandos, al estilo de MSDOS, pero mucho más potente y con más solera. Ubicua en entornos *NIX -Unix, Linux-.

- **Diff**: Ver "*Issues, Pull Requests y Diffs*" en **Anexo A**.
- **eXtreme Programming**: Metodología de desarrollo software que da el pistoletazo de salida a todo el movimiento *agile* a partir del año 1999. Una de sus señas de identidad es que obliga a los programadores a trabajar en parejas delante de un ordenador (*pair programming*).
- **Fish**: Ventana de comandos, al estilo de MSDOS, y que algunos -es el caso del autor de este TFM- prefieren antes que Bash, por ser un poco más moderna y amigable.
- **Frontend**: Otra forma de referirse a la programación web o, por su ubicuidad, a la programación en el lenguaje *JavaScript*. Hay más lenguajes de programación asociados al *frontend*, como *Elm*, *PureScript*, *Reason* o *TypeScript*, por nombrar unos cuantos.
- **Geek**: Persona que siente fascinación por la tecnología, ya sea dispositivos, software o el funcionamiento del Mars Pathfinder. En España, por desgracia, no se suele utilizar y se recurre más a menudo a *freak*, pese a que *freak*, en su acepción original, no tiene nada que ver con *geek*.
- **Git**: Sistema *open source* de control de versiones para software, creado por Linus Torvalds -autor también del núcleo de Linux-, y que se ha convertido en el sistema más usado por la comunidad *open source*. Se encuentra en las entrañas de GitHub.
- **Hacker**: Programador extremadamente curioso con un afán por aprender desmedido. Como ejemplo de esta idea -o espíritu-, resulta especialmente reveladora la frase con la que Kelly Sommers (s.f.) cierra su perfil de *Twitter*: "*Relentless learner. I void warranties*" (Sommers, s.f.). Espíritu hacker en estado puro.
- **Issue**: Ver "*Issues, Pull Requests y Diffs*" en **Anexo A**.
- **Jedi**: Ver *Ninja*.
- **Log**: Historial de acciones realizadas al interactuar con un ordenador de alguna manera. Suelen materializarse en varios ficheros de texto en los que los programas vuelcan automáticamente las interacciones registradas.
- **Maker**: Versión tecnológica del más analógico *Do It Yourself* (DIY). Generalmente se refiere a aquellas personas que están involucradas en la cultura de la robótica casera.

- **Ninja**: Acepción usada para referirse a los programadores como verdaderas estrellas y divos que todo lo saben. Popularizada desde hace unos años en las ofertas de empleo de Silicon Valley, inicialmente, pero ampliamente usada desde entonces. Una búsqueda en Google de "*Silicon Valley job offers ninja*", arroja unos cuantos resultados interesantes. A veces también se usan los términos *Rockstar* y en menor medida *Jedi*.
- **Pull Request**: Ver "*Issues, Pull Requests y Diffs*" en **Anexo A**.
- **REPL**: Acrónimo de Read-Eval-Print-Loop. Herramienta que permite probar código de forma rápida e intuitiva. Se utiliza mucho para dar forma a ideas, como herramienta de exploración o simplemente de aprendizaje. No todos los lenguajes de programación ofrecen un REPL; sin embargo, los asociados al *frontend -JavaScript* o *Elm*- sí disponen de uno.
- **Rockstar**: Ver *Ninja*.
- **SCRUM**: La metodología *Agile* por excelencia. Muy extendida por todo el globo. Cuenta con un gran número de partidarios y unos cuantos detractores muy ruidosos como Erik Meijer (Reaktor, 2014; GOTO Conferences, 2015b).
- **SCRUM Master**: En la metodología *Agile* SCRUM se describen varios roles. Uno de los más importantes es el de especialista en SCRUM y facilitador, que recibe el nombre de SCRUM Master.
- ***Startup***: Empresas de nueva creación muy dinámicas y generalmente con toque tecnológico. *Airbnb* o *Uber* podrían considerarse, en sus inicios, *startups*.
- **Trap**: Género musical que bebe a dosis iguales del *hip hop*, música electrónica y hits comerciales. Muy publicitada en España desde hace un par de años. Se podría considerar el bastardo *punk* del *rap*.

1 Introducción

1.1 Justificación y planteamiento del problema

Cuando Bauman (2000) establece la metáfora de la liquidez, en cuanto a la capacidad intrínseca de los sólidos para permanecer inmutables en el tiempo frente a la transformación constante de los líquidos, es muy improbable que tuviese en mente un área concreta del saber. Sin embargo, en pocas se experimenta a diario de una forma tan drástica como en la programación. Y dentro de la programación, en la parcela del *frontend*, programación web o programación en el lenguaje *JavaScript*.

En este Trabajo Fin de Máster (TFM) se abordará este problema mediante la propuesta de una innovación para el módulo profesional **MP0612** *Desarrollo web en entorno cliente* del **Título LOE SIFC03** *Desarrollo de aplicaciones web* en la Comunidad Autónoma de Galicia. Dicha innovación intentará mimetizar una de las formas mediante la cual los programadores web afrontan esa necesidad de eterna reinvención: contribuyendo al desarrollo de productos *open source* alrededor de la plataforma GitHub.

Pero para poder entender mejor el porqué de este TFM, es necesario en primer lugar, ser conscientes de la situación a la que se enfrenta un profesional del *frontend* en el 2018.

1.1.1 El desarrollo web, la industria y el contexto

El desarrollo web

En el contexto actual de la industria asociada al desarrollo web, el número de tecnologías y librerías que han tenido que **aprender y desaprender** los programadores web en el último lustro para mantener su valía profesional, desafía cualquier análisis racional. Considérense dos muestras diferentes que justifican la afirmación vertida.

Como primera muestra la siguiente cita -en su idioma original pero con **énfasis** del autor de este TFM- de febrero de 2017, realizada por un programador web y que sintetiza una opinión generalizada en la industria:

> *"2016 was a pivotal year for JavaScript developers. That seems rather ironic considering that* ***every year is somewhat of a pivotal year*** *since JavaScript and* ***the web platform seem to be in a state of constant evolution***. *The* ***best***

> ***practices of yesterday*** *are* ***today's anti-patterns****;* ***yesterday's libraries****,* ***today's technical debt****."* (Holland, 2017, párr. 2)

Lo relevante de la afirmación citada -más allá de la literalidad de la misma- es que desde aproximadamente 2013 hasta la actualidad, se podría cambiar el año del comienzo de la cita (2016) sin que la afirmación dejase de ser cierta.

La segunda muestra atañe a la página web *JavaScripting* (Sokhan, 2014), que lista por popularidad las librerías usadas en desarrollo web -sólo en el lenguaje de programación *JavaScript*- y que en la fecha de redacción de esta justificación -febrero de 2018-, contaba con más de 1.200 librerías en competencia directa.

Ésta es la realidad a la que se enfrenta a diario un programador web y lo que es más grave, una situación para la que no han sido preparados todos aquellos que consigan el **Título LOE SIFC03** de *Desarrollo de aplicaciones web*.

La industria y el contexto

Por otra parte, es necesario considerar que todo el desarrollo web no acontece en el éter; existe un contexto en la industria caracterizado por una serie de prácticas, procesos y valores. Ese contexto, a día de hoy, podría ser denominado ***post-agile*** por las razones que se describen a continuación.

Primeramente, es innegable que desde la redacción del *Manifesto for agile software development* (Beck et al., 2001) la influencia de los valores y prácticas del *agile* no ha hecho más que crecer, hasta convertirse en el estándar *de facto* de la industria. El número de libros publicados sobre metodologías *agile* como SCRUM (Rubin, 2012; Sutherland y Sutherland, 2014) o Lean/Kanban (Anderson, 2010; Kniberg, 2011), prácticas relacionadas con el *agile* como Test-driven Development (Beck, 2003; Thomas y Heinemeier Hanson, 2005; Martin, 2009; Ruby y Copeland, 2017) o Behaviour-driven Development (Freeman y Pryce, 2009; Dees, Wynne y Hellesøy, 2013; Wynne, Hellesøy y Tooke, 2017) deberían justificar esta afirmación, pero es todavía más definitorio el hacer referencia a los más críticos con el *agile* para ser conscientes de su nivel de extensión e influencia (Reaktor, 2014; GOTO Conferences, 2015b).

Una vez establecida la ubicuidad del *agile*, debe tenerse en cuenta que, Dave Thomas, uno de los firmantes originales del *Manifesto for agile software development*, ha puesto voz a una parte de la industria con su "*agile is dead*" (Thomas, 2014; GOTO Conferences, 2015a); de ahí el ***post-*** en ***post-****agile* de este TFM.

Sin embargo, la realidad es algo más caprichosa, ya que la vigencia y adopción de metodologías *agile* sigue siendo muy significativa (SCRUM Alliance, 2015). La razón, probablemente, haya que buscarla en que metodologías ágiles como SCRUM -que originalmente surgían para liberar al

programador y disminuir la gestión y burocracia en las organizaciones- han acabado contribuyendo justamente a lo contrario. En las palabras del visionario y provocador Erik Meijer el *agile* (y más en concreto SCRUM) es un *"cáncer que debe ser eliminado de la industria"* (Reaktor, 2014).

Por si la convivencia de *agile* y "*agile is dead*" no fuese suficientemente confusa, es necesario considerar la alternativa surgida desde el corazón de Silicon Valley, ***The Hacker Way***. Mark Zuckerberg (2012) propone una filosofía que venera al programador como elemento central de la industria, en confrontación directa con la burocracia existente en todas las corporaciones. Esta concepción se ha impuesto allí donde ha surgido (Silicon Valley), pero no está tan claro que sea el caso en el resto del mundo. De hecho, Europa, es paradigmática por su alergia a *The Hacker Way* (GOTO Conferences, 2015b). La paradoja de la situación global se agranda si consideramos que *The Hacker Way* sí tiene un gran impacto a nivel mundial en la cultura del *open source*, que ha convertido en bandera y una suerte de filosofía vital su frase más famosa: "***code wins arguments***" (Zuckerberg, 2012, p. 69).

En este TFM, cuando se utiliza el término ***post-agile***, se hace referencia a la era en la que *agile*, "*agile is dead*", *The Hacker Way* y lo que queda de antes del *agile*, se ven forzados a convivir. Si bien, como se verá en el marco teórico, a la hora de establecer la innovación propuesta, se primará *The Hacker Way* y "*agile is dead*".

Y con esto, se finaliza la caracterización del contexto en el que van a trabajar todos aquellos que consigan el **Título LOE SIFC03** de *Desarrollo de aplicaciones web*.

1.1.2 Liquidez, aprendizaje y *open source*

Ante tal panorama es lógico plantearse la siguiente cuestión: ¿cómo hacen los profesionales de la programación web para actualizar su conocimiento bajo estas circunstancias? O desde otro ángulo mucho más interesante para el objetivo de este TFM, ¿cómo aprenden y desaprenden?

Para empezar, podría parecer que la única forma de mantenerse actualizado y hacer que el saber sea competitivo en un mercado tan cambiante -otra forma de referirse al concepto de liquidez antes tratado- sería plantear un aprendizaje continuo, iterativo, estratégico, pero profundo.

¿Y cómo lo consiguen?

El *open source* y el aprendizaje

Observando cualquiera de los proyectos *open source* más populares, es fácil darse cuenta de que los desarrolladores implicados **aprenden haciendo** (Dewey, 1897). Teniendo en cuenta lo inabarcable del conocimiento asociado al desarrollo software, los desarrolladores *open source* aprenden bajo demanda, según necesidad.

Además, la incorporación a estos proyectos no suele consistir en una inmersión total en el mismo, sin ningún tipo de filtro. Más bien al contrario, generalmente se produce un proceso paulatino de exposición al proyecto, muchas veces guiado por un **mentor**, en plena coincidencia con algunos de los principios defendidos por el *Manifesto for Software Craftmanship* (Bradbury et al., 2009).

Es muy importante recalcar este aspecto pues va a servir de *leit motiv* de todo este TFM: **si se aprende haciendo, el desarrollo de *open source* es una manera de aprender**.

Auge, diversificación y normalización del *open source*

Para poder hacerse una idea de la ubicuidad del fenómeno del *open source* en 2018, es necesario contextualizar tres fases determinantes -se solapan entre sí- que permiten entender mejor su naturaleza y evolución en los últimos 10 años -curiosamente el tiempo de vida de la plataforma GitHub, que ha servido de hogar y catalizador-:

- **Auge**: Proyectos *open source* personales, sin vinculación alguna por parte de las empresas.
- **Diversificación**: Empresas publicando *open source* como *Facebook* con 178 proyectos y 198 personas directamente implicadas (GitHub Inc., 2018h), *Twitter* con 147 proyectos / 45 personas (GitHub Inc., 2018w), *Airbnb* con 153 proyectos / 29 personas (GitHub Inc., 2018b) o *Microsoft* con 1.655 proyectos / 3.731 personas (GitHub Inc., 2018l).
- **Normalización**: Empresas contratando a desarrolladores de *open source* como forma de captar talento y proporcionando soporte para que sigan manteniendo esa pieza de software como los casos del creador de *Redux* -Dan Abramov- por *Facebook* (Abramov, 2015) o el creador del lenguaje de programación *Elm* -Evan Czaplicki- desde *Prezi* a *NoRedInk* (Toledo, 2016).

Ante esto, en este TFM se afirma que la implicación en la comunidad *open source* es la manera más correcta de acercarse al aprendizaje y enseñanza del desarrollo de software -si es que siquiera existe un manera "*correcta*" o "*más correcta*", teniendo en cuenta las reservas expresadas por el propio Erik Meijer al respecto (GOTO Conferences, 2015b)-.

El *open source*, el aprendizaje en red y GitHub

Existe una última característica de los proyectos *open source* que no se ha considerado todavía: la del aprendizaje dual.

Por un lado, está la construcción del conocimiento que hace cada uno al resolver un problema y contribuir a una solución -se ha hablado hasta ahora de este aspecto-. Pero por otro, la implicación en un proyecto de este estilo abre la puerta a un tipo de aprendizaje mucho más general y propio del siglo XXI: el **aprendizaje en red** (Illich, 1970; Siemens, 2005). Al participar en uno de estos

proyectos se está accediendo a una red potencial de 24 millones de programadores (GitHub Inc., 2018t). Y es aquí donde entra en juego la última pieza que conforma este TFM: GitHub, el hogar del *open source*.

Así se obtiene la respuesta que permite combinar todas las características del aprendizaje -continuo, iterativo, estratégico, pero profundo- que se enumeraban al principio: unirse al desarrollo de un proyecto *open source*. Y es, precisamente, esa manifestación del ***aprender a aprender*** (Orden ECD/65/2015) la que se debería intentar trasladar al aula.

1.1.3 La propuesta

En este TFM se destilará una propuesta, basada en todo lo expuesto en esta introducción, para ser aplicada en un entorno de formación reglada cómo es la Formación Profesional (FP). Al fin y al cabo, si estos métodos son válidos para los profesionales web, ¿por qué no van a ser igualmente válidos para los que aspiran a serlo?

Con esto en mente, se propondrá una innovación para el módulo profesional **MP0612** *Desarrollo web en entorno cliente* del **Título LOE SIFC03** *Desarrollo de aplicaciones web* en la Comunidad Autónoma de Galicia. Dicha innovación se basará en una traslación de la cultura del desarrollo software *open source* al aula, usando GitHub como centro neurálgico y buscando inculcar una filosofía de amor por el código cercana a *The Hacker Way*.

El reto es importante ya que, para empezar, no existen apenas escritos y evidencias académicas que documenten estos aspectos. Los profesionales que contribuyen a dichos proyectos comparten su experiencia, no a través de *papers* o escritos académicos, sino como parte de las discusiones que se establecen en el seno de los propios proyectos.

Sin embargo, al plantearse que los futuros profesionales web van a tener que integrarse en esta industria con condicionantes tan cambiantes, parece lógico inferir la necesidad de traer algo de la gestión del aprendizaje en la misma a la formación reglada, o utilizando un concepto mucho más cercano al objeto de este TFM, ***aprender a aprender*** (Orden ECD/65/2015). De esta forma se intentaría que la formación reglada no se sitúe tan fuera de la realidad que acabe por ser más un impedimento que una facilitadora en la formación de profesionales, mitigando el riesgo de ofrecer una educación sintética y alejada de la realidad (Díaz Barriga y Hernández, 2002, citado en Díaz Barriga, 2003).

Se finaliza esta introducción con una cita del propio Mark Zuckerberg (2012) de su ya famosa carta a los inversores en la oferta inicial de acciones de Facebook y donde daba a conocer *The Hacker Way*: "***Hackers believe that something can always be better, and that nothing is ever***

complete" (p. 69). Si ésta no es la síntesis perfecta de liquidez, curiosidad y aprendizaje, debe acercarse mucho.

1.2 Objetivos

1.2.1 Objetivo principal

Realizar una propuesta de intervención que mejore la competencia de aprender a aprender de los alumnos del módulo profesional *Desarrollo web en entorno cliente* mediante la introducción de la plataforma de aprendizaje y gestión del código GitHub, promoviendo un aprendizaje en red que mimetice un entorno real de trabajo como el usado en el desarrollo software de los proyectos *open source* más exitosos.

1.2.2 Objetivos específicos

1. Favorecer el trabajo en equipo en una red de iguales.
2. Fomentar una actitud emprendedora y una mentalidad que prime la solución de problemas.
3. Inculcar la filosofía *The Hacker Way*, poniendo en práctica el mantra "*code wins arguments*" (Zuckerberg, 2012, p. 69).
4. Seleccionar prácticas del *post-agile* para disminuir la fricción existente entre el entorno en el que se desarrollan los proyectos *open source* que se toman como modelo y la formación reglada, con un especial énfasis en la evaluación**.**
5. Explicitar la responsabilidad compartida en la confección, mantenimiento y actualización de la documentación.
6. Favorecer una nueva cultura alrededor de la práctica profesional que impida los efectos más nocivos causados por el *Imposter Phenomenon* o *Impostor Syndrome* (Clance e Imes, 1978).

2 Marco teórico

2.1 Justificación bibliográfica

Este TFM pretende trasladar a las aulas una visión realista del aprendizaje de los profesionales del desarrollo de software, usando sus mismas técnicas y herramientas. Se trata de una propuesta un tanto arriesgada para un proyecto de estas características pues no abunda, precisamente, la documentación al respecto y la existente no siempre pasaría un filtro académico riguroso, tal y como ya se dejaba constancia en la introducción de este TFM. Sin embargo, no por ello debe ser desechado, ya que las prácticas, las técnicas, los enfoques, las metodologías, están ahí esperando a que alguien se anime a destilarlas. Con este TFM se espera contribuir, modestamente, a esa labor.

Teniendo en cuenta que **GitHub** es la pieza central de este TFM, las referencias a repositorios, organizaciones o discusiones sobre código son constantes. Algunas veces, ante la falta de escritos académicos sobre algunos de estos aspectos, se utilizará en su lugar referencias a ejemplos concretos en GitHub.

También se citan algunas charlas paradigmáticas del programador, diseñador de lenguajes y educador **Erik Meijer**. Alguien con la suficiente estatura como para, inmediatamente después de abandonar *Microsoft*, contribuir significativamente en *Facebook* (con *Hack*), *Netflix* (con *RxJava*) y *Google* (con *Dart*) durante el año siguiente (GOTO Conferences, 2015b). Pocas personas pueden afrontar un reto semejante. Además, ha organizado e impartido un *Massive Online Open Course* (MOOC) extraordinario (edX, 2018) (el autor de este TFM tuvo la fortuna de asistir).

Para **aprendizaje en red** se toma como referencia casi única a Ivan Illich (1970). Sus ideas son tan poderosas y están tan bien descritas que más que un ensayo sobre aprendizaje podría parecer un relato con tintes distópicos de Phillip K. Dick. También se utiliza a Siemens (2005) por su proximidad en el tiempo y por la relevancia que tiene el **conectivismo** en nuestra sociedad siempre conectada. Pero es Illich quién debe ser considerado ***la*** referencia en **aprendizaje en red**.

Se utiliza *Ruby on Rails* (Ruby on Rails, 2018b) como ejemplo de *open source*, por su historia (*Twitter*, originalmente, se programó sobre *Ruby on Rails*) y por disponer de una comunidad suficientemente desarrollada. Sin lugar a dudas, se trata de uno de los proyectos *open source* más exitosos de los últimos 15 años, que ha catapultado a su creador, David Heinemeier Hansson, a la categoría de icono pop.

Para caracterizar "*agile is dead*" se cita a Dave Thomas (Thomas, 2014; GOTO Conferences, 2015a), responsable del término, mientras que para *The Hacker Way*, se recurre a su carta fundacional, responsabilidad del mismísimo Mark Zuckerberg (2012). Además se tomará en consideración la sapiencia del *enfant terrible* Erik Meijer (Reaktor, 2014; GOTO Conferences, 2015b), uno de sus más acérrimos defensores.

Se maneja más bibliografía relevante, pero su presentación y justificación sirve, muchas veces, de introducción a los temas pertinentes, por lo que no se nombra aquí.

2.2 Marco legislativo

Es en el Real Decreto 686/2010, de 20 de mayo, en el que se establece el carácter oficial del título de *Técnico Superior en Desarrollo de Aplicaciones Web*. En consonancia, en la Comunidad Autónoma de Galicia se define el módulo profesional **MP0612** *Desarrollo web en entorno cliente* -para el que se propone la innovación en este TFM-, que se engloba dentro del **Título LOE SIFC03** *Desarrollo de aplicaciones web* a través del Decreto 109/2011, en las páginas 14674 a 14682.

El módulo profesional MP0612, y 4 módulos más, están programados en el segundo año del Título LOE SIFC03, siendo impartidos entre los meses de septiembre a junio. Una vez superados, los alumnos podrán acceder a los 2 módulos finales de prácticas y de proyecto, que les permitirán obtener el Título LOE SIFC03 (Xunta de Galicia, s.f.). Para el módulo MP0612 serán destinadas 157 horas lectivas (Decreto 109/2011).

En el Decreto 109/2011 se expone que recae en la *Consellería de Educación y Ordenación Universitaria* la responsabilidad de la adaptación del currículo al contexto productivo, necesidades y realidad socioeconómica gallega. También se destaca que la adaptación debe posibilitar "una inserción laboral inmediata y una proyección profesional futura" (Decreto 109/2011, p. 14610).

Con esta última cita en mente, se desarrolla la propuesta de innovación de este TFM, siendo la inmediata empleabilidad -que podría asimilarse a la competencia LOMCE de "*sentido de la iniciativa y espíritu emprendedor*" (Orden ECD/65/2015, pp. 6999-7000)- y la capacidad para mantener el conocimiento construido actualizado -en clara sintonía con la competencia LOMCE de "*aprender a aprender*" (Orden ECD/65/2015, pp. 6997-6998)- los objetivos últimos que se persiguen, en total consonancia con la legislación vigente.

En el Título LOE SIFC03 se hace hincapié en la importancia de la "prevención de riesgos laborales" (Decreto 109/2011, p. 14611). Este aspecto se alinea perfectamente con uno de los objetivos específicos establecidos en este TFM: "Aprender a lidiar con el *Impostor Syndrome* (Clance e Imes,

1978)", una de las enfermedades silenciosas -por estar relacionadas con la mente- que está creando verdaderos estragos en la profesión (Hanselman, 2011; Bort, J., 2014), siendo incluso más devastadora entre las mujeres (Liu, 2013). Para ello se busca habituar al alumnado a trabajar sin conocer previamente todos los conceptos necesarios para la resolución del problema que se tenga entre manos. Una correcta gestión de este aspecto es de vital importancia para poder sobrevivir ante la continua demanda de reinvención que se ha intentado contextualizar en el *epígrafe 1.1.1 El software, la industria y el contexto* de este TFM.

2.3 El *open source* y su metodología de desarrollo

2.3.1 Origen del *open source*

Existe cierto consenso sobre el momento fundacional del *open source* (Neary, 2018). Acontece a finales de los 70 y principios de los 80 del siglo pasado, cuando Richard M. Stallman, en uno de los primeros *hacks* sociales documentados, programa el mensaje: "*La impresora está atascada. Por favor, arréglalo*". Con ello, buscaba informar a los usuarios de la impresora compartida que usaban en su departamento del MIT (*Massachusetts Institute of Technology*) del porqué de la tardanza, con la esperanza de que arreglasen el impedimento.

La situación se complica inesperadamente -ya entrados los 80- cuando una actualización del *driver* de la impresora y, lo más importante, un cambio en la licencia de distribución del mismo, hacen imposible que el propio Stallman pueda reproducir el *hack* social. El problema subyacente era que algunos de los investigadores del MIT, que previamente compartían el código de casi todo lo que hacían sin pensárselo dos veces, empezaron a montar empresas de software, y con ellas llegaron los NDA (*Non-Disclosure Agreements* o contratos de confidencialidad).

Ante esta situación, Stallman, decide crear un sistema operativo completo con una licencia que impidiese una situación como la que se acaba de relatar. Había nacido el *open source* (Neary, 2018).

2.3.2 La metodología de desarrollo del *open source* (MOS)

Avanzando unas cuantas décadas, y ya con el objetivo de este TFM en mente, se hace necesaria una reflexión sobre los proyectos *open source*.

En este TFM, se defiende la necesidad de trazar una línea divisoria que considere los dos enfoques -o puntos de vista- posibles al contribuir a un proyecto *open source* con cierto impacto social (de entre los miles que se encuentran alojados en GitHub en la actualidad):

- El punto de vista del creador del proyecto y de los *early adopters*.
- El punto de vista de los cientos de desarrolladores que contribuyen durante la vida útil del proyecto.

Al ponerse en la piel del creador -y de los que se animan desde un principio a contribuir- se entiende que lo único relevante es el proyecto en sí. El objetivo es sacar las primeras versiones *alpha* o *beta* cuanto antes, sin importar cómo. Todo es pensado a través de la óptica del proyecto: utilidad, relevancia social, esfuerzo dedicado, etc. Si fuese necesario pensar en una técnica de aprendizaje que se asemejase a lo explicado, se podría proponer el aprendizaje basado en proyectos (ABPr) (Barrows, 1996).

Sin embargo, es necesario plantearse la siguiente cuestión, ¿cuantos desarrolladores se encuentran en la situación de liderar un proyecto *open source* famoso o haber contribuido al mismo, de forma significativa, en sus primerísimas etapas?

Atendiendo a un proyecto *open source* de éxito contrastado como *Ruby on Rails* (2018b), con más de 38.000 estrellas y la nada despreciable cifra de 3.465 personas diferentes contribuyendo a su código (GitHub Inc., 2018s), se estaría hablando de:

- 12 personas en el *core team* -incluyendo al creador, David Heinemeier Hansson-.
- 7 en el *commiter team*.

Es decir, se encontrarían en esta situación un total de 19 personas de las 3.465 que han contribuido (Ruby on Rails, 2018a). La proporción, por lo tanto, es despreciable.

Pero ¿cuál es la realidad de los que contribuyen a un proyecto durante unos meses, luego saltan a otro que les parece más interesante, para finalmente cambiar a un tercero pues las prioridades, desde el punto de vista del aprendizaje, han vuelto a cambiar una vez más?

En ese caso se debería analizar cómo se producen esas contribuciones (y aquí GitHub juega un papel determinante). Entendiendo que cada proyecto tiene sus propias normas a la hora de contribuir código -a veces incluso de forma explícita con guías creadas *ex profeso* para ello (Rails Guides, 2018)-, la mecánica se asemeja mucho a (Frontend Masters, 2016):

1. Un usuario abre una incidencia en forma de *Issue* en la página de GitHub respectiva. Es decir, se plantea un problema en un espacio predeterminado para esos menesteres.
2. Se produce una discusión sobre la naturaleza de esa incidencia en la propia *Issue*.
3. Se clasifica -simplificándolo mucho- como *bug* (fallo) o nueva funcionalidad.

4. Se arregla el *bug* o se añade la nueva funcionalidad a través de uno o varios *Pull Requests* en la página de GitHub habilitada al respecto. Es decir, se contribuye el código que va a acabar formando parte del proyecto.

Si se elimina toda la mística asociada a GitHub y el código, la mecánica recuerda a:

1. Se plantea un problema.
2. Se debate el problema.
3. Se clasifica el problema.
4. Se resuelve el problema.

Desde el punto de vista del análisis que se está realizando en este epígrafe, el grado de participación de la persona que contribuye al proyecto puede variar en los 3 primeros puntos, pero claramente es determinante en el punto 4. Por lo tanto, si -de nuevo- se tuviese que pensar en una técnica de aprendizaje que asimilase esta experiencia, ya no tendría por qué ser ABPr -como en el caso del creador del proyecto-. Es más, aprendizaje basado en problemas (ABP) (Barrows, 1986) podría ser una opción clara, teniendo en cuenta su idea de usar problemas como "denominador común" (p. 20) "en el proceso instructivo" (p. 20).

Es decir, de las dos visiones o puntos de vista que se pueden usar para entender el proceso mediante el cual **aprenden** los desarrolladores **profesionales**, en este TFM se usará el que más se asemeja al ABP, denominándolo ***Metodología de desarrollo del Open Source*** o **MOS**.

2.4 Aprendizaje en red

La teoría del aprendizaje en red tiene su origen a comienzos de la década de los setenta del siglo pasado. Illich (1970) ya defendía entonces la necesidad de una revisión radical en la concepción misma de la escuela -y por ende, del proceso de aprendizaje-, abogando incluso, por un desmantelamiento de la misma.

Teniendo en cuenta que por aquel entonces ARPANET a duras penas empezaba a dar sus primeros pasos (Featherly, 2016) y que el concepto de ordenador personal tardaría otra década en ver la luz y popularizarse con el Apple II (Computer History Musseum, 2018a) y el ordenador personal (PC) de IBM (Computer History Musseum, 2018b), se puede argumentar que una de sus propuestas más visionarias tiene que ver con lo que él denominó "*Learning Webs*" (Illich, 1970, p. 72) o **redes de aprendizaje**. Según Illich (1970), ante la frustración mutua que sentían tanto docentes como alumnos a la hora de entregar los trabajos para conseguir diplomas y acreditaciones, se hacía necesario un estilo de aprendizaje diferente. La idea propuesta se puede resumir en las siguientes **tres condiciones**:

1. Facilitar el acceso al material de estudio a cualquiera que se proponga aprender.
2. Habilitar la comunicación entre quienes quieran compartir su conocimiento y quiénes quieran aprenderlo.
3. Promover un espacio en el que cualquier persona que desee explicar un problema pueda ser escuchada.

Los canales que facilitarían la materialización de las tres condiciones serían las redes de aprendizaje implementadas sobre una "tecnología moderna" (Illich, 1970, p. 76).

Es también interesante la reflexión de Illich (1970) sobre las nuevas instituciones educativas que era necesario que surgiesen. Según su punto de vista, no deberían basar el concepto de aprendizaje en la pregunta *¿qué debe aprender un estudiante?* sino en *¿a qué clase de recursos y con qué gente debe ponerse en contacto alguien para aprender?* Con este planteamiento, Illich (1970) defiende la necesidad de que la crítica provenga tanto de compañeros que están a su vez aprendiendo, como de voces más autorizadas. En cierta forma se podría afirmar que apostaba por una descentralización de la educación sin descuidar la idea *vygotskyana* de lo social en el aprendizaje (Vygotsky, 1979).

Para materializar esta visión, Illich (1970) propone **cuatro tipos diferentes de redes** (entendidas como *lugares en los que intercambiar aprendizaje*):

- **Catálogos de material educativo**: Que facilitarían el acceso al mismo, ya fuesen cosas o procesos.
- **Intercambios de habilidades**: Que facilitarían a los miembros la posibilidad de listar sus habilidades, las condiciones bajo las cuales estarían dispuestos a servir de modelos para aquellos que quieran aprender sobre ello y dónde se podrían poner en contacto.
- **Encuentro de iguales**: Redes de comunicación a través de las cuales se pueda explicar los intereses de aprendizaje con la esperanza de encontrar compañeros con similares inquietudes.
- **Catálogos de educadores en general**: Que contendrían la dirección de contacto, una breve autodescripción y las condiciones bajo las cuales se puede acceder al servicio proporcionado por los educadores, ya fuesen profesionales o *freelancers*.

Algunas de las ideas sugeridas por Illich (1970) (tecnología, libre acceso a los recursos, conexiones entre miembros de una red) reaparecen, ya entrado el siglo XXI, en Siemens (2005) (tecnología y conexiones en una red). Es interesante notar que en 2005 Internet ya se había colado en casi todos los hogares y que la tecnología *peer-to-peer* (p2p) era mundialmente famosa gracias a Napster en 1999 (Bloomberg L. P., 2000), eDonkey2000 en 2000 y BitTorrent en 2001. Por lo tanto, no es

descabellado afirmar que el concepto de red no centralizada, al menos para intercambio de ficheros, era común y conocido en 2005.

Es en este contexto en el que hay que situar la afirmación de Siemens (2005) sobre tecnología y establecimiento de conexiones en una red como actividades propias de la adquisición de competencias, mientras proponía su teoría de aprendizaje alternativa para la era digital: el **conectivismo**.

Siemens (2005) expresa la paradoja que supone la dificultad para experimentar en primera persona el aprendizaje teniendo en cuenta la ingente cantidad de información disponible en la red de redes; como si se tratase de una reacción ante la caótica naturaleza de internet. Además, sobre el concepto de competencia, afirma que alguien es más o menos competente en función de las conexiones que tenga en la red. Es decir, la competencia se deriva directamente del establecimiento de conexiones en un grupo. Tras esta afirmación de Siemens, se podría tener la impresión de que Ivan Illich vivió 35 años adelantado a su tiempo.

2.5 Aprendizaje en red y *open source* en GitHub

Si tomamos como referencia las 3 condiciones que establecía Illich (1970) para definir las redes de aprendizaje, sorprende comprobar la exactitud con la que parece definir todo el fenómeno de la explosión del *open source* alrededor de GitHub acontecido 40 años más tarde.

El *open source* se desarrolla en público, por lo que la mayor parte de los recursos y material de estudio necesarios están disponibles (**condición 1 de Illich**). Además, para poder resolver los problemas que plantea el *open source*, se establece una discusión en la que es común que se produzca una interacción entre gente que sabe cómo resolver una pequeña parte del puzle y gente que quiere aprender a resolver esa misma parte (**condición 2 de Illich**). Por último, cuando cualquier usuario de una librería *open source* desea informar de un problema al resto de la comunidad, proponer una nueva funcionalidad o explicar una mejora, puede hacerlo a través de una *Issue* en GitHub (**condición 3 de Illich**).

Por otra parte, cuando Illich (1970) propone sus **cuatro tipos de redes** -y recordemos de nuevo que, para Illich, red es sinónimo de *lugar en el que se intercambia aprendizaje*- un concepto que flota sobre ellas es la necesidad de establecer una **identidad dentro de la red** que permita a sus usuarios ser buscados y contactados. Este concepto es una realidad en nuestros días. Basta ver como en Twitter los usuarios mantienen un *@MiNick* que les permite identificarse y al mismo tiempo ser interpelados -generalmente recibiendo algún tipo de aviso ante una *mención*-. Este mecanismo es el mismo que pone a disposición de sus usuarios GitHub, copiando incluso el formato *@MiNick*.

Por lo tanto, no es para nada descabellado afirmar que el modo en que se desarrolla *open source* en GitHub es un ejemplo de aprendizaje en red según Illich (1970). Es más, teniendo en cuenta que en 2017 GitHub alcanzó la impresionante cifra de 24 millones de usuarios/desarrolladores (GitHub Inc., 2018t), probablemente estemos ante una de las redes más numerosas dedicada primordialmente al aprendizaje (la idea de que desarrollo *open source* y aprendizaje son intercambiables es vital para la innovación propuesta).

2.6 GitHub

GitHub se erige como la herramienta sobre la que pivota la innovación propuesta en este TFM. Ha sido nombrado varias veces y se ha comentado muy por encima alguna de sus características, sobre todo en su vertiente social y de red. Sin embargo, es en este epígrafe donde se proporcionará una descripción más formal y exhaustiva, haciendo un especial hincapié en la propuesta de López-Pellicer, Béjar, Latre, Nogueras-Iso y Zarazaga-Soria (2015) y el estudio de Zagalsky, Feliciano, Storey, Zhao y Wang (2015).

2.6.1 Qué es GitHub

GitHub es ***la*** plataforma de colaboración preferida de los desarrolladores de *open source*, albergando más de 67 millones de proyectos diferentes (GitHub Inc., 2018t). A estos proyectos se los conoce en el ámbito de la informática como **repositorios** o **repos** (Beer, 2018) pudiendo ser públicos (de acceso general para cualquiera que quiera consultarlos) o privados (acceso restringido a los usuarios que el autor del proyecto decida). A lo largo de este TFM se usará indistintamente las palabras proyecto o repositorio para referirse a un proyecto alojado en GitHub, pero generalmente se favorecerá el uso de proyecto cuando se busque resaltar la acepción más general, dejando repositorio para enfatizar el aspecto más técnico.

Es importante comprender que GitHub es mucho más que un simple portal al que se pueden subir proyectos. En el **Anexo B** se incluyen muestras que justifican la naturaleza poliédrica de GitHub, como plataforma de colaboración, red social, portafolio o indicador de competencia en la acepción de Siemens (2005).

En este TFM se usará una porción relativamente pequeña de las posibilidades que brinda GitHub, pero es muy importante destacar que las herramientas o funcionalidades usadas no son una simulación ni una versión simplificada, son las mismas que se van a encontrar los alumnos cuando se incorporen a la industria en el futuro.

La cantidad de términos técnicos necesarios para poder empezar a comprender qué es GitHub en su totalidad, literalmente, ocuparía al menos un libro. De hecho, *Introducing GitHub: A non-*

technical guide, 2nd Edition (Beer, 2018), podría ser ese libro (al menos la versión "*no técnica*" del mismo). Sin embargo, desde la perspectiva de este TFM, los requisitos necesarios -relativos a GitHub- para comprender la innovación propuesta son mucho más modestos. Para una explicación de algunos de los términos más técnicos como *Issue* y *Pull Request* (muy usados de aquí en adelante) se puede consultar el **Anexo A**.

2.6.2 Git y GitHub

Git (git-scm.com, 2018) es un sistema *open source* de control de versiones para software, creado por Linus Torvalds -autor también del núcleo de Linux-, y que se ha convertido en el sistema más usado por la comunidad *open source* (GitHub Inc., 2018t). Pese a la ausencia de datos fiables, derivado del oscurantismo imperante alrededor del software propietario, no es descabellado afirmar que, a través de GitHub, Git tiene también una enorme penetración en la industria del software comercial, a tenor del millón y medio de organizaciones que figuran en GitHub (GitHub Inc., 2018t) y de las cuales se supone que un buen número corresponde a empresas. De lo que sí hay datos es del 52% de empresas del Fortune 50 y 45% del Fortune 100 que usan Git, gracias a una versión especial de GitHub llamada GitHub Enterprise (GitHub Inc., 2018t). Por lo tanto, la relevancia de Git en la industria es incuestionable.

Sin embargo, la única razón para establecer la relación Git-GitHub en el marco teórico es que, en las entrañas de GitHub, habita Git. GitHub es, en cierta forma, su Interfaz Web (Web UI) por defecto.

Dicho esto, y contradiciendo la aproximación de López-Pellicer et al. (2015), no va a haber apenas referencias a Git en este TFM, pues, desde el punto de vista de la innovación propuesta, Git no resulta relevante y se corre el riesgo de complicar la lectura con términos y conceptos técnicos superfluos. Por esta misma razón se usa como referencia *Introducing GitHub: A non-technical guide, 2nd Edition* (Beer, 2018), al ser el texto sobre GitHub que menos menciona Git -y aun así se pueden encontrar referencias abundantes-.

Desde otro punto de vista: no debe confundirse que los alumnos del módulo profesional MP0612 sí vayan a tener que aprender Git y alguno de sus múltiples *workflows* (Chacon y Straub, 2014; Atlassian, s.f.; Thoughtbot Inc., 2016) con el hecho de que Git es simplemente un detalle técnico cuando se trata de entender y evaluar la innovación propuesta.

Para una aproximación a GitHub y la docencia mucho más centrada en Git, López-Pellicer et al. (2015) describen fielmente la mecánica y *workflows* existentes alrededor de esta herramienta. Algo esperable, por otra parte, al considerar que su ámbito profesional es el universitario -Departamento de Informática e Ingeniería de Sistemas-.

2.6.3 GitHub Classroom

Tal y como se ha establecido con anterioridad, el renacer del *open source* en esta última década tiene mucho de aprendizaje en red (Illich, 1970). Sin embargo, no parece que hubiese sido posible sin el nacimiento de GitHub en 2008 (Wanstrath, 2008). GitHub se ha convertido en el punto de encuentro *de facto* de la comunidad *open source*, el espacio donde los mejores desarrolladores del mundo, además de escribir código, discuten sobre el mismo.

Ya en sus inicios existía una potencialidad en GitHub como plataforma de aprendizaje (Wilson, 2011) -similar a la que se defiende y propone en este TFM-. Es más, esta noción ha acabado materializándose en **GitHub Classroom** (GitHub Inc., 2018j), una versión especial de GitHub impulsada y desarrollada por ellos mismos y utilizada por diversas universidades a nivel internacional. A saber: Harvard, Princeton, Stanford, Columbia, Calgary, Oslo, Stockholm, Kent, Oxford, Sun Yat-Sen, entre otras (GitHub Inc., 2018u).

Bajo el paraguas de la iniciativa *GitHub Education* (GitHub Inc., 2018k), **GitHub Classroom** puede ser usada en conjunción con servicios adicionales de empresas como Amazon AWS, Digital Ocean, Microsoft, Travis CI o UnReal a través del *Student Developer Pack* (GitHub Inc., 2018v).

De forma sintética, se puede entender **GitHub Classroom** como una plataforma sobre GitHub que (GitHub Inc., 2018j):

- Automatiza la creación de repositorios de código para los proyectos.
- Facilita el control de acceso o asignación de permisos.
- Simplifica las tareas de distribución del código inicial del proyecto, entrega de ejercicios y recolección de evidencias evaluables.
- Permite que los ejercicios puedan ser resueltos individualmente o en grupo, de forma pública o privada.
- Simplifica la labor de asignar estudiantes a los proyectos mediante enlaces de invitación, en contraste con la necesidad de mantener manualmente una lista de alumnos que, de alguna forma, deben ser asignados a los ejercicios correspondientes.
- Proporciona una herramienta, creada específicamente, para consultar los envíos de los alumnos según las diferentes tareas asignadas.

Por lo tanto, se puede concluir que **GitHub Classroom** es una capa extra sobre GitHub que permite a los docentes automatizar la parte más tediosa de su labor como educadores en lo relativo a la gestión de grupos, tareas propuestas y recolección de evidencias, sobre todo en MOOCs o cursos presenciales con gran número de alumnos y tareas individuales evaluables.

En el caso de este TFM, y teniendo en cuenta que toda la innovación se implantará con un único proyecto al que todos los alumnos estarán asociados, no parece razonable el trabajo extra que requiere introducir una herramienta más como es **GitHub Classroom**. Sin embargo, esto no debe impedir reconocer que **GitHub Classroom** es un paso claro en la dirección correcta y que, por otro lado, contribuye a reforzar la propuesta de este TFM de usar GitHub como plataforma de aprendizaje.

2.6.4 GitHub en la enseñanza

Debido a lo reciente del fenómeno, no existen tantas referencias académicas que relacionen GitHub y enseñanza desde la óptica que se persigue en este TFM, pese a lo afirmado por López-Pellicer et al. (2015), aunque con el advenimiento de *GitHub Classroom* (GitHub Inc., 2018j) da la impresión de que la situación podría cambiar en poco tiempo, al haber conseguido implantarse en algunas de las universidades más prestigiosas del mundo (GitHub Inc., 2018u).

López-Pellicer et al. (2015) no dejan lugar a dudas sobre la importancia de GitHub a la hora de gestionar proyectos software. Es en base a ello que deciden probar la plataforma como herramienta educativa aglutinadora de toda la parte práctica de la asignatura *Ingeniería Web* del grado de *Ingeniería Informática* en la Universidad de Zaragoza.

Una primera similitud con la experiencia de López-Pellicer et al. (2015) es el enfoque práctico que GitHub contribuye a dar a toda la asignatura/módulo. En su caso, la parte práctica supone un 80% de la nota. En este TFM, se intentará, además, que la construcción del aprendizaje por parte del alumnado surja a partir de la experimentación en los términos que ellos decidan (sin descuidar, en ningún momento, la consecución de los objetivos marcados por la legislación vigente).

Hay un aspecto extra que puede suponer una divergencia con respecto al enfoque de López-Pellicer et al. (2015). En la propuesta de este TFM se persigue el objetivo de trasladar la cultura y valores del *open source* al aula. Este aspecto nos lleva a transformar el módulo profesional MP0612 en una serie de *Issues* focalizadas en:

- Aprender en red un lenguaje de programación (*JavaScript* o *Elm*, por ejemplo).
- Desarrollar una serie de competencias (aprender a aprender, iniciativa y habilidades sociales).
- Satisfacer unas necesidades educativas estrictas marcadas por la administración.

Otra posible diferencia, ya esbozada con anterioridad, con López-Pellicer et al. (2015) es que ellos centran su experiencia en la parte más técnica (Git como herramienta), corriendo el riesgo de infravalorar la parte que sitúa a GitHub como uno de los ejemplos paradigmáticos de aprendizaje

en red (Illich, 1970; Siemens, 2005) de nuestros días. En este TFM, al trasladar las dinámicas de aprendizaje del *open source* al aula, se busca propiciar un contexto realista y con valor social en la senda de la cognición situada (Díaz Barriga y Hernández, 2002, citado en Díaz Barriga, 2003).

Conviene mencionar también que López-Pellicer et al. (2015) avisan de los problemas que puede acarrear la introducción de GitHub en el aula, tanto en la forma de "*aumento de las tareas de gestión docente*" (p. 67), como en que suponga un reto, *per se,* suficientemente importante para descarrilar el "*normal desarrollo del curso*" (p. 67). Estos dos aspectos deben ser tomados en consideración, ya que formarían parte de los riesgos que deben ser minimizados.

Desde una perspectiva diferente, Zagalsky et al. (2015), aciertan plenamente, al señalar la importancia de GitHub en la mejora de las rutinas de trabajo de los profesionales del software y ya desde un principio hablan de GitHub como "*plataforma colaborativa para educación*" (p. 1906) resaltando la parte social y de red de la misma, en total sintonía con lo defendido en este TFM.

Una de las conclusiones de Zagalsky et al. (2015) es la problemática que supone la "*empinada curva de aprendizaje*" (p. 1916) de GitHub para los alumnos, coincidiendo con los avisos de López-Pellicer et al. (2015). Esto reafirma la voluntad expresada de usar las herramientas mínimas necesarias, de entre todas las que proporciona GitHub, para poder llevar a cabo esta innovación.

Nuevamente, al usar el estudio de Zagalsky et al. (2015) como referencia, se descubre que la comunidad educativa ha utilizado GitHub para dos aspectos diferentes: entrega de tareas y repositorio de material relativo al curso. Se pone de manifiesto que este enfoque apenas difiere de cómo se usan los Learning Management Systems (LMS) como Moodle (Malikowski, Thompson y Theis, 2007, citado en Zagalsky et al., 2015). Sin embargo, en Zagalsky et al. (2015) se apuntan las posibilidades extra que ofrece GitHub con respecto a los LMS (ver *figura 1*).

Un último aspecto relevante de Zagalsky et al. (2015) es que ante la pregunta de si GitHub está preparado para saltar a la educación de forma generalizada, concluían que casi. Teniendo en cuenta que ya han pasado 3 años desde el estudio, y que la experiencia de *GitHub Classroom* (GitHub Inc., 2018j) es una realidad, parece que ahora sí es el momento de dar el salto.

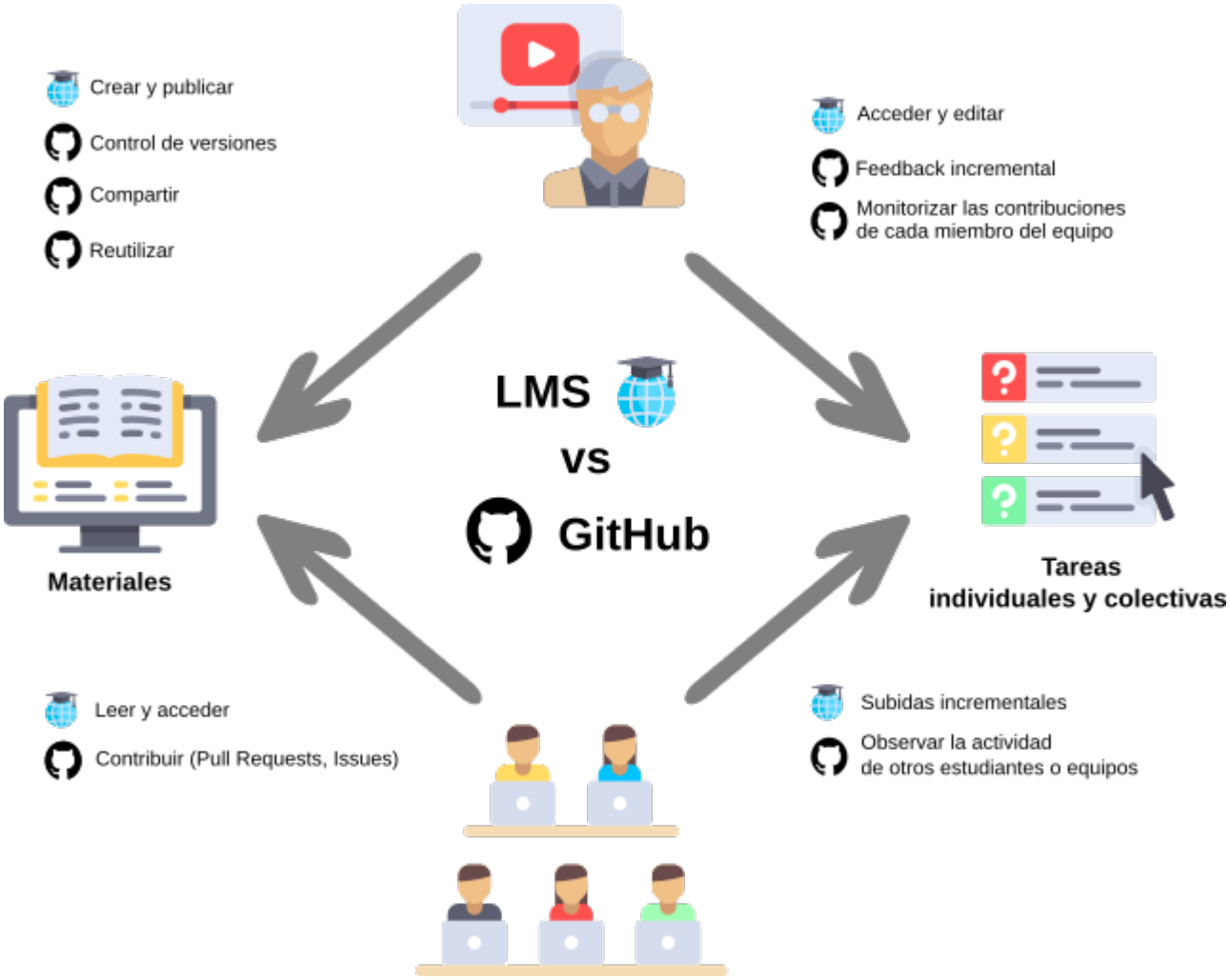


Figura 1: Posibilidades extra que ofrece GitHub frente a LMS tradicionales. Fuente: Elaboración propia basada en Zagalsky et al. (2015).

2.7 "*Agile is dead*"

"*Agile is dead*" es una de las corrientes más importantes dentro de ese contexto llamado *post-agile*, explicado en la introducción de este TFM.

2.7.1 Origen y motivación de "*agile is dead*"

Para poder entender el porqué de su nacimiento, es necesario retrotraerse a la redacción del *Manifesto for agile software development*. Según Dave Thomas (GOTO Conferences, 2015a), uno de los firmantes originales del *Manifesto*, los valores que se defendían se han ido perdiendo con el tiempo. La raíz del problema habría que buscarla en un cambio aparentemente inocuo: el *Manifesto for agile software development* pasa a ser conocido como *The Agile Manifesto*. Es decir,

"*agile*" pasa de adjetivo -como debe ser usado en inglés-, a nombre, permitiendo frases a todas luces incorrectas: "I'm doing Agile". Por si esto no fuese suficiente, empieza a escribirse en mayúsculas "como si se tratase de Dios" (GOTO Conferences, 2015a). Esto es determinante desde el punto de vista del marketing, ya que un nombre propio es infinitamente más fácil de vender que un adjetivo: "I can't sell *green* to you. I can't come to you and say: Here, have some *green*" (GOTO Conferences, 2015a).

Por lo tanto, atendiendo a la opinión de Thomas (GOTO Conferences, 2015a), "*agile is dead*" nace como reacción a esta mercantilización y deformación de los valores defendidos en el *Manifesto*.

2.7.2 Qué es y por qué "*agile is dead*"

Se podría argumentar que "*agile is dead*" supone un nuevo comienzo para el *agile* de forma que, desde una perspectiva crítica, se puedan recuperar los valores fundacionales del *Manifesto*. Para ello, Thomas (2014) reduce casi quince años de experiencia a un ciclo de tres pasos que deben seguir los desarrolladores de software:

1. Entender en qué situación se encuentran.
2. Dar un pequeño paso hacia su objetivo.
3. Evaluar que ha ocurrido.
4. Volver al punto 1.

Al implantar MOS en el aula hay dos áreas para las que el *open source* carece de herramientas:

- Aflorar problemas e impedimentos ocultos.
- Evaluar.

Es aquí donde "*agile is dead*" resulta muy atractivo, ya que brinda la oportunidad de revisar todo el *agile* y seleccionar las herramientas más útiles en función de la necesidad. En concreto, al diseñar las **Actividades 3 y 5** de la intervención, así como para justificar una agrupación en parejas, se toma inspiración de 3 prácticas contrastadas del *agile* descritas en el **Anexo D**.

2.8 *The Hacker Way*

The Hacker Way sería la otra corriente, dentro de ese *totum revolutum* denominado *post-agile,* que se utiliza en este TFM. Estamos ante la filosofía empresarial de Facebook; cinco puntos, convertidos en mandamientos, que toda *startup* tecnológica que se precie debe obedecer.

2.8.1 Origen y adopción de *The Hacker Way*

The Hacker Way parte de Facebook como compañía, pero se oficializa con la carta a los inversores del propio Zuckerberg (2012) en su salida a bolsa. Ante la pérdida de los valores originales del *agile* y su reconversión en industria pura y dura (GOTO Conferences, 2015a), se hacía necesaria una nueva filosofía que en vez de potenciar hablar sobre código, se pusiese a escribirlo -"***We are developers. We write code. We don't talk about code***" (Reaktor 2014)-, y es ahí donde *The Hacker Way* parece haber encontrado su nicho -un buen número de proyectos *open source* siguen esta filosofía-.

Sin embargo, en Europa, su adopción está muy lejos de los números de los que puede presumir la industria del *Agile*, debido, según Meijer (GOTO Conferences, 2015b), al terror que sienten los *managers* a que los programadores reclamen su verdadero protagonismo, además de la alergia que tienen, por definición, al riesgo -uno de los valores propuestos también por *The Hacker Way*-. Más allá de la aparente categoría de *boutade* con la que Meijer, a veces, adorna sus afirmaciones, es necesario entender la matización que hace a través de la frase "deberíamos tratar a los desarrolladores como reyes, pagarles salarios de millones de dólares, como si fuese atletas profesionales (en referencia a Messi)" (GOTO Conferences, 2015b). Probablemente ahora sea más sencillo entender a qué se refiere Meijer cuando habla del terror de los *managers*.

2.8.2 Qué es y por qué *The Hacker Way*

Se caracterizaba a *The Hacker Way* en la introducción como la alternativa al *agile* surgida desde el corazón de Silicon Valley, en concreto en el número 1 de la calle Hacker -*1 Hacker Way* o *One Hacker Way*, nótese el juego de palabras- de Menlo Park, hogar de Facebook (TechCrunch, 2011). La esencia que el *Manifesto for agile software development* capturó, el *Agile* convirtió en industria y deformó (Thomas, 2014), *The Hacker Way* la devuelve al punto de vista del desarrollador (Reaktor 2014).

Más allá de su mantra "*code wins arguments*", la filosofía subyacente a *The Hacker Way* se resume en los siguientes 5 valores (Zuckerberg, 2012):

- ***Focus on Impact***: Para poder dejar huella, una compañía debería enfocar su actividad a la resolución de los problemas más relevantes. Meijer (GOTO Conferences, 2015b) va más allá y apostilla que la responsabilidad de un desarrollador para con su compañía es, únicamente, encontrar y resolver el problema más importante.
- ***Move Fast***: Una empresa que se mueve con rapidez construye más cosas y por lo tanto aprende más rápido. Esta idea es una de las claves para entender por qué *The Hacker Way* es tan interesante desde el **punto de vista de la educación**. Meijer (GOTO Conferences,

2015b) pone la lupa sobre el "*Move fast and break things*" de Zuckerberg (2012, p. 70), traduciéndolo en la necesidad que tiene un programador de **aprender constantemente**. Pero es imposible aprender de forma relevante algo relacionado con el código simplemente hablando de ello; es necesario escribirlo, experimentar con él. Siguiendo ese espíritu, en este TFM, se propone el uso del REPL como herramienta de experimentación para hablar de código escribiendo código (ver **Actividades 2 y 6** de la intervención).

- ***Be Bold***: Para una empresa es preferible equivocarse a no tomar riesgos ya que no se va a equivocar todo el tiempo. Meijer (GOTO Conferences, 2015b) vuelve a ir más allá al afirmar que "en un mundo que cambia con tanta rapidez" -nótese la similitud con la liquidez de Bauman (2000)- está garantizado fallar, precisamente, si se evitan los riesgos. Esto podría plantear la siguiente pregunta, ¿fallar hasta acertar -con un método y una orientación clara hacia la solución de un problema- no es acaso una forma de aprender? En otras palabras, ser curioso, ser osado, atreverse, no tener miedo a experimentar, aprender jugando, aprender haciendo.
- ***Be Open***: Facebook intenta que todos sus empleados tengan el mayor nivel de acceso a la información posible. Uno de los puntos más evangelizadores y discutibles -en cuanto a la honestidad- de *The Hacker Way*. Sin embargo, más allá de los juicios de valor, este punto está en clara consonancia con algunas de las premisas y afirmaciones de Illich (1970) que se explicaron al hablar del aprendizaje en red.
- ***Build Social Value***: Se espera que Facebook aporte valor real y social al mundo, con independencia de ser una compañía. El otro punto discutible -en cuanto a su honestidad- de *The Hacker Way*. Aun así, se alinea de forma casi perfecta con la teoría de la cognición situada. Cuanto más valor social, menos riesgo de artificio en lo aprendido, evitando la descontextualización en lo enseñado (Díaz Barriga y Hernández, 2002, citado en Díaz Barriga, 2003).

Finalmente, resaltar que la importancia de todo lo expuesto en este epígrafe se fundamenta en dos hechos diferenciados. Por un lado "*el software se está comiendo el mundo*" (Reaktor 2014). Por otro, la esencia del *hacker* es la esencia del que aprende sin descanso.

3 Propuesta de intervención

3.1 Presentación

Tal y como se expuso en la **introducción** de este TFM, el reto del aprendizaje en el contexto de los lenguajes de programación -y más en concreto en el ámbito del *frontend*- es especialmente complejo. También en la **introducción** se establecía que los profesionales de la programación web aprendían a través de su implicación en diversos proyectos *open source*, estableciendo una suerte de equivalencia entre aprendizaje y desarrollo de *open source*.

Ya en el **marco teórico** se desarrollaron **tres ejes** sobre los que hacer pivotar este TFM: la *Metodología de desarrollo del Open Source* (MOS), el aprendizaje en red y GitHub. Además, se enunciaron algunas de las ideas asociadas a las dos corrientes más interesantes del *post-agile*: *The Hacker Way* y "*agile is dead*".

En la propuesta de intervención que se desarrollará en este epígrafe 3 se combinarán todos los elementos citados y se detallará una metodología *ad hoc* para el módulo profesional **MP0612** *Desarrollo web en entorno cliente* del **Título LOE SIFC03** *Desarrollo de aplicaciones web* en la Comunidad Autónoma de Galicia, que debería abordar todos los problemas señalados hasta el momento y cumplir los objetivos que motivan este TFM.

3.2 Contextualización

3.2.1 El centro educativo y su entorno

Se procederá con la implantación de la innovación propuesta en un centro de Formación Profesional de una importante ciudad de Galicia. Se trata de un centro privado situado a las afueras de la ciudad. La oferta educativa que ofrecen es bastante amplia. Incluye Enseñanza Secundaria Obligatoria (ESO), Bachillerato y diversas ramas de Formación Profesional (FP) orientadas, principalmente, a la atención sanitaria y las TIC.

Al encontrarse situado a las afueras de la ciudad, se disfruta de un ambiente rural en las inmediaciones, carencia de ruidos y distracciones. El edificio principal data de principios del siglo pasado, y está dedicado en su totalidad a labores administrativas, ESO y Bachillerato. Existen dos edificios más de los años setenta del siglo XX, en uno de los cuales se encuentran las aulas de FP de la rama TIC.

Las instalaciones están equipadas con las últimas tecnologías y todos los alumnos de la rama TIC de FP obtienen en préstamo, para todo el curso, un ordenador portátil completamente configurado para que tomen notas en clase, realicen prácticas y trabajos. Una de las ventajas que proporciona este hecho es que no es necesario habilitar un aula informática de prácticas (aunque hay 3). Todas las aulas donde se imparten las clases disponen de proyector y pantalla de proyección, de forma que los ejemplos presentados por el docente en su ordenador puedan ser vistos por todos los alumnos.

En la actualidad hay más de 60 personas en plantilla, de las cuales, la mayoría, se dedica a labores docentes. La rama TIC de FP del centro consta de unos 15 docentes y el ambiente que se respira es de colaboración y respeto. La media de edad es sorprendentemente baja, en torno a los 35 años.

La reputación del centro es muy positiva, siendo bastante conocido incluso fuera de la ciudad. Para mitigar su fama de elitista, hace 10 años que han abierto una línea de becas que cubren un amplio espectro social. Pese a ello, la mayor parte del alumnado procede de familias de clase media-alta.

Tanto en la rama sanitaria como en la rama TIC de FP, han establecido lazos sólidos con el tejido empresarial de la Comunidad Autónoma y son muy valoradas por los alumnos las ofertas de prácticas al finalizar los ciclos profesionales.

3.2.2 El alumnado

Se propone la innovación para el módulo profesional **MP0612** *Desarrollo web en entorno cliente* del **Título LOE SIFC03** *Desarrollo de aplicaciones web* ubicado en el segundo año del ciclo formativo. Por lo tanto, la gran mayoría de los alumnos se conocen del año anterior y el nivel, en cuanto a conocimientos informáticos, es bastante bueno.

Durante los últimos años, el número de chicas que se matriculan en ciclos relacionados con TIC en el centro ha crecido exponencialmente, hasta darse el caso de superar en número a los chicos en los dos últimos años. Este año, el grupo está compuesto por 10 chicas y 8 chicos.

El trato entre ellos es correcto y prima una actitud creativa y *geek*. Algunos profesores han constatado una cierta resistencia por parte de los alumnos a la hora de hacer grupo, pero nada que no hayan visto otros años y todos ellos coinciden en que cuando es realmente necesario, se ayudan. Algunos de los alumnos han manifestado su intención de emprender al finalizar el ciclo. Otros van a buscar trabajo en la industria. Finalmente hay un pequeño grupo (la mayor parte chicas) que se está planteando continuar sus estudios en la Universidad. Ninguno repite curso.

Los profesores están satisfechos con el trato que reciben por parte de los alumnos. Reconocen la dificultad de conectar con ellos cuando deben explicar conceptos alejados de la práctica y han recibido críticas constructivas en referencia al exceso de clases teóricas.

La mayoría de los alumnos tienen aficiones relacionadas con la informática. Tres de ellos tienen un grupo de música *trap*, dos han realizado trabajos de diseño web y tres más pertenecen a un grupo *maker* local.

3.2.3 Objetivos didácticos

Por un lado, hay que considerar que, en consonancia con los objetivos desarrollados en el Decreto 109/2011, se busca que la innovación propuesta facilite la construcción del conocimiento por parte del alumnado desde una perspectiva eminentemente práctica. Esto incluye la asimilación de los contenidos curriculares básicos (Decreto 109/2011, pp. 14679-14681) y la conformidad a los resultados de aprendizaje y criterios de evaluación (Decreto 109/2011, pp. 14675-14679) especificados por la ley.

Por otra parte, también se busca mimetizar la experiencia de desarrollo software usada por la comunidad *open source*, de forma que la futura transición al mundo laboral sea más fluida. Este último aspecto debería, además, facilitar la empleabilidad, ya fuese por cuenta ajena o por cuenta propia. Debe recordarse que algunos alumnos han manifestado su intención de emprender al finalizar el ciclo, por lo que proporcionarles herramientas y *workflows* que vayan más allá del currículo *per se*, debería resultarles útil.

Con la implantación de MOS se espera que los alumnos consigan:

(a) Aprender a utilizar GitHub como herramienta de gestión de versiones del código desde un punto de vista social, de red o simplemente colaborativo.

(b) Experimentar los efectos positivos del aprendizaje en red, desarrollando una intuición que les permita manejarse en una profesión donde la liquidez del conocimiento se experimenta a diario.

(c) Descubrir un flujo de trabajo grupal orientado a resolver problemas.

(d) Generar documentación que detalle la experiencia del aprendizaje y potencialmente sirva de referencia para otros.

(e) Acostumbrarse a indagar lo justo -enfoque estratégico- para poder resolver un problema, experimentando como algo natural no disponer de toda la información necesaria de antemano.

(f) Valorar la experiencia de vivir alrededor del código: discutir, aportar ejemplos y desarrollar.

(g) Perder el miedo a exponer en público el trabajo realizado.

(h) Estar abiertos a recibir críticas constructivas por parte de los demás.

Algunos de estos objetivos didácticos se relacionan directamente con los objetivos secundarios o específicos de este TFM, tal y como se muestra en la *tabla 1*.

Objetivos específicos	**Objetivos didácticos**
(1) Aprendizaje en red	(a), (b), (c), (d), (f) y (h)
(2) Actitud emprendedora	(e), (f) y (h)
(3) *The Hacker Way*	(f)
(4) *Post-Agile*	(g) y (h)
(5) Documentación	(d)
(6) Impostor Syndrome	(b), (e), (f), (g) y (h)

Tabla 1: *Relación entre objetivos específicos y objetivos didácticos*. Fuente: Elaboración propia.

Del objetivo principal del TFM, evidentemente, se derivan todos los demás, pero en concreto (a), (b), (c) y (f) están en total consonancia.

Al finalizar el módulo profesional **MP0612** es realista esperar que los alumnos estén suficientemente familiarizados con el trabajo en grupo, en red y en público como para:

- Poder unirse al proyecto *open source* que deseen -continuando con su aprendizaje y al mismo tiempo preparando un portafolio que incremente su valía como profesionales-.
- Incorporarse a la industria como programadores web, independientemente de la metodología de trabajo.

3.2.4 Competencias

La formación del módulo profesional **MP0612** en la Comunidad Autónoma de Galicia contribuye a alcanzar los siguientes objetivos generales especificados en el Decreto 109/2011, páginas 14618 a 14620 (y por tanto la innovación propuesta en este TFM contribuye a ello, a su vez):

- f) Seleccionar lenguajes, objetos y herramientas, e interpretar las especificaciones, para desarrollar aplicaciones web con acceso a bases de datos.

- g) Utilizar lenguajes, objetos y herramientas, e interpretar las especificaciones, para desarrollar aplicaciones web con acceso a bases de datos.
- i) Utilizar lenguajes de marcas y estándares web, asumiendo el manual de estilo, para desarrollar interfaces en aplicaciones web.
- q) Programar y realizar actividades para gestionar el mantenimiento de los recursos informáticos.
- r) Analizar y utilizar los recursos y las oportunidades de aprendizaje relacionados con la evolución científica, tecnológica y organizativa del sector, y las tecnologías de la información y de la comunicación, para mantener el espíritu de actualización y adaptarse a nuevas situaciones laborales y personales.

Asimismo, también contribuye a desarrollar las siguientes **competencias** profesionales, personales y sociales especificadas en el Decreto 109/2011, páginas 14613 a 14615:

- a) Configurar y explotar sistemas informáticos, adaptando la configuración lógica del sistema según las necesidades de uso y los criterios establecidos.
- e) Desarrollar aplicaciones web con acceso a bases de datos utilizando lenguajes, objetos de acceso y herramientas de mapeado adecuados a las especificaciones.
- k) Desarrollar servicios para integrar sus funciones en otras aplicaciones web, de modo que se asegure su funcionalidad.
- n) Elaborar y mantener la documentación de los procesos de desarrollo, utilizando herramientas de generación de documentación y control de versiones.
- p) Adaptarse a las nuevas situaciones laborales, manteniendo actualizados los conocimientos científicos, técnicos y tecnológicos relativos a su ámbito profesional, y gestionando su formación y los recursos existentes en el aprendizaje a lo largo de la vida, utilizando las tecnologías de la información y de la comunicación.
- r) Organizar y coordinar equipos de trabajo y supervisar su desarrollo, con responsabilidad, manteniendo relaciones fluidas, asumiendo el liderazgo y aportando soluciones a los conflictos que se presenten en los grupos.

En la Comunidad Autónoma de Galicia, el módulo profesional MP0612 se engloba dentro del Título LOE SIFC03 y está desarrollado en base a la Ley Orgánica de Educación (LOE) (Ley Orgánica 2/2006), siguiendo el Real Decreto 686/2010. Por lo tanto, las 7 **competencias clave** de la LOMCE (Orden ECD/65/2015) no habían sido especificadas todavía. Sin embargo, ya en la LOE se introducía el concepto de **competencias básicas** siguiendo la Recomendación 2006/962 EC, del Parlamento Europeo y del Consejo, de 18 de diciembre de 2006 (Orden ECD/65/2015). Con esto se quiere poner de manifiesto que el concepto de competencia tal y como se desarrolla en la LOMCE

ya se encontraba de forma indirecta en la LOE. Desde este punto de vista, con esta innovación se pretende **desarrollar de forma específica** las siguientes competencias clave (Orden ECD/65/2015):

- ***Aprender a aprender***: Los alumnos deben ser capaces de construir conocimiento en cualquier situación que se puedan encontrar, tanto en su vida laboral como en su vida personal. En este TFM se hace especial hincapié en el aprendizaje en red y el papel e importancia que tienen los nodos.
- ***Sentido de la iniciativa y espíritu emprendedor***: Los alumnos deben ser capaces de mostrar liderazgo en las situaciones que lo requieran, pero también es importante que sepan cuándo deben ceder ese protagonismo. En este TFM se trabajan estas cuestiones a nivel grupal, potenciando al máximo su creatividad y ofreciéndoles la oportunidad de experimentar todos los roles posibles dentro del grupo.
- ***Competencias sociales y cívicas***: Los alumnos deben ser capaces de formar parte de un grupo, siendo respetuosos y aprendiendo a manifestar desacuerdo sin que suponga un conflicto. En esta propuesta se trabaja este aspecto a diario, ya sea organizando el trabajo en las *Issues*, calificando y ofreciendo *feedback* sobre el trabajo de los demás o recibiéndolo sobre el propio.

Todo el Título LOE SIFC03 lleva implícita la ***competencia digital***. No tiene sentido, por tanto, ahondar más en ella al exponer la propuesta de innovación, ya que cada vez que estén utilizando una herramienta digital relacionada con la programación de forma ética, están trabajando la *competencia digital*, y eso se supone que va a ocurrir casi a diario en este módulo, independientemente de que se utilice MOS o no.

3.2.5 Resultados de aprendizaje, criterios de evaluación y bloques de contenido

Teniendo en cuenta que la propuesta de innovación de este TFM es aplicable a todo el módulo profesional **MP0612** y no únicamente a una unidad de trabajo, se reproducen:

- En el **Anexo E**, los siete resultados de aprendizaje (RA) con sus criterios de evaluación (CE) asociados según se desarrollan en el Decreto 109/2011, páginas 14675 a 14679.
- En el **Anexo F**, los siete bloques de contenido (BC) según se desarrollan en el mismo Decreto, páginas 14679 a 14681.

3.3 Metodología

El presente TFM se sustancia en una propuesta de innovación alrededor de la plataforma GitHub en función de cómo se organizan, aprenden y escriben código los desarrolladores de los proyectos *open source* más exitosos. En base a ello, se crea la ***Metodología de desarrollo del Open Source*** o **MOS**. Por lo tanto, la metodología que se propone es el propio objeto del TFM: la implantación de MOS en el módulo profesional MP0612 *Desarrollo web en entorno cliente* del Título LOE SIFC03 *Desarrollo de aplicaciones web* en la Comunidad Autónoma de Galicia.

3.3.1 Recapitulación de MOS

Se procederá, inicialmente, con una recapitulación sobre lo ya expuesto acerca de **MOS**.

En el **epígrafe 2.3 del Marco Teórico**, se hacía una breve descripción de MOS, explicando:

- Los orígenes del *open source*.
- El porqué del nacimiento del *open source*.
- Las diferentes perspectivas para analizar un proyecto *open source* de esta última década.
- Las coincidencias con técnicas de aprendizaje (ABP y ABPr) usadas en la educación, para poder anclar lo propuesto en experiencias extendidas en la docencia.

Al mismo tiempo en el **epígrafe 2.5 del Marco Teórico**, se establecía cuánto de **aprendizaje en red** hay en el desarrollo de *open source* moderno, mientras que en el **epígrafe 2.6** se explicaba la herramienta que ha facilitado todo este proceso: **GitHub**, que pasa por ser la pieza clave en la innovación propuesta.

Finalmente, en el **epígrafe 1.1.2 de la Introducción**, se recalcaba el aspecto que va a servir de *leit motiv* de todo este TFM: **si se aprende haciendo, el desarrollo de *open source* es una manera de aprender**.

3.3.2 Implantación de MOS en el aula

Para implantar **MOS** en el aula:

- Se explicarán los **pasos previos** (3.4) a seguir por el docente antes del comienzo del curso lectivo.
- Se detallarán **8 actividades** (3.5) que ilustran las dinámicas típicas de MOS en el aula.
- Se ofrecerán ejemplos de **temporalización** (3.6) que combinan las actividades.

- Se listarán los **recursos** (3.7) necesarios para cada actividad.
- Se planteará una posible **evaluación de alumnos** (3.8) justa y adecuada al espíritu práctico de MOS.
- Se analizará cómo **evaluar la propuesta** (3.9).

Se intentará seguir un orden cronológico de los acontecimientos (**pasos previos** y **actividades**), que facilite la utilización de estas secciones del TFM a modo de **guía** detallada para la **implantación**, con la pretensión de que así aumenten las posibilidades de éxito. A ello también contribuye la **temporalización** y **evaluación de alumnos**, donde se verá cómo relacionar y evaluar estas actividades de una forma coherente.

Pero antes debe ser tenido en cuenta, tal y como se relatará en el **Paso Previo 1** del epígrafe 3.4.1, que la implantación de MOS afectará a todo el módulo profesional **MP0612** y no sólo a alguna de sus Unidades de Trabajo (**UT**), por lo que **es necesario** contar con el apoyo del **departamento** correspondiente. Esto no significa que MOS no pueda ser usada exclusivamente en una o dos UT, sino que el esfuerzo necesario -tanto del docente como de los alumnos- para la implantación, sólo puede ser justificado de forma realista desde una perspectiva global a nivel módulo profesional.

3.3.3 MOS *Workflow* a nivel Unidad de Trabajo

Para hacerse una idea de cuál sería el flujo de trabajo típico que favorece MOS se debe considerar que:

- En el **Paso Previo 1** del epígrafe 3.4.1 se detallará la equivalencia entre UT e *Issue* -una UT por cada *Issue*-.
- En la **Actividad 7** se detallará el *workflow* o flujo de trabajo típico dentro de esa UT o *Issue*.

3.3.4 MOS y la atención a la diversidad

MOS se adapta extraordinariamente bien a alumnos con **diferentes capacidades de aprendizaje**. La propia naturaleza de la *Issue* permite que los alumnos se repartan el trabajo, por lo que, llegados al caso, cada pareja asume la cantidad de trabajo para la que se siente capacitada. Si existiese cualquier tipo de desequilibrio al respeto, el docente podrá intervenir en la *Issue* sugiriendo una repartición de tareas más acorde a la situación enfrentada. De todas formas, en aras de una completa integración de estos alumnos, se debería buscar que los problemas que surjan se debatan a nivel *Issue*. Sería una forma de naturalizar la situación aplicando una política de hechos consumados.

En caso de haber algún alumno con dificultades de aprendizaje derivadas de una discapacidad sensorial o motriz, no habría mucha diferencia entre usar MOS y cualquier otra metodología y tanto el centro como el docente deberían adaptarse a la situación.

3.4 Pasos previos

3.4.1 Paso 1 – Adaptar el currículo a MOS

El primer paso que debe dar el docente para la implantación de la innovación es adaptar el currículo especificado por la legislación vigente a la *Metodología de desarrollo del Open Source* (MOS), explicada brevemente en los **apartados 2.3 y 3.3** y desarrollada en las **actividades** que se plantean más adelante.

Debe ser tomado en consideración que, para poder llevar a cabo algo así, es necesario contar con la aprobación del **departamento** correspondiente, ya que, la responsabilidad no recae solo en el docente sino también en el departamento. En el caso especificado en este TFM, tras una reunión, se ha obtenido la aprobación por parte de todos los miembros, existiendo incluso un interés importante por ver si se cumplen los objetivos planteados, ya que esta metodología podría ser exportada a otros módulos del ciclo profesional.

Tras el análisis de los BC, RA y CE y tomar en consideración la limitación de 157 horas para todo el módulo, se decide hacer coincidir las Unidades de Trabajo (**UT**) con las ***Issues*** que se van a plantear a los alumnos. Además, se ha valorado la necesidad de ofrecer una formación mínima previa sobre la herramienta (GitHub) y la dinámica de trabajo que se espera provocar con MOS. Es por ello que se divide el currículo en 4 UT, tal y como refleja la *tabla 2*.

La razón por la que se asigna una UT diferenciada para BC4 y BC7 (**UT2** y **UT4**, respectivamente), hay que buscarla en su relevancia dentro de la materia impartida en este módulo. El correcto trabajo con colecciones (BC4) es uno de los pilares de la programación funcional, y la web tiene mucho de funcional, de la misma forma que lo tienen los lenguajes de programación *JavaScript* y sobre todo *Elm*. La asincronía en las comunicaciones web (BC7) también es de importancia capital para el futuro laboral de los alumnos.

Por otra parte, las **UT1** y **UT3** agrupan los BC correspondientes por afinidad, facilitando el diseño de *Issues* un poco más ambiciosas que si se planteasen UT más cortas.

UT - Issue	Bloque de Contenido
1	BC1 Selección de arquitecturas y herramientas de programación BC2 Manejo de la sintaxis del lenguaje BC3 Uso de los objetos predefinidos del lenguaje
2	BC4 Programación con *arrays*: funciones y objetos definidos por el usuario
3	BC5 Interacción con el usuario: eventos y formularios BC6 Uso del modelo de objetos del documento
4	BC7 Uso de mecanismos de comunicación asíncrona

Tabla 2: *División del currículo de MP0612 en 4 unidades de trabajo*. Fuente: Elaboración propia.

La mayor parte de la carga formativa sobre GitHub y MOS recaerá en la **UT1**, ya que, por un lado, las exigencias curriculares son menores y por otro, debe ser al comienzo cuando se exponga la mayor parte de esta información. Pese a ello, se debe tener en cuenta que a medida que se vaya desarrollando el módulo y aumente el dominio del proceso por parte de los alumnos, es muy probable que busquen personalizar su manera de trabajar. Por lo tanto, el docente deberá planear más sesiones en las UT siguientes que faciliten esa construcción incremental del conocimiento. En la introducción se hablaba de "*aprendizaje continuo, iterativo, estratégico, pero profundo*", como características del aprendizaje en el *open source* (o MOS); con esto se estaría fomentando lo iterativo, estratégico y continuo.

3.4.2 Paso 2 – Preparar proyecto contenedor e *Issues*

El segundo y último paso previo que debe realizar el docente antes de comenzar las clases es escoger un proyecto sobre el que poder plantear las *Issues* que requiere MOS para lanzar el proceso de aprendizaje.

Las características del proyecto diseñado deben ser bastante concretas. Por un lado, es necesario que sea realista y no muy artificial. Por otro, es indispensable que permita añadir *Issues* que se adecúen al currículo (RA y CE asociados a cada BC), respetando las 4 UT programadas en el **Paso Previo 1**.

Otro condicionante a tener en cuenta es que, en el aula donde va a ser implantada la innovación, hay 18 alumnos. 18 es un número demasiado alto para poder trabajar de forma cómoda con MOS. Puede suponer todo un reto controlar la aportación homogénea de los alumnos, así como ofrecer una evaluación justa y adecuada. Por lo tanto, va a ser necesario establecer grupos. Para ello, se

pueden considerar las propuestas al respecto tanto de SCRUM como de ABP. SCRUM sitúa el número ideal de miembros de un equipo entre 3 y 9 (Schwaber y Sutherland, 2017), mientras que ABP se mueve en una horquilla de 5 a 8 personas (Exley y Dennick, 2004). Teniendo en cuenta que dentro de los grupos se pretende crear parejas de alumnos, siguiendo una práctica del eXtreme Programming (Beck, 2000) conocida como *pair programming* (Cockburn y Williams, 2000), ayudaría que el número de componentes de cada grupo fuese par.

Finalmente se decide establecer 3 grupos de 6 alumnos. Por lo tanto, el docente va a tener que desarrollar un total de 12 *Issues*. Para que no quede ninguna duda al respecto, se representa toda esta información -*Issues* que debe desarrollar el docente y su temática-, en la *tabla 3*. La clave aquí es entender que por cada UT del **Paso Previo 1**, el docente va a diseñar 3 *Issues* similares pero lo suficientemente diferentes para evitar que los distintos grupos se puedan copiar entre sí.

UT	Issue
UT1	**Issue1A**: Conocimiento web básico (para **Grupo A**) **Issue1B**: Conocimiento web básico (para **Grupo B**) **Issue1C**: Conocimiento web básico (para **Grupo C**)
UT2	**Issue2A**: Trabajo con colecciones (para **Grupo A**) **Issue2B**: Trabajo con colecciones (para **Grupo B**) **Issue3C**: Trabajo con colecciones (para **Grupo C**)
UT3	**Issue3A**: Interacción con el *Document Object Model* o DOM (para **Grupo A**) **Issue3B**: Interacción con el *Document Object Model* o DOM (para **Grupo B**) **Issue3C**: Interacción con el *Document Object Model* o DOM (para **Grupo C**)
UT4	**Issue4A**: Comunicación asíncrona (para **Grupo A**) **Issue4B**: Comunicación asíncrona (para **Grupo B**) **Issue4C**: Comunicación asíncrona (para **Grupo C**)

Tabla 3: *Relación de Issues agrupadas por unidades de trabajo.* Fuente: Elaboración propia.

Ésta no es una tarea trivial. Probablemente la mayor parte del trabajo extra que deba realizar el docente con MOS provenga del diseño y validación de estas 12 *Issues*. La implantación satisfactoria de la innovación depende, en gran medida, de este factor.

Para la definición del proyecto y desarrollo de *Issues* asociadas, se puede tomar como referencia e inspiración el sitio Frontend Masters (2018) que ofrece talleres a profesionales de la programación web. En ella se pueden encontrar algunos ejemplos en los que poder basar una decisión, como los *workshops* ofrecidos por Brian Holt (Frontend Masters, 2017a), Richard Feldman (Frontend

Masters, 2017b) y Henrik Joreteg (Frontend Masters, 2015). El liderazgo y solvencia de los tres está contrastado -Holt (s.f.) en *React*, Feldman (s.f.) en *Elm*, Joreteg (s.f.) en *Progressive Web Apps*-, como lo demuestra el haber impartido estos seminarios en las conferencias más importantes del mundo. Además, el código de los tres talleres (GitHub Inc., 2018g; GitHub Inc., 2018i; GitHub Inc., 2018d) tiene una licencia abierta, permitiendo la consulta y su utilización directa.

Tras considerar todos estos factores, se decide que el mejor proyecto para esta innovación es una interfaz web sobre los servicios proporcionados por la API o *Application Programming Interface* de GitHub (GitHub Inc., 2018r), ya que, de esta forma, se incide en los siguientes aspectos:

- **Realismo y relevancia social**: Se accede a una API real que puede ser usada por cualquier empresa, programador o estudiante.
- **Uso de colecciones** en **Issue2**: Como resultado de las consultas a la API de GitHub se obtiene una variedad de colecciones muy amplia, con conceptos como: Usuarios, *Stars*, *Forks*, *Issues*, *Pull Requests*, etc.
- **Asincronía** en **Issue4**: La API de GitHub puede ser consumida de modo asíncrono.
- **Transformación del DOM** en **Issue3**: Cada vez que se recibe información de la API al realizar una consulta, se debe actualizar los datos que se muestran por pantalla. Y en la web, esa actualización se hace mediante transformaciones del **DOM**.
- **Uso básico de la Web** en **Issue1**: Casi cualquier proyecto que se hubiese escogido ofrecería esta oportunidad.
- **Tres itinerarios diferentes**: Gracias a la gran cantidad de datos que maneja la API de GitHub es relativamente sencillo diseñar 3 itinerarios equivalentes, pero lo suficientemente diferentes, para cada grupo. A modo de ejemplo:
  - El **Grupo A** centraría su actividad en *Usuarios* y número de Repositorios publicados.
  - El **Grupo B** podría trabajar con *Forks* y *Commits* de los Repositorios.
  - El **Grupo C** debería centrarse en la parte más social de la API, trabajando con *Stars* y *Follows* de cada Usuario.
- **Meta-aprendizaje**: La herramienta que permite plantear esta innovación es la misma que que proporciona la API para llevar a cabo las *Issues*: **GitHub**. Esto debería contribuir a la profundización del conocimiento asociado a esta plataforma.

Para completar este **Paso Previo 2**, es necesario que el docente haya resulto de antemano las 12 *Issues*, comprobando que:

- El nivel de dificultad es adecuado para ser resuelto por grupos de 6 estudiantes.
- Cumple con los RA y CE de cada UT.

- Es posible resolverlas en el tiempo asignado.
- Los itinerarios de cada grupo son equivalentes.

Llegados a este punto, el docente ya cuenta con toda la información necesaria para completar el diseño y redacción de las 4 UT descritas en la *tabla 2*.

Finalmente, y a modo de corolario, si se compara lo expuesto en este **Paso Previo 2** con cómo se desarrollan proyectos reales de *open source*, se puede considerar que el docente, al diseñar el proyecto, está asumiendo el rol del *owner* esbozado en el **apartado 2.3**, pero con algunas particularidades propias del aula:

- Donde el *owner* ve todo en función del proyecto, el docente usa el proyecto como excusa para poder plantear *Issues* y provocar el aprendizaje en sus alumnos.
- El *owner* tiene un único objetivo: generar un producto cuanto antes. Sin embargo, el docente debe asumir unos condicionantes de los que el *owner* carece: los RA y los CE. Es por ello que se hace necesario que adapte la experiencia del *open source* a la realidad del aula, de forma que se maximice la transferencia enseñanza-aprendizaje.

3.5 Actividades

3.5.1 Actividad 1 – Preparar el repositorio inicial y hacer grupos

Para empezar a trabajar el docente debe:

1. Crear una organización en GitHub (GitHub Inc., 2018a) que represente al módulo MP0612.
2. Crear el proyecto en su ordenador.
3. Añadir el código del que partirán los alumnos para resolver las ***Issues 1A, 1B o 1C***.
4. Subir el proyecto a GitHub.

El centro en el que se implanta la innovación ya es una organización en GitHub y han decidido que no es necesario tener otra extra que represente al módulo **MP0612**. Por lo tanto, el paso 1 estaría resuelto de antemano. Los otros 3 pasos son triviales para un docente que imparta esta asignatura -no tiene sentido detenerse más en ello-. Con esto el proyecto estaría subido y compartido.

Ahora es necesario añadir a los alumnos (consultar aclaración técnica al respecto en el **Anexo C**), para lo que se requerirá su participación activa mediante la apertura de una *Issue* en el proyecto antes mencionado. En ella se presentarán y especificarán a que grupo pertenecen. Para poder llevar a cabo esta parte, se habrá dividido previamente a los alumnos en tres grupos mediante sorteo.

Aunque en este TFM se denomine a los tres grupos como **Grupo A**, **Grupo B** y **Grupo C**, en realidad, una vez formados se les asignará un **color**. De esta forma, gracias a la funcionalidad de etiquetado que ofrece GitHub, se facilitará la visualización de qué grupo está asignado a qué *Issue*.

En esta **Actividad 1**, por tanto, se pone en marcha MOS en el aula, de forma que:

- El docente sigue en su rol de *owner* -usando la nomenclatura de los proyectos *open source*- siendo aplicables exactamente las mismas consideraciones expuestas al respecto al final del **Paso Previo 2**.
- Los alumnos asumen el rol de colaborador de un proyecto *open source*, ya que abren una *Issue* mostrando su interés y anuncian que pretenden contribuir, por lo que piden derechos para añadir código siempre que lo deseen.

3.5.2 Actividad 2 – El REPL y las clases expositivas

En esta **Actividad 2**, se detallará cómo se puede transformar la porción más expositiva del currículo en algo completamente interactivo y que además fomente la filosofía de *The Hacker Way* con sus mantras "*be bold*" y "*code wins arguments*". Siendo completamente estrictos, esta **actividad** no formaría parte de MOS como tal (no se usa GitHub, no se aprende en red), pero sin embargo es un complemento ideal para probar código, tanto por parte del docente como por parte del alumno. Si se considera que los desarrolladores de *open source* necesitan probar código y aprender bajo demanda, el REPL (*Read-Eval-Print-Loop*) es un excelente compañero de viaje.

Antes de lanzar cualquiera de las 4 *Issues* que conforman el módulo, se debería impartir una o dos clases (dependerá de la extensión de la UT y los conocimientos teóricos asociados al lenguaje de programación usado) donde se exponga a los alumnos a la sintaxis que van a necesitar para resolver la *Issue* correspondiente. Esto sería el **equivalente a las clases maestras** donde se explica la teoría mínima necesaria.

Desde el **punto de vista del docente**, exponer esta información a través del REPL, permite mantener un cierto control sobre el currículo y minimizar el riesgo de lagunas en el mismo. Además, carece de mucho sentido que los alumnos se sumerjan directamente en la *Issue* sin ni siquiera tener nociones de qué van a necesitar para comprender lo que se pide. Comparando con lo que suele ocurrir en el desarrollo de *open source*, se debe valorar la figura del **mentor** que guía y ayuda a la hora de hacer las primeras aportaciones al proyecto, hasta que se coge cierta soltura. En el aula esta labor debe recaer inicialmente en el docente; si bien se espera que a medida que avance el módulo surjan alumnos que puedan asumir ese rol en tareas determinadas. Al fin y al cabo, en eso consiste el **aprendizaje en red**.

Desde el **punto de vista del alumno**, con el REPL se espera conseguir una mayor motivación e implicación en las siempre tediosas clases expositivas. Además, al tener que seguir los pasos del docente y reproducirlos en sus ordenadores, las clases pasan de ser potencialmente pasivas para el alumno a **activas**. Por ejemplo, un alumno, al replicar lo hecho por el docente, podría equivocarse y preguntar. Otro alumno podría experimentar más allá de lo que el docente ejecuta en su REPL.

El docente, para preparar estas clases, debe comprimir la mayor parte de los conceptos básicos y sintaxis del lenguaje relativos a la UT correspondiente. Más tarde, en el aula, expondrá esos conceptos partiendo del código.

Tomando en consideración la **UT2**, centrado en las colecciones, se podría mostrar a través del REPL las funciones que permiten manejarlas de forma abstracta y funcional. Por ejemplo:

- Aplicar una función a todos los elementos de una colección (función ***map***).
- Filtrar la colección de acuerdo a un criterio (función ***filter***).
- Combinar todos los elementos de la colección buscando un resultado final (función ***fold***).

Sin embargo, no se trata de usar ejemplo artificiales o técnicos a la hora de explicar estas funciones en el REPL. Al menos no durante toda la sesión, ya que un enfoque realista o situado ayudaría a entender el sentido real de las mismas.

Para ilustrarlo se podría considerar, por ejemplo, una colección con el número de *favs* de los 10 *tweets* más famosos de la cantante Selena Gómez, entonces:

- Se quiere sumar 10 *favs* más a cada uno de sus *tweets*: se explica un ejemplo de cómo ***map*** permite sumar 10 a todos los elementos.
- Se quiere filtrar (*deshacerse de*) todos los *tweets* con menos de 1.000.000 de *favs*: se explica un ejemplo con ***filter*** que permite usar un predicado para deshacerse de todos los elementos que no cumplan la condición, creando una nueva colección de *tweets*.
- Se quiere obtener el número total de *favs* de esos 10 *tweets*: se explica que con ***fold*** se pueden sumar todos los elementos que conforman una colección.

Usando, nuevamente, los contenidos de la **UT2**, se pueden ver las diferencias de estos dos enfoques en la *tabla 4*. Tal y como ocurría con el ejemplo de los *tweets* de Selena Gómez, se debe favorecer siempre que sea posible la columna de la izquierda, incluso en las sesiones de REPL.

Enfoque realista o situado	Enfoque artificial o técnico
¿Qué es una colección de usuarios?	¿Qué es un *array*?
¿Cómo saco un listado de todos los usuarios?	¿Cómo recorro un *array*?
¿Cómo accedo a un usuario de la colección?	¿Cómo leo un valor de un *array*?
¿Cómo modifico un usuario de la colección?	¿Cómo modifico un valor de un *array*?
¿Cómo añado un usuario a la colección?	¿Cómo añado un elemento a un *array*?
¿Cómo borro un usuario de la colección?	¿Cómo elimino un elemento de un *array*?

Tabla 4: *Comparativa de enfoque realista frente a enfoque técnico para arrays.* Fuente: Elaboración propia.

Finalmente, hay que recordar que uno de los objetivos de esta propuesta es que los alumnos se acostumbren a adquirir de forma iterativa los conocimientos necesarios (y no más). Para ello deben profundizar únicamente en aquellos conceptos que representen ideas importantes. En otras palabras, se necesita fomentar el **aprendizaje profundo** a través de un **enfoque estratégico** y esta **actividad** en el REPL es una candidata perfecta para ilustrarlo.

3.5.3 Actividad 3 – Conversación informal diaria

Esta **actividad** se inspira en las reuniones *Daily Scrum* de SCRUM (Schwaber y Sutherland, 2017). Se realizará a diario y consistirá en preguntar a cada grupo:

1. ¿Qué habéis hecho en la clase anterior?
2. ¿Qué pensáis hacer en la clase de hoy?
3. ¿Qué problemas os habéis encontrado?

El docente dedicará menos de 5 minutos por grupo a esta actividad, justo al comienzo de la clase. Todos los grupos deben acudir a las conversaciones de los demás, así, si se plantease algún impedimento, cualquiera estaría en disposición de ayudar a solucionarlo. Además, cada grupo deberá rotar a diario al interlocutor, para que ninguno de los alumnos tenga la tentación de esconderse.

Con esta actividad:

- Se fomenta la competencia social.

- Se practica la síntesis en la expresión oral: al exponer de forma sucinta los resultados del trabajo realizado.
- Se normaliza la idea de exponer los problemas en público: una forma de inmunizar a los alumnos contra el *Impostor Syndrome*.
- Se desarrolla la competencia del sentido de la iniciativa: al forzar variaciones en la organización interna del grupo.

También se trabaja el **aprendizaje en red**, ya que los alumnos van descubriendo:

- Quién sabe qué.
- A quién pueden dirigirse, dentro de su red, en función del problema.
- Cuáles son sus verdaderas habilidades, o desde el punto de vista opuesto, cuándo les van a preguntar algo en busca de una solución.

Estos tres puntos suponen la manifestación de las **condiciones 2 y 3** de **Illich** (1970), expuestas en el apartado 2.4:

- **Condición 2**: Habilitar la comunicación entre quienes quieran compartir su conocimiento y quiénes quieran aprenderlo.
- **Condición 3**: Promover un espacio en el que cualquier persona que desee explicar un problema pueda ser escuchada.

Debe aclararse que el **objetivo** de esta actividad **no es fiscalizar** el trabajo del equipo. La idea es que las respuestas obtenidas sirvan al docente para saber cuándo debe asumir una actitud pasiva o de espejo y cuando una de ayuda proactiva.

Cuando las respuestas constaten que los problemas encontrados no son muy graves, el docente monitorizará cómo evolucionan los eventos a la espera de que el grupo encuentre una solución. Cuando las respuestas planteen un problema serio que ponga en riesgo la finalización de la *Issue* en curso por parte del grupo, el docente tomará las medidas que considere oportunas para solucionar la situación.

Ejemplos de estas medidas pueden ser:

- Aclarar conceptos en el REPL para toda la clase.
- Escribir una aclaración en la *Issue* o *Pull Request*.
- Contribuir directamente a la *Issue* en curso con código en un *Pull Request*.
- Describir una solución en los apuntes del módulo.
- Plantear una clase especial dedicada al tema en cuestión.

Se trata de que el docente intervenga lo mínimo posible para que los alumnos -en red- encuentren una solución que les permita construir el conocimiento a su ritmo y de forma activa. De hecho, se podría argumentar que las respuestas obtenidas son más útiles al grupo en sí que al docente.

Con esta **actividad** el docente obtiene *feedback* diario sobre la implantación de MOS en el aula, asumiendo una labor que toma prestadas atribuciones propias de un *SCRUM Master* (Schwaber y Sutherland, 2017) con la capacidad para ejercer de mentor cuando se le requiera.

3.5.4 Actividad 4 – Apuntes colaborativos

Esta **actividad** plantea una forma diferente de relacionarse con la información y el conocimiento. El docente **no** va a proporcionar unos **apuntes cerrados**, ni completos, ni definitivos. Se subirán a GitHub -en la Wiki asociada al proyecto- los apuntes iniciales relativos a cada *Issue* o **UT**. Pero no estarán completos y se esperará que los alumnos **plasmen sus experiencias en ellos**, como parte de las actividades evaluables que deben realizar.

Para ello:

- El docente liberará la versión inicial de los apuntes cuando dé a conocer las *Issues*.
- Los alumnos podrán consultarlos cuando deseen, pero también deberán contribuir a mejorarlos y ampliarlos mediante *Pull Requests* (como si fuese código).

Cuando se compara este proceso con su equivalente en el *open source*, no se encuentran apenas diferencias:

- En el *open source* es tan necesario contribuir con código a un proyecto, como con documentación. En el aula también.
- En el *open source* una de las claves para que un proyecto amplíe su base de usuarios es la calidad y relevancia de su documentación. En el aula, el aporte a la Wiki de apuntes es una responsabilidad conjunta, que se premia como actividad evaluable. Por lo tanto, se está fomentando el mismo espíritu.

Al realizar esta **actividad** se busca:

- Transmitir la idea de que los repositorios de información son entes vivos a los que todos pueden contribuir y de los que todos pueden beneficiarse.
- Mostrar que cada uno aprende de una forma diferente y a un ritmo propio.
- Potenciar una visión estratégica del aprendizaje.

No disponer de toda la información necesaria para resolver las *Issues* tiene unas implicaciones **mucho más profundas** de lo que pueda parecer, ya que los alumnos deberán habituarse a trabajar con conocimientos incompletos. Esta misma situación, multiplicada por cien, va a formar parte de su día a día cuando se incorporen al mercado laboral. La clave aquí es que consigan entender que todos los profesionales web trabajan así, con información y conocimientos incompletos. La **diferencia entre el éxito y el fracaso** puede venir marcado, no por el conocimiento previo adquirido (conocimiento cuasi-memorístico), sino por la habilidad para detectar un problema, entender el problema y saber dónde/a quién acudir para resolverlo. Al final ésta es una de las manifestaciones más claras de la necesidad de desarrollar la competencia de **aprender a aprender** y el **aprendizaje en red**. Aquellos que interioricen la idea de que un buen profesional conoce de antemano toda la información necesaria para resolver un problema, caerá inexorablemente en el *Impostor Syndrome*, porque en su visión del mundo, existirán unos profesionales mitificados que todo lo saben. Sin embargo, los que entiendan la realidad tal y como es, y desarrollen la capacidad para aproximarse de forma estratégica -casi *just-in-time*- al aprendizaje, triunfarán.

3.5.5 Actividad 5 – Retrospectiva evaluable

En esta **actividad** todas las parejas deben realizar una breve exposición -de unos 5 minutos- por cada UT o *Issue*, basada en algo aprendido y sin utilizar las herramientas típicas de presentación como Powerpoint o Keynote.

La idea es que la retrospectiva verse sobre:

- Algo relacionado con el código, y por qué no, utilizando el REPL.
- El aprendizaje.
- La resolución de un problema.

La prohibición de usar Powerpoint o Keynote no es un capricho ya que, preparar la retrospectiva apenas debería consumir tiempo. Al centrarse en un aspecto concreto, incluso anecdótico, relacionado con el aprendizaje y el código, no hay una necesidad de *vender* los logros. Todo lo contrario, se espera que sea casual y que de forma natural se incorpore a su rutina habitual.

Al haber 9 parejas en el módulo donde se va a implantar la innovación, el docente debe reservar unos 90 minutos (10 minutos por pareja) en cada UT para esta actividad. Al contrario de lo que ocurre en las *Sprint Retrospectives* (Schwaber y Sutherland, 2017) de SCRUM -de donde toma inspiración esta **Actividad 5**-, no se habilitarán clases íntegras al final de la UT para llevarlas a cabo. Es mucho más interesante que, en aras de conseguir que la auto-revisión y divulgación de los

conocimientos se incorporen al vocabulario y prácticas de los alumnos, se dediquen los últimos minutos de una buena parte de las clases a las retrospectivas.

Lo ideal sería que los propios alumnos fuesen anunciando cuándo expondrían su retrospectiva y que éstas se repartiesen de forma equitativa por toda la UT. Si no ocurre así y se acumulan todas, por ejemplo, en las dos últimas clases de la UT1, el docente debería intervenir en la UT2 y sortear los días en los que cada pareja hace su intervención en público, con la esperanza de que en la UT3 volviesen a ser los alumnos los que cogiesen el timón de esta actividad y repartiesen las presentaciones por toda la UT.

Ejemplos de retrospectivas podrían ser:

- Para qué y para qué no usar *map* al trabajar con colecciones.
- Dos maneras distintas de actualizar el DOM y por qué solo deberías usar una de ellas.
- Qué he aprendido en lo que llevamos de *Issue2*.
- Cosas que no debemos hacer cuando colaboramos con otros grupos (y cosas que sí).
- Cuándo usar *rebase* al resolver un *Pull Request*.

Estas retrospectivas formarían parte de la rúbrica de evaluación de los alumnos. A la hora de ser calificadas, la coevaluación supondría una magnífica oportunidad para abrir la participación a toda la clase, ofreciendo *feedback* sobre lo expuesto. El docente necesitaría reservar algo más de 10 minutos por presentación, en ese caso, pero nunca más de 15 minutos, para evitar que discusiones estériles puedan descarrillar el espíritu de la retrospectiva.

3.5.6 Actividad 6 – Prueba evaluable sobre el REPL

Esta **actividad** sustituiría a los exámenes de papel y bolígrafo más clásicos. Se trata de poder verificar que los RA y CE son alcanzados y que no existen lagunas conceptuales derivadas de la implantación de MOS en el aula.

Estas pruebas se deberían realizar la última semana de cada UT, por lo que se programarían 4 en todo el módulo. Constarían de unas 10 preguntas a resolver en 50 minutos que, en su mayoría, deberían haber sido tratadas por el docente en su sesión inicial sobre el REPL descrita en la **Actividad 2**. La dificultad de las mismas, por lo tanto, sería baja.

Para controlar completamente el entorno (acceso a internet, información previa en el ordenador, etc.), estas pruebas se realizarían sobre los ordenadores del aula de prácticas. El alumno dispondría de acceso al REPL, donde iría probando las posibles soluciones que se le ocurriesen, hasta dar con

la solución final. Una vez hallada esta solución, la incorporaría a la hoja de la prueba junto con el resultado del REPL tras su ejecución.

Se podrían diseñar diferentes exámenes con variaciones mínimas, en caso de que se temiese que los alumnos pudiesen copiarse entre ellos. En este caso, si un alumno fuese capaz de copiar la solución de otro compañero -ya no se trataría de una copia literal-, sería porque en realidad ha adquirido ese conocimiento por el que se le está evaluando.

Dependiendo del lenguaje de programación utilizado en el módulo profesional MP0612, cabría la posibilidad de que la entrega del examen consistiese en los *logs* de la interacción del alumno con el REPL, de forma que se pudiese observar cuál ha sido el proceso por el que cada alumno ha llegado a la solución, analizando todos los intentos fallidos. Sin embargo, la recogida de estos *logs* podría complicar mucho una actividad que pretende ser sencilla de implementar y llevar a cabo.

El resultado de estas pruebas debería dar al docente una idea muy clara sobre si los objetivos de conocimiento más básicos están siendo alcanzados, y no debería ser extraño que la inmensa mayoría de los alumnos la superasen de forma sobresaliente. En caso de no ser así, el docente debería revisar:

- La dificultad de las preguntas de estas pruebas.
- La materia tratada en la **Actividad 2**.
- La implantación de MOS en sí.

3.5.7 Actividad 7 – MOS *Workflow*

En esta **actividad** se explica cuál será el *workflow* o flujo de trabajo típico dentro de una UT o *Issue*. Esta dinámica es muy parecida a la que acontece en cualquier contribución a un proyecto de *open source* socialmente relevante y consiste en:

1. El docente abre la ***Issue*** original en la que se explica qué se pretende conseguir.
2. Cada grupo discute sobre la ***Issue*** original respectiva:
    - La mejor forma de conseguir lo que se pide.
    - Cómo repartirse el trabajo de forma efectiva.
3. Cada pareja abre un ***Pull Request*** en el que:
    - Dejan constancia de la parte del problema que pretenden resolver.
    - Colgarán, paulatinamente, el código con su solución.
4. Los alumnos establecerán el diálogo que consideren necesario en el ***Pull Request***, en la ***Issue*** original o en *Issues* abiertas al respecto por ellos mismos.

5. Los alumnos contribuirán a la **Wiki** de apuntes a medida que vayan dando pasos hacia la consecución del objetivo.
6. Una pareja cree que ya tiene una solución parcial en forma de código a través de un ***Pull Request***.
7. El resto del grupo y el docente la revisan y ofrecen *feedback* orientado a la mejora de la solución. Si hace falta hacer correcciones, se pasa al **punto 8**. Si no hace falta, se pasa al **punto 9**.
8. La pareja toma el *feedback* en consideración y vuelve al **punto 4** con la idea de mejorar su solución.
9. El resto del grupo y el docente aprueban el ***Pull Request***, por lo que la pareja puede añadir el código al proyecto.
10. La pareja tomará nota del enlace del ***Pull Request*** y de las *Issues* relevantes en las que han participado. Si para resolver la ***Issue*** planteada por el docente deben aportar más código vuelven al **punto 3**. Si ya no van a aportar más código continúan al **punto 11**.
11. La pareja hace llegar al docente, a través de una página previamente creada en la Wiki, un informe con:
    - Los enlaces de los ***Pull Requests*** realizados en la ***Issue*** planteada.
    - Los enlaces a las *Issues* en las que han contribuido de forma relevante (ayudando a otros compañeros, planteando problemas que no se habían tenido en cuenta, etc.).
    - Los enlaces a ***Pull Requests*** o *Issues* de otros grupos en las que han contribuido de alguna forma.
12. La página de la **Wiki** presentada en el **punto 11** será evaluada por otra pareja de otro grupo -**coevaluación**- y revisada por el docente para cerciorarse de que se actúa acorde a una rúbrica previamente acordada.

Es importante resaltar tres aspectos:

- Los alumnos propondrán sus soluciones parciales/finales a la ***Issue*** en curso mediante ***Pull Requests***.
- Para que un ***Pull Request*** pueda ser añadido (*merge*) al proyecto, debe haber recibido el visto bueno de todos los miembros del grupo y del docente. Generalmente esto no suele acontecer a la primera, ya que siempre hay cosas que mejorar.
- Durante todo este proceso, el docente asumirá alternativamente una actitud pasiva -de espejo- o una de ayuda proactiva, tal y como se explica en la **Actividad 3**.

3.5.8 Actividad 8 – Búsqueda de información

A la hora de buscar información (algo que los alumnos estarán haciendo constantemente), se propondrán las siguientes fuentes:

1. Wiki de apuntes del módulo.
2. *Logs* o *screencasts* -en caso de ser grabados- de sesiones REPL.
3. Documentación oficial del lenguaje de programación escogido.
4. Preguntas a otro compañero o al docente.
5. Stack Overflow (Stack Exchange Inc., 2018).
6. Búsqueda en Google.

La idea es que utilicen las fuentes en este orden a modo de plantilla inicial o referencia, pero se espera que a medida que avance el módulo desarrollen sus propias estrategias y su propio método.

Por poner un ejemplo, al mejorar el aprendizaje en red, el punto 4 -**preguntar a otro compañero**- puede convertirse en la primera opción si se sabe que otro alumno -**nodo de su red**- puede explicar la situación en unos minutos. En otras ocasiones ocurrirá al revés y serán los compañeros los que recurran al conocimiento que uno ha ido adquiriendo. De todas formas, no se debe confundir esta interacción con pedir a un compañero el código que soluciona el problema (muy cercano a *copiar* la solución). Muy al contrario, se trata de acudir al compañero que pueda guiar en la dirección correcta, facilitando la creación de conocimiento y la profundización en el mismo. Esta labor de guía, inicialmente, va a ser ejercida, casi con exclusividad, por el docente.

3.6 Temporalizaciones

En esta propuesta de innovación no se desarrolla una UT concreta como parte de la intervención, por lo tanto, no tiene sentido realizar una temporalización exacta de la misma. Sin embargo, sí es posible imaginar:

- Cómo podría ser un día cualquiera en el aula (**Temporalización 1**).
- Cómo se puede planear una UT completa (**Temporalización 2**).

Contextualizar todas las actividades propuestas en el tiempo debería ayudar a entender en qué consiste realmente MOS.

3.6.1 Temporalización 1 – Dinámica diaria en el aula

La sesión 15 de la **UT1** podría discurrir según se detalla en la *tabla 5*.

Minutos	Actividades
0 a 10	Conversación informal diaria – **Actividad 3**
10 a 40	Trabajo sobre ***Issue1***: • Discusiones generales en *Issues* – **Actividad 7** • Discusiones sobre código en *Pull Requests* – **Actividad 7** • Escribir código en editor – **Actividad 7** • Apuntes y experiencias de aprendizaje en Wiki – **Actividad 4** • Búsqueda de información – **Actividad 8**
40 a 50	Retrospectiva evaluable de una pareja – **Actividad 5**

Tabla 5: *Ejemplo de posible temporalización de la sesión 15 de la UT1*. Fuente: Elaboración propia.

3.6.2 Temporalización 2 – Una posible *Issue2*

La **UT2**, a la que se ha asignado un total de 27 sesiones de las 157 del módulo, podría discurrir según se detalla en la *tabla 6*.

3.7 Recursos

MOS se beneficia enormemente de un entorno flexible donde los alumnos puedan agruparse a su gusto y dispongan de, al menos, un ordenador por pareja. También es necesario algún sistema de visualización que permita al docente compartir la pantalla de su portátil.

3.7.1 Recursos humanos

Para poder implantar MOS hace falta un **docente** con gran conocimiento de la materia, muchas ganas de trabajar y experiencia profesional con Git, GitHub y el *frontend* en general (*JavaScript* y/o *Elm*). Además debería ser capaz de adoptar diferentes roles según la situación y mostrar empatía hacia las dificultades que van a experimentar sus alumnos.

3.7.2 Recursos materiales

- **Aula** con un ordenador cada dos alumnos, como mínimo. Ideal si disponen de un portátil cada uno.

- Para controlar el entorno (conexión a Internet, información disponible en el ordenador, etc.) en la **Actividad 6** -prueba evaluable sobre el REPL-, sería recomendable disponer de un **aula de informática** con ordenadores previamente preparados por el docente (desconectados de internet).
- **Proyector** en aula o pantalla de TV plana equivalente.

Sesión	Actividades
1	Lanzar ***Issue2*** y apuntes colaborativos asociados para los 3 grupos de alumnos Manejo de colecciones en el REPL – **Actividad 2**
2	Manejo de colecciones en el REPL – **Actividad 2** Discusión en Issue2 – **Actividad 7**
3 a 13	Dinámica diaria en el aula – **Temporalización 1**
14	Dudas sobre colecciones en el REPL – **Actividad 2** Retrospectiva (últimos 10 minutos) – **Actividad 5**
15 a 18	Dinámica diaria en el aula – **Temporalización 1**
19	Clase práctica sobre *rebase* (reescribir la historia) en Git Dinámica diaria en el aula – **Temporalización 1**
20 a 25	Dinámica diaria en el aula – **Temporalización 1**
26	Prueba sobre el REPL – **Actividad 6**
27	Retrospectivas pendientes – **Actividad 5** Discusión sobre lo aprendido y diferentes soluciones encontradas a ***Issue2***

Tabla 6: *Ejemplo de posible temporalización de la Issue2*. Fuente: Elaboración propia.

3.7.3 Recursos software

- Una copia del editor *Atom* (GitHub Inc., 2018c) por cada portátil.
- Una cuenta en GitHub por cada alumno y otra más para el docente.
- Un navegador moderno como Firefox o Chrome por cada portátil.
- Una ventana de comandos -*bash* o incluso mejor *fish*- por cada portátil.
- Se puede optar por cualquier sistema operativo, pero OS X o Linux mejor que Windows.

- Si se opta por *JavaScript* como lenguaje de programación del módulo, una instalación de Node.js y su REPL por cada portátil.
- Si se opta por *Elm* como lenguaje de programación del módulo, una instalación de *Elm* y su REPL por cada portátil.

3.7.4 Recursos económicos

Todo el software y servicios usados en esta propuesta de innovación son *open source* y gratuitos, por lo tanto, no habría que destinar ningún recurso económico extra. Sin embargo, el centro para el que se propone la intervención -de carácter privado-, dota a cada alumno de un portátil en préstamo durante la duración del ciclo. Por lo tanto, se estaría incurriendo en un gasto económico extra.

3.8 Evaluación

La evaluación de los alumnos que propone este TFM quiere ser objetiva, consistente, justa, idónea y transparente. Además, las dimensiones y muestras evaluables deben ser amplias, diversas y adaptadas a potenciar el aprendizaje del alumno, tal y como corresponde a una evaluación moderna. Por todo ello, se pueden encontrar evidencias evaluables de todas las diferentes modalidades existentes, clasificadas según:

- **Carácter**: formal o informal.
- **Formato**: oral o escrita.
- **Destinatarios**: individual o grupal.
- **Agentes**: heteroevaluación, autoevaluación y coevaluación.
- **Finalidad**: inicial, continua, final.

A modo de ejemplo, la *tabla 7* lista algunas de las evidencias evaluables propuestas en las **actividades** del epígrafe 3.5 y sus características.

También se debe tomar en consideración el carácter evaluador de la **reunión informal diaria**, que no repercute directamente en la calificación final pero sí contribuye a la evaluación continua del aprendizaje, desde una perspectiva de:

- **Autoevaluación**: El primer destinatario de las respuestas del alumno es él mismo.
- **Coevaluación**: Otros alumnos pueden ofrecer *feedback* o ayuda.
- **Heteroevaluación**: Información real, diaria e informal de cómo va desarrollándose el proceso de aprendizaje en cada uno de los alumnos.

Evidencias	Clasificación
Retrospectivas	Informal, oral, grupal, coevaluada y continua
REPL	Formal, escrita, individual, heteroevaluada y continua
PRs y código	Formal, escrita, grupal, heteroevaluada (coevaluación parcial) y continua
Wiki de apuntes	Informal, escrita, grupal o individual, coevaluada y continua

Tabla 7: *Evidencias evaluables propuestas y su clasificación.* Fuente: Elaboración propia.

No existe una **evaluación inicial** formal entendida como una prueba de conocimientos previos, pero al comienzo de la **UT1**, con la primera sesión de REPL y las primeras reuniones informales diarias, el docente va a recoger esa información de manera indirecta. De esta forma tendrá margen a la hora de reelaborar grupos, cambiar parejas, programar sesiones extra de REPL, o cualquier tipo de actividad que ayude a mejorar y personalizar el aprendizaje para cada alumno. En casos extremos, una entrevista personal a los alumnos con mayores lagunas en el conocimiento previo supuesto puede ayudar a elaborar un plan más personalizado.

Las **evaluaciones parciales** se deberían plantear en función de cada UT o *Issue*. Siguiendo con el ejemplo que se proponía en la **Temporalización 2**, en el **Anexo H** (*tabla 9* y *tabla 10*) se puede consultar una **posible rúbrica de evaluación**.

En cuanto a la **evaluación final**, podría consistir en una suma simple de las experiencias e indicadores acumulados durante todo el módulo. En casos donde el resultado de la **evaluación final** fuese negativo -suspenso- o considerado injusto por parte del alumno, podría plantearse una actividad que mezclase autoevaluación, coevaluación y heteroevaluación -la última palabra la tendría el docente-. Para ello, el profesor debería haber sugerido a principios de curso que cada alumno o pareja mantuviese un **portafolio** con los informes de PRs, contribuciones a Wiki de apuntes, y cualquier otro aspecto relevante desde el punto de vista de su proceso de aprendizaje. De esta forma, en esta **actividad de revisión de nota final**, el alumno, reflexionaría en alto sobre lo aprendido y el porqué de no haber sumado lo suficiente para aprobar según los indicadores establecidos. Sus compañeros podrían expresar también su opinión en cuanto a la ayuda recibida.

El uso de rúbricas debería **reducir la subjetividad** en la evaluación, un aspecto positivo a todas luces, sin embargo, con esta **actividad de revisión de nota final** se invierte el criterio. Ahora sí es necesario valorar la subjetividad y el punto de vista del alumno en la evaluación. Esto debería, también, proporcionar material al docente para su autoevaluación y la autoevaluación de la propuesta de este TFM. De hecho, en caso de saldarse de forma positiva (el alumno aprueba tras la revisión final), debería llevar al docente a cuestionarse:

- Si la recogida de evidencias evaluables es la correcta.
- Si las dimensiones evaluables son realmente representativas del aprendizaje y trabajo ejercido por el alumno.
- Si el docente era consciente de la situación de este alumno y si no debería haberse planteado algún tipo de intervención previa.

Atendiendo a la transparencia es también importante señalar que, con la publicación de la *Issue* correspondiente, debería hacerse pública la **rúbrica asociada** – ver **ejemplo** en *tabla 9* y *tabla 10* de **Anexo H**- para que los alumnos tengan claro cuáles van a ser las expectativas del docente de cara a la evaluación, abriendo la puerta a que el estudiante pueda utilizarlo como parte de su proceso de aprendizaje.

3.9 Evaluación de la propuesta

A la hora de evaluar este TFM se manifiesta un problema obvio: la propuesta no ha sido implantada. Por lo tanto, se corre el riesgo de que todo lo descrito hasta el momento pudiese ser calificado de especulativo. Es por ello que se considera completamente contraproducente una matriz DAFO para este caso, ya que lo único que se conseguiría sería ahondar en el problema descrito. Desde esta óptica, la forma más útil -y honesta- de llevar a cabo la evaluación sería a través de la elaboración de un **cuestionario** con diferentes dimensiones e indicadores asociados para ser cubierto por el alumnado. De esta forma se obtendría ***feedback* real** -no especulativo- y de primera mano de los **verdaderos protagonistas** de la propuesta de este TFM: los **alumnos**.

Desde este prisma, resulta interesante recopilar información sobre cuatro dimensiones diferentes: MOS, actividades MOS, actitud y evaluación, amén de posibles mejoras, comentarios o sugerencias.

En el **Anexo G** se puede consultar la ***tabla 8*** con la relación de **indicadores** agrupados por **dimensiones** y un breve análisis del porqué de algunos de los indicadores escogidos.

4 Conclusión

De los objetivos marcados inicialmente en el epígrafe 1.2, es el **objetivo principal** -*aprender a aprender con GitHub*- el que más peso ha tenido a la hora de diseñar la propuesta y, por lo tanto, el proceso de creación e implantación de MOS en el aula. De hecho, es casi imposible concebir MOS sin GitHub. Además, al reproducir algunas de las prácticas que se pueden observar en los proyectos *open source* más populares, se proporciona la coartada perfecta para que los alumnos experimenten, en primera persona, las virtudes de GitHub como plataforma de aprendizaje más allá de su vertiente de gestor de versiones del código. En definitiva, una muestra de aprender a aprender.

Ahondando en este aspecto, se llega al **primero** de los **objetivos específicos** -*aprendizaje en red*-, que, al formar los diferentes grupos y potenciar el uso de *Issues* y *Pull Requests* como herramientas favoritas para la comunicación e interacción entre los alumnos, se satisface plenamente. También se potencia el autoconocimiento -puntos fuertes vs puntos débiles-, característica importante a la hora de establecer las alianzas en la red de cada uno ya que, por un lado, se deben compensar las lagunas de conocimiento con conexiones que ayuden a la competencia según Siemens, y por otro, es importante ser capaces de aportar y completar la competencia de los demás.

La forma en la que se plantean las *Issues* supone un giro claro hacia la mentalidad *problem solver* que se planteaba en el **objetivo específico número 2** -*actitud emprendedora*-. Incluso se adapta el currículo y las Unidades de Trabajo para que sea más evidente este aspecto, equiparando UT con *Issue* en muchos aspectos.

El **tercer objetivo específico** -*The Hacker Way*- es uno de los más difíciles de evaluar. Claramente todas las piezas han sido puestas para que la filosofía *The Hacker Way* se impregne en los alumnos. Al estar centrados en GitHub y reducir al mínimo (desde un punto de vista de la docencia) los elementos no directamente relacionados con el código, se produce una adhesión al código de conducta de *The Hacker Way*. Incluso se introduce la herramienta de experimentación definitiva sobre código -el REPL- con pruebas evaluables sobre el mismo.

Para compensar esa visión centrada en el código del tercer objetivo, se proponen una serie de prácticas directamente sacadas del *post-agile* que ayudan a mantener un control sobre parte de los procesos y dinámicas que se establecen con la implantación de MOS. Las **Actividades 3 y 5** hunden sus raíces en SCRUM. La **retrospectiva evaluable**, tal y como su nombre indica, será una de las muestras que se utilizarán para poner nota a los alumnos y las agrupaciones en parejas,

siguiendo la práctica del *pair programming*, es también *post-agile* en estado puro. Por lo tanto, parece justo concluir que se consigue el **objetivo específico número 4** -*post-agile*-.

Con el **objetivo específico número 5** -*documentación*- la situación es similar a la del objetivo específico número 3. Se establece la responsabilidad grupal en torno a la documentación -véase al respecto la **Actividad 5**- y se envían las señales correctas al promocionarla a la categoría de evaluable, pero no es sencillo saber si es suficiente para satisfacer el objetivo en su totalidad.

El **objetivo específico número 6** -*Impostor Syndrome*- se trabaja de la única forma posible: en el día a día. La imagen que proyecte el docente de sí mismo en el aula será determinante. ¿De qué sirve hacer hincapié en la importancia de no tener miedo a preguntar y ser capaz de manifestar, de forma natural, lo que se desconoce (importantísimo en una disciplina del saber de liquidez manifiesta) si el docente se presenta como un semidios griego que todo lo sabe? "Eso no lo sé pero creo que podemos buscarlo aquí", "esperad que se me ha olvidado cómo se hace eso", "es posible que no haya una única respuesta correcta" y un largo etcétera de frases similares deben salir de la boca del docente con total naturalidad. Si eso se cumple, se habrá caminado en la dirección correcta con respecto a este objetivo específico.

Finalmente, en este TFM se establecía una cuasi-hipótesis que servía de *leit motiv*: ***si se aprende haciendo, el desarrollo de open source es una manera de aprender***. Tras la propuesta, y a falta de datos reales provenientes de **implantar** y **medir**, se cree estar en la posición de concluir: ***si se aprende haciendo, MOS es una buena manera de aprender***.

5 Limitaciones y prospectiva

La **limitación** inicial más grande al plantear este TFM ha sido la escasez de escritos académicos que documenten propuestas similares. GitHub está empezando a popularizarse en entornos escolares, por lo que ya empiezan a aparecer reflexiones sobre su uso, pero a día de hoy, esas referencias, no son tan abundantes.

Otra **limitación** innegable viene motivada por los recursos. Para poder llevar a cabo algo parecido a lo descrito en este TFM, es deseable disponer de un aula equipada con ordenadores pero sin mobiliario fijo, que permita adecuar el espacio a las necesidades de cada momento. La mayor parte de aulas de informática de los centros suelen contar con ordenadores de sobremesa, lo que convierte la maleabilidad del aula en un oxímoron. Por lo tanto, la capacidad de ofrecer portátiles a los alumno podría suponer una limitación seria. Por otro lado, la capacidad de proyectar la pantalla del docente ha dejado de ser, a día de hoy, algo raro, habida cuenta la cantidad de escuelas que

empiezan a contar con un sistema de pizarra digital implementado, por ejemplo, con un AppleTV y un Televisor.

Una última **limitación** vendría dada por el hecho de que, en el momento de redacción de este TFM, todavía no se ha puesto en marcha la iniciativa presentada. Por ello cabría plantear el argumento de que todo lo detallado se encuentra en la esfera de la elucubración más o menos afortunada.

En cuanto a la **prospectiva**, se pueden destacar varios aspectos. Si bien la propuesta está perfectamente delimitada a un curso y materia concreta, nada impide intentar lo aquí expuesto en casi cualquier asignatura de un ciclo profesional que implique el desarrollo de software. En la contextualización se hace referencia a que varios docentes del departamento han mostrado su interés en ver los resultados para poder intentar aplicarlo a sus aulas. Esto abre una cantidad de posibilidades enormes que se materializarán en **tres propuestas de prospectiva**.

En **primer lugar**, sería interesante, en función de los resultados obtenidos en la primera iteración de implantación de MOS, plantearse realizar un verdadero proyecto *open source* con valor real para la comunidad del *frontend*, de forma que una vez finalizado el propio centro acabe asumiendo su mantenimiento de forma directa (con implicación de alguno de los docentes y otros voluntarios), indirecta (con implicación de ex-alumnos) o híbrida (la situación ideal).

Otra prospectiva muy interesante sería la de colaboración entre diferentes materias del ciclo. Pongamos por caso que al menos dos módulos están interesados en esta propuesta. Se podría realizar un proyecto conjunto que implicase a ambos docentes y sus respectivos alumnos. El proyecto ganaría en autenticidad y los alumnos asimilarían mucho mejor los conceptos y procesos propuestos. Sería esperable que de esta forma se multiplicasen los beneficios al mismo tiempo que disminuirían algunos de los riesgos.

Finalmente -siguiendo lo expuesto en las dos propuestas anteriores-, se podría soñar con un ciclo profesional que consistiese en un megaproyecto (con algunas partes heredadas de alumnos de años anteriores) en el que todas las asignaturas se impartiesen siguiendo MOS (con las consiguientes adaptaciones necesarias según la materia). De esta forma se podría convertir al centro en un referente mundial a nivel de producción *open source* y -en la opinión del autor de este TFM- por qué no, en un referente pedagógico de igual nivel. Es cierto que existen trabas que habría que salvar y la adaptación curricular podría resultar, en muchos casos, casi imposible, pero en un centro con las dosis adecuadas de ambición, voluntad y preparación, podría llevarse a cabo.

6 Bibliografía

Abramov, D. (2015, noviembre). First day at @facebook tomorrow! [Tweet]. Recuperado de https://twitter.com/dan_abramov/status/671135846830075904

Anderson, D. J. (2010). *Kanban: successful evolutionary change for your technology business*. Blue Hole Press.

Atlassian (Sin fecha). Comparing Workflows. Recuperado de https://www.atlassian.com/git/tutorials/comparing-workflows

Barrows, H. S. (1986). A taxonomy of problem–based learning methods. *Medical education*, *20*(6), 481-486.

Barrows, H. S. (1996). Problem–based learning in medicine and beyond: A brief overview. *New directions for teaching and learning, 1996*(68), 3-12.

Bauman, Z. (2000). *Modernidad Líquida*. FCE - Fondo de Cultura Económica.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, J., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001). Manifesto for agile software development. Recuperado de http://agilemanifesto.org/

Beer, B. (2018). *Introducing GitHub: A non-technical guide, 2nd Edition*. O'Reilly Media, Inc..

Bloomberg L. P. (2000, agosto). Napster's High And Low Notes. Recuperado de https://www.bloomberg.com/news/articles/2000-08-14/napsters-high-and-low-notes

Bort, J. (2014, marzo). The Stress Of Being A Computer Programmer Is Literally Driving Many Of Them Crazy. Recuperado de http://www.businessinsider.com/syndromes-drive-coders-crazy-2014-3

Bradbury, D., Haines, C., Pagel, P., Martin, M., Martin, R. C., Chelimsky, D., Rady, B., Holser, P., Lane, D., Vaughn, G., Nijhof, M., et al. (2009). Manifesto for Software Craftmanship. Recuperado de http://manifesto.softwarecraftsmanship.org/

Clance, P. R. e Imes, S. A. (1978). The imposter phenomenon in high achieving women: Dynamics and therapeutic intervention. *Psychotherapy: Theory, Research & Practice*, *15*(3), 241.

Cockburn, A. y Williams, L. (2000). The costs and benefits of pair programming. *Extreme programming examined*, *8*, 223-247.

Computer History Musseum (2018a). Apple II introduced. Recuperado de http://www.computerhistory.org/timeline/1977/#169ebbe2ad45559efbc6eb35720e7660

Computer History Musseum (2018b). IBM introduces its Personal Computer (PC). Recuperado de http://www.computerhistory.org/timeline/1981/#169ebbe2ad45559efbc6eb35720105c3

Chacon, S. y Straub, B. (2014). *Pro git*. Apress.

Decreto 109/2011, de 12 de mayo, por el que *se establece el currículo del ciclo formativo de grado superior correspondiente al título de técnico superior en desarrollo de aplicaciones web*. Diario Oficial de Galicia, 113, de 14 de junio de 2011.

Dees, I., Wynne, M. y Hellesøy, A. (2013). *Cucumber Recipes: Automate Anything with BDD Tools and Techniques*. Pragmatic Bookshelf.

Dewey, J. (1897). *My pedagogic creed* (No. 25). EL Kellogg & Company.

Díaz Barriga, F. (2003). Cognición situada y estrategias para el aprendizaje significativo. *Revista electrónica de investigación educativa*, *5*(2), 1-13. Recuperado de https://redie.uabc.mx/redie/article/view/85/1396

edX (2018). Introduction to Functional Programming. Recuperado de https://www.edx.org/course/introduction-functional-programming-delftx-fp101x-0

Exley, K. y Dennick, R. (2004). *Small group teaching: Tutorials, seminars and beyond*. Routledge.

Featherly, K. (2016). *ARPANET United States defense program*. Encyclopaedia Britannica [versión electrónica]. NewYork, EU: Encyclopaedia Britannica Inc. Recuperado de https://www.britannica.com/topic/ARPANET#ref321039

Feldman, R. [rtfeldman]. (Sin fecha). Tweets [Twitter]. Recuperado el 18 de marzo de 2018 de https://twitter.com/rtfeldman

Freeman, S. y Pryce, N. (2009). *Growing object-oriented software, guided by tests*. Pearson Education.

Frontend Masters (2015). Building Web Apps (with React, Ampersand, ES6 and Webpack) by Henrik Joreteg. Recuperado de https://frontendmasters.com/courses/modern-web-apps/

Frontend Masters (2016). Creating an Open Source JavaScript Library on Github Workshop by Kent C. Dodds. Recuperado de https://frontendmasters.com/courses/open-source/

Frontend Masters (2017a). Complete Intro to React, v3 (feat. Redux, Router & Flow) by Brian Holt. Recuperado de https://frontendmasters.com/courses/react/

Frontend Masters (2017b). Elm by Richard Feldman. Recuperado de https://frontendmasters.com/courses/elm/

Frontend Masters (2018). Overview. Recuperado de https://frontendmasters.com

git-scm.com (2018). git --fast-version-control. Recuperado de https://git-scm.com/

GitHub Inc. (2015a). Redux. How to dispatch many actions in one action creator, Issue #1. Recuperado el 20 de abril de 2018 de https://github.com/reduxjs/redux/issues/1

GitHub Inc. (2015b). Redux. Reduce context footprint, Pull Request #12. Files. Recuperado el 28 de abril de 2018 de https://github.com/reduxjs/redux/pull/12/files

GitHub Inc. (2015c). Redux. Spelling fix, Pull Request #2. Recuperado el 28 de abril de 2018 de https://github.com/reduxjs/redux/pull/2

GitHub Inc. (2016). redux-form. Help Needed, Issue #768. Recuperado el 15 de mayo de 2018 de https://github.com/erikras/redux-form/issues/768

GitHub Inc. (2018a). About organizations. Recuperado el 24 de abril de 2018 de https://help.github.com/articles/about-organizations/

GitHub Inc. (2018b). Airbnb @ GitHub. Recuperado el 7 de marzo de 2018 de https://github.com/airbnb

GitHub Inc. (2018c). Atom. Recuperado de https://atom.io/

GitHub Inc. (2018d). A Complete Intro to React by @btholt. Recuperado el 18 de marzo de 2018 de https://github.com/btholt/complete-intro-to-react

GitHub Inc. (2018e). Dan Abramov (@gaearon). Followers. Recuperado el 15 de mayo de 2018 de https://github.com/gaearon?tab=followers

GitHub Inc. (2018f). Dan Abramov (@gaearon). Repositories. Recuperado el 15 de mayo de 2018 de https://github.com/gaearon?tab=repositories

GitHub Inc. (2018g). elm-workshop by @rtfeldman. Recuperado el 18 de marzo de 2018 de https://github.com/rtfeldman/elm-workshop

GitHub Inc. (2018h). Facebook @ GitHub. Recuperado el 7 de marzo de 2018 de https://github.com/facebook

GitHub Inc. (2018i). Frontend Masters Workshop App by @HenrikJoreteg. Recuperado el 18 de marzo de 2018 de https://github.com/HenrikJoreteg/masters

GitHub Inc. (2018j). GitHub Classroom. Recuperado de https://classroom.github.com/

GitHub Inc. (2018k). GitHub Education. Recuperado de https://education.github.com/

GitHub Inc. (2018l). Microsoft @ GitHub. Recuperado el 7 de marzo de 2018 de https://github.com/microsoft

GitHub Inc. (2018m). React. Pulse. Recuperado el 14 de mayo de 2018 de https://github.com/facebook/react/pulse

GitHub Inc. (2018n). Redux. Recuperado el 14 de mayo de 2018 de https://github.com/reduxjs/redux

GitHub Inc. (2018ñ). Redux. Issues. Recuperado el 20 de abril de 2018 de https://github.com/reduxjs/redux/issues

GitHub Inc. (2018o). Redux. Pull Requests. Recuperado el 20 de abril de 2018 de https://github.com/reduxjs/redux/pulls

GitHub Inc. (2018p). Redux. RFC: Redux Starter Kit, Issue #2859. Recuperado el 20 de abril de 2018 de https://github.com/reduxjs/redux/issues/2859

GitHub Inc. (2018q). Redux. TypeScript: `combineReducers` returns reducer that accepts partial state, Issue #2809. Recuperado el 20 de abril de 2018 de https://github.com/reduxjs/redux/pull/2809

GitHub Inc. (2018r). REST API v3. Recuperado el 24 de abril de 2018 de https://developer.github.com/v3/

GitHub Inc. (2018s). Ruby on Rails. Recuperado el 6 de marzo de 2018 de https://github.com/rails/rails

GitHub Inc. (2018t). The State of the Octoverse 2017. Recuperado el 7 de marzo de 2018 de https://octoverse.github.com/

GitHub Inc. (2018u). Stories - See how people are using GitHub in education. Recuperado de https://education.github.com/stories

GitHub Inc. (2018v). Student Developer Pack - The best developer tools, free for students. Recuperado de https://education.github.com/pack

GitHub Inc. (2018w). Twitter, Inc. @ GitHub. Recuperado el 7 de marzo de 2018 de https://github.com/twitter

GOTO Conferences (2015a). *GOTO 2015 • Agile is Dead • Pragmatic Dave Thomas*. Presentado en GOTO Amsterdam 2015. Recuperado de https://www.youtube.com/watch?v=a-BOSpxYJ9M

GOTO Conferences (2015b). *GOTO 2015 • One Hacker Way • Erik Meijer*. Presentado en GOTO Copenhagen 2015. Recuperado de https://www.youtube.com/watch?v=FvMuPtuvP5w

Hanselman, S. (2011, agosto). I'm a phony. Are you?. Recuperado de http://www.hanselman.com/blog/ImAPhonyAreYou.aspx

Holland, B. (2017, febrero). JavaScript in 2017 – Libraries and Frameworks. Recuperado el 26 de febrero de 2018 de https://developer.telerik.com/topics/web-development/javascript-2017-libraries-frameworks/

Holt, B. [btholt]. (Sin fecha). Tweets [Twitter]. Recuperado el 18 de marzo de 2018 de https://twitter.com/ btholt

Illich, I. (1970, noviembre). Learning Webs. En *Deschooling society*. (pp. 72-104). Recuperado el 27 de febrero de 2018 de http://learning.media.mit.edu/courses/mas713/readings/DESCHOOLING.pdf

Joreteg, H. [henrikjoreteg]. (Sin fecha). Tweets [Twitter]. Recuperado el 18 de marzo de 2018 de https://twitter.com/henrikjoreteg

Kniberg, H. (2011). *Lean from the trenches: Managing large-scale projects with Kanban*. Pragmatic Bookshelf.

Ley Orgánica 2/2006, de 3 de mayo, de Educación. Boletín Oficial del Estado, 106, de 4 de mayo de 2006.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

Liu, A. (2013, junio). Overcoming Impostor Syndrome Or How I Learned to Stop Worrying and Love Coding. Recuperado de https://medium.com/@aliciatweet/overcoming-impostor-syndrome-bdae04e46ec5

López-Pellicer, F. J., Béjar, R., Latre, M. A., Nogueras-Iso, J. y Zarazaga-Soria, F. J. (2015, julio). GitHub como herramienta docente. En *Actas de las XXI Jornadas de la Enseñanza Universitaria de la Informática* (pp. 66-73). Universitat Oberta La Salle.

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

Neary, D. (2018, febrero). 6 pivotal moments in open source history. Recuperado de https://opensource.com/article/18/2/pivotal-moments-history-open-source

Orden ECD/65/2015, de 21 de enero, por la que *se describen las relaciones entre las competencias, los contenidos y los criterios de evaluación de la educación primaria, la educación secundaria obligatoria y el bachillerato*. Boletín Oficial del Estado, 25, de 29 de enero de 2015.

Rails Guides (2018). Contributing to Ruby on Rails. Recuperado el 6 de marzo de 2018 de http://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html

Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley.

Ruby, S. y Copeland, D. B. (2017). *Agile Web Development with Rails 5.1*. Pragmatic Bookshelf.

Ruby on Rails (2018a). Community. Recuperado de https://rubyonrails.org/community/

Ruby on Rails (2018b). Home. Recuperado de http://rubyonrails.org/

Reaktor (2014). *One Hacker Way - Erik Meijer*. Presentado en Reaktor Dev Day 2014. Recuperado de https://vimeo.com/110554082

Real Decreto 686/2010, de 20 de mayo, por el que *se establece el título de Técnico Superior en Desarrollo de Aplicaciones Web y se fijan sus enseñanzas mínimas*. Boletín Oficial del Estado, 143, de 12 de junio de 2010.

Schwaber, K. y Sutherland, J. (2017, noviembre). The SCRUM guide. *The definitive guide to scrum: The rules of the game.* Scrum Alliance.

Scrum Alliance (2015, julio). The 2015 State of Scrum Report - How the world is successfully applying the most popular Agile approach to projects. Recuperado de https://www.scrumalliance.org/ScrumRedesignDEVSite/media/scrumalliancemedia/files%20and%20pdfs/state%20of%20scrum/scrum-alliance-state-of-scrum-2015.pdf

Siemens, G. (2005). Connectivism: A learning theory for the digital age. *International journal of instructional technology and distance learning*, *2*(1), 3-10.

Sokhan, K. (2014). JavaScripting. Recuperado y actualizado el 26 de febrero de 2018 de https://www.javascripting.com/?sort=rating

Sommers, K. [kellabyte]. (s.f.). Tweets [Twitter]. Recuperado el 14 de mayo de 2018 de https://twitter.com/kellabyte

Stack Exchange Inc. (2018). Stack Overflow. Recuperado de https://stackoverflow.com/

Sutherland, J. y Sutherland, J. J. (2014). *Scrum: the art of doing twice the work in half the time*. Currency.

TechCrunch (2011, diciembre). The Nerdy Address Of Facebook's New Headquarters? 1 Hacker Way. Recuperado el 23 de abril de https://techcrunch.com/2011/12/05/1-hacker-way/

Thomas, D. (2014, marzo). Agile is Dead (Long Live Agility). Recuperado de https://pragdave.me/blog/2014/03/04/time-to-kill-agile.html

Thomas, D. y Heinemeier Hanson, D. (2005). *Agile Web Development with Rails*. Pragmatic Bookshelf.

Thoughtbot Inc. (2016). Git Protocol. Recuperado de https://github.com/thoughtbot/guides/tree/master/protocol/git

Toledo, M. (2016). Welcome Evan Czaplicki! [Blog post]. En el Blog de NoRedInk. Recuperado de http://blog.noredink.com/post/136615783598/welcome-evan

Vygotsky, L. S. (1979). Problemas de método. *El desarrollo de los procesos psicológicos superiores*. Grijalbo.

Wanstrath, C. (2008, abril). We Launched. Recuperado de https://blog.github.com/2008-04-10-we-launched/

Wilson, G. (2011, diciembre). Fork, Merge, and Share. Recuperado de https://software-carpentry.org/blog/2011/12/fork-merge-and-share.html

Wynne, M., Hellesøy, A., Tooke, S. (2017). *The Cucumber Book: Behaviour-driven Development for Testers and Developers, Second Edition*. Pragmatic Bookshelf.

Xunta de Galicia (Sin fecha). Distribución por curso dos módulos dos ciclos de formación profesional con oferta na Comunidade Autónoma de Galicia, duración anual en horas e distribución horaria semanal en períodos lectivos. Recuperado de http://www.edu.xunta.es/fp/webfm_send/7285

Zagalsky, A., Feliciano, J., Storey, M. A., Zhao, Y. y Wang, W. (2015, febrero). The emergence of GitHub as a collaborative platform for education. En *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (pp. 1906-1917). ACM.

Zuckerberg, M. (2012, febrero) Letter from Mark Zuckerberg. En *Shares Facebook Class A Common Stock* (pp. 67-70). Recuperado de https://www.sec.gov/Archives/edgar/data/1326801/000119312512034517/d287954ds1.htm#toc287954_10

7 Anexos

7.1 Anexo A: *Issues*, *Pull Requests* y *Diffs*

Issue: Es el mecanismo más usado para aportar información -sin ser código- al proyecto (aunque se puede incluir código). Una especie de "*micro foro de discusión*" (López-Pellicer et al., 2015, p. 70) que permite a los usuarios señalar *bugs* o sugerir nuevas funcionalidades (Beer, 2018). En la innovación propuesta en este TFM, el docente comunicará a los alumnos a través de una *Issue* los problemas que deben resolver, esperando que, a continuación, se produzca en ella una discusión sobre la mejor forma de abordarlo, cómo repartirse el trabajo, etc.

El verbo que se utiliza con *Issue* es *abrir*. Es decir, *voy a abrir una Issue porque tengo una sugerencia para mejorar el proyecto*. Una *Issue* no resuelta se dice que permanece *abierta*, mientras que una *Issue* resuelta ha sido *cerrada*. En las figuras que se muestran a continuación se pueden ver diferentes ejemplos relacionados con las *Issues* sacados del proyecto *open source Redux* (GitHub Inc., 2018n).

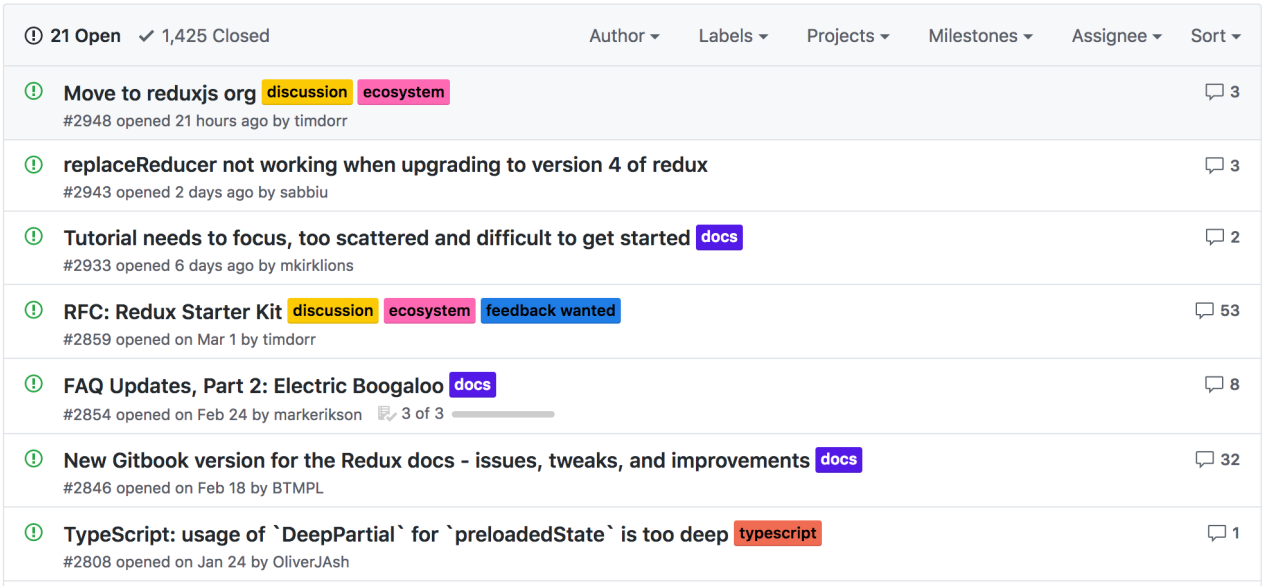
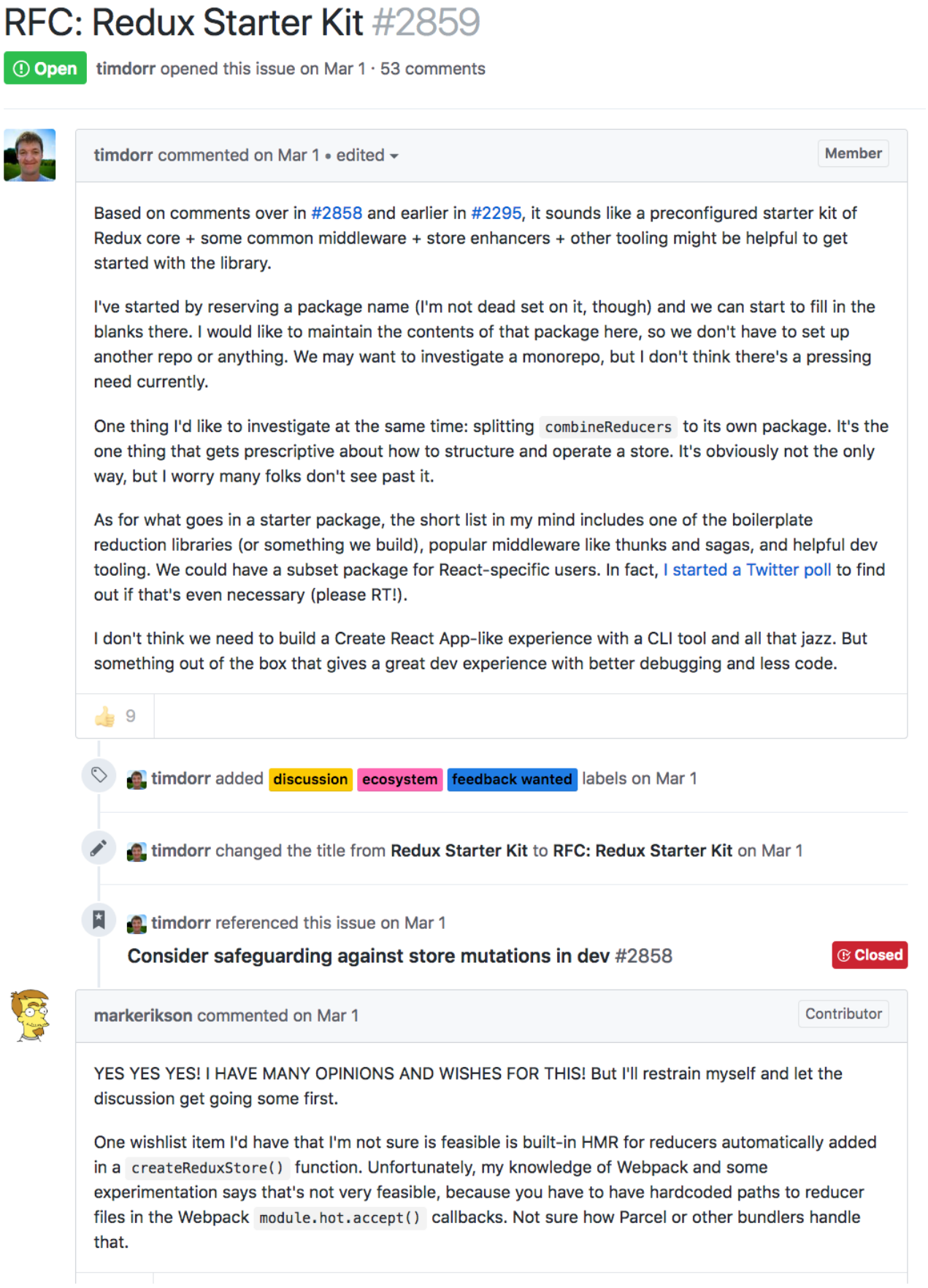


Figura 2: Detalle de la lista de Issues abiertas para el proyecto Redux. Fuente: GitHub Inc. (2018ñ).

A modo ilustrativo, destacar que, en el momento en el que se captura la *figura 2* (tal y como se puede apreciar en la esquina superior izquierda), *Redux* contaba con 21 *Issues* abiertas y 1.425 cerradas.



RFC: Redux Starter Kit #2859

Open timdorr opened this issue on Mar 1 · 53 comments

timdorr commented on Mar 1 • edited — Member

Based on comments over in #2858 and earlier in #2295, it sounds like a preconfigured starter kit of Redux core + some common middleware + store enhancers + other tooling might be helpful to get started with the library.

I've started by reserving a package name (I'm not dead set on it, though) and we can start to fill in the blanks there. I would like to maintain the contents of that package here, so we don't have to set up another repo or anything. We may want to investigate a monorepo, but I don't think there's a pressing need currently.

One thing I'd like to investigate at the same time: splitting `combineReducers` to its own package. It's the one thing that gets prescriptive about how to structure and operate a store. It's obviously not the only way, but I worry many folks don't see past it.

As for what goes in a starter package, the short list in my mind includes one of the boilerplate reduction libraries (or something we build), popular middleware like thunks and sagas, and helpful dev tooling. We could have a subset package for React-specific users. In fact, I started a Twitter poll to find out if that's even necessary (please RT!).

I don't think we need to build a Create React App-like experience with a CLI tool and all that jazz. But something out of the box that gives a great dev experience with better debugging and less code.

9

timdorr added discussion ecosystem feedback wanted labels on Mar 1

timdorr changed the title from **Redux Starter Kit** to **RFC: Redux Starter Kit** on Mar 1

timdorr referenced this issue on Mar 1

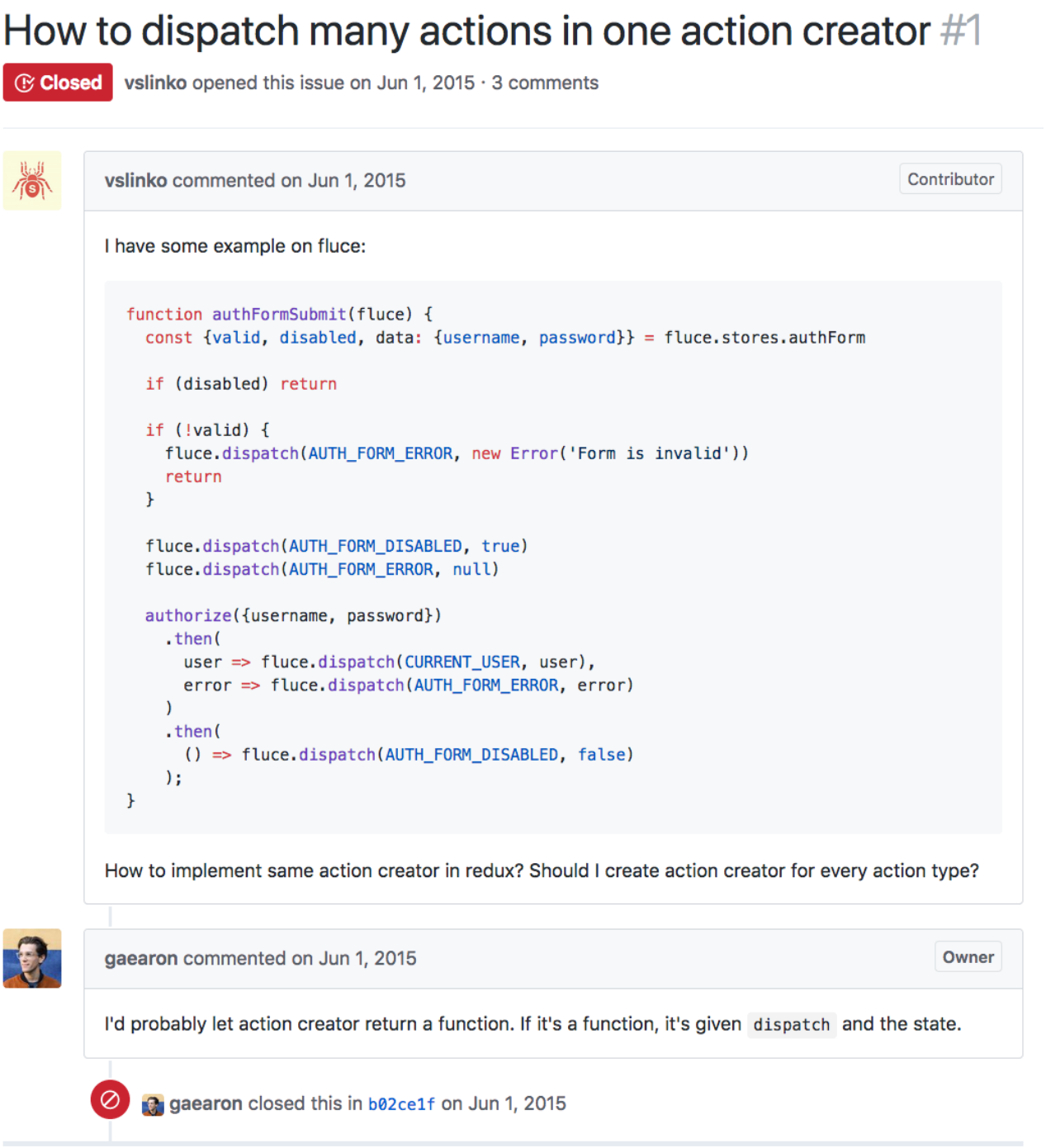
Consider safeguarding against store mutations in dev #2858 — Closed

markerikson commented on Mar 1 — Contributor

YES YES YES! I HAVE MANY OPINIONS AND WISHES FOR THIS! But I'll restrain myself and let the discussion get going some first.

One wishlist item I'd have that I'm not sure is feasible is built-in HMR for reducers automatically added in a `createReduxStore()` function. Unfortunately, my knowledge of Webpack and some experimentation says that's not very feasible, because you have to have hardcoded paths to reducer files in the Webpack `module.hot.accept()` callbacks. Not sure how Parcel or other bundlers handle that.

Figura 3: Detalle de una Issue abierta para el proyecto Redux. Fuente: GitHub Inc. (2018p).



How to dispatch many actions in one action creator #1

Closed **vslinko** opened this issue on Jun 1, 2015 · 3 comments

vslinko commented on Jun 1, 2015 — Contributor

I have some example on fluce:

```
function authFormSubmit(fluce) {
  const {valid, disabled, data: {username, password}} = fluce.stores.authForm

  if (disabled) return

  if (!valid) {
    fluce.dispatch(AUTH_FORM_ERROR, new Error('Form is invalid'))
    return
  }

  fluce.dispatch(AUTH_FORM_DISABLED, true)
  fluce.dispatch(AUTH_FORM_ERROR, null)

  authorize({username, password})
    .then(
      user => fluce.dispatch(CURRENT_USER, user),
      error => fluce.dispatch(AUTH_FORM_ERROR, error)
    )
    .then(
      () => fluce.dispatch(AUTH_FORM_DISABLED, false)
    );
}
```

How to implement same action creator in redux? Should I create action creator for every action type?

gaearon commented on Jun 1, 2015 — Owner

I'd probably let action creator return a function. If it's a function, it's given `dispatch` and the state.

gaearon closed this in b02ce1f on Jun 1, 2015

Figura 4: Detalle de la primera Issue cerrada en Redux por el propio Dan Abramov. Fuente: GitHub Inc. (2015a).

Pull Request: Es la forma que tiene GitHub de permitir que el código que se quiere incorporar a un proyecto -generalmente tras discusiones previas en una *Issue*- pueda ser revisado por otros colaboradores, el dueño (*owner*) del proyecto o cualquiera que decida echar una mano (Frontend Masters, 2016). Una vez revisado ese código se pueden sugerir mejoras hasta que cumpla una serie de estándares o, si es correcto en forma y fondo, se puede añadir al proyecto. Desde la perspectiva de este TFM, el mecanismo que se usará para que los alumnos aporten las soluciones a los problemas planteados será un *Pull Request*. Esas soluciones serán revisadas por sus compañeros y/o por el docente antes de ser dadas como correctas. Por lo tanto, tal y como se comentaba anteriormente, no hay diferencia alguna entre la mecánica para aportar código en un proyecto real *open source* y cómo van a aportar código los alumnos.

Al igual que las *Issues*, los *Pull Requests* pueden estar *abiertos* o *cerrados*. Cuando están *cerrados* y se han añadido al proyecto -es decir, no han sido descartados- se suele decir que han sido *añadidos* o *incorporados*. En las figuras que se muestran a continuación se pueden ver diferentes ejemplos relacionados con los *Pull Requests* sacados del proyecto *open source Redux* (GitHub Inc., 2018n).

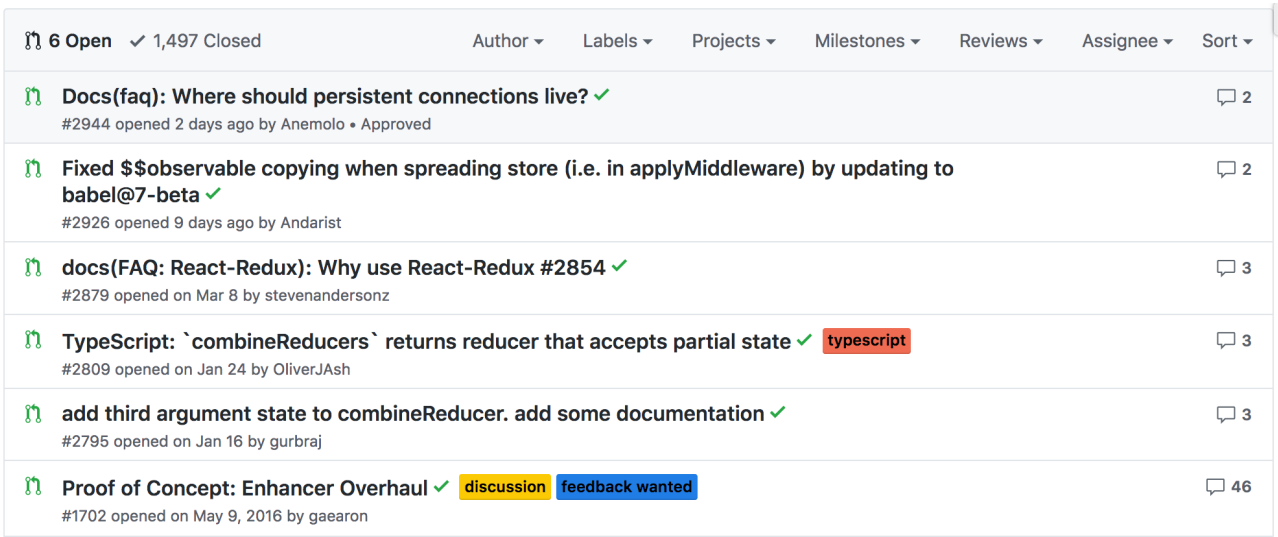


Figura 5: Detalle de la lista de Pull Requests abiertos para el proyecto Redux. Fuente: GitHub Inc. (2018o).

A modo ilustrativo, destacar que, en el momento en el que se captura la *figura 5* (tal y como se puede apreciar en la esquina superior izquierda), *Redux* contaba con 6 *Pull Requests* abiertos y 1.497 cerrados -unos pocos *descartados* y la mayoría *añadidos* o *incorporados*-.

Figura 6: Detalle de un Pull Request abierto para el proyecto Redux. Fuente: GitHub Inc. (2018q).

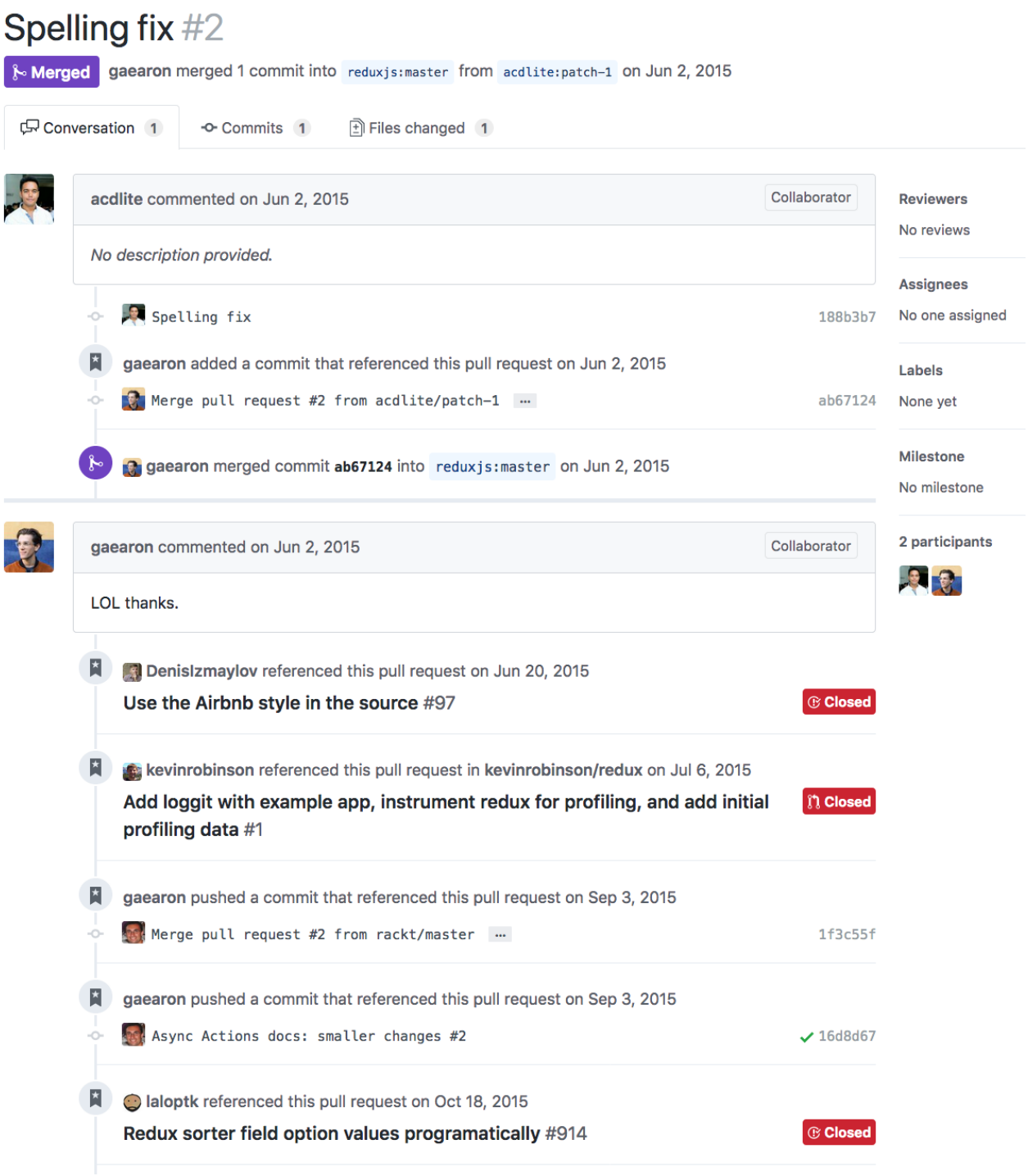


Figura 7: Detalle de uno de los primeros Pull Requests añadidos al proyecto Redux. Fuente: GitHub Inc. (2015c).

Diff: Este término, pese a no ser usado en este TFM, va a formar parte de la rutina diaria de los alumnos, por lo que se considera importante que aparezca en este anexo. "Diff" es un diminutivo de *difference* y se refiere a la diferencia entre las líneas de código añadidas y las borradas. Las añadidas aparecerán en color verde y las borradas en rojo. Desde una perspectiva un poco más técnica debe valorarse que cuando se habla de líneas añadidas y borradas se circunscribe a un *commit* -una "aportación" de código-.

La herramienta que proporciona GitHub para mostrar estas "*diferencias*" es muy gráfica -*figura 8*- y permite a los alumnos, de un solo vistazo, tener una indicación de cuanto código han añadido y cuanto han borrado. Por ello se decía que va a ser muy utilizada.

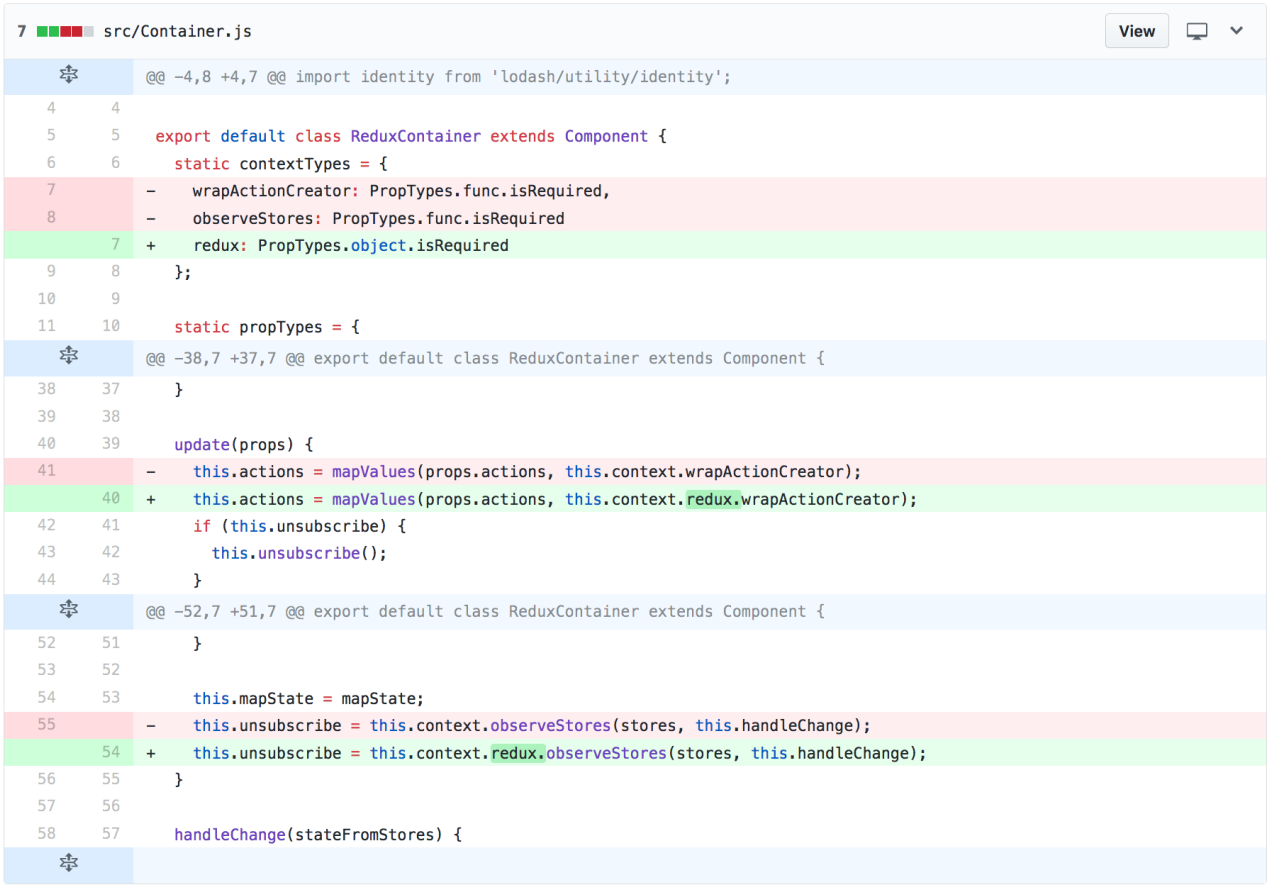


Figura 8: La herramienta Diff de GitHub en acción. Fuente: GitHub Inc. (2015b).

Recordar que en la *figura 8* se muestran las líneas añadidas en verde y las líneas borradas en rojo.

7.2 Anexo B: Las diferentes caras de GitHub

GitHub es una plataforma con múltiples caras. A continuación, se presentan ejemplos que ilustran las vertientes de colaboración (*figura 9*), red social (*figura 10*), portafolio (*figura 11*) o portal donde dar pistas sobre competencia en la acepción de Siemens (2005) (*figura 12*).

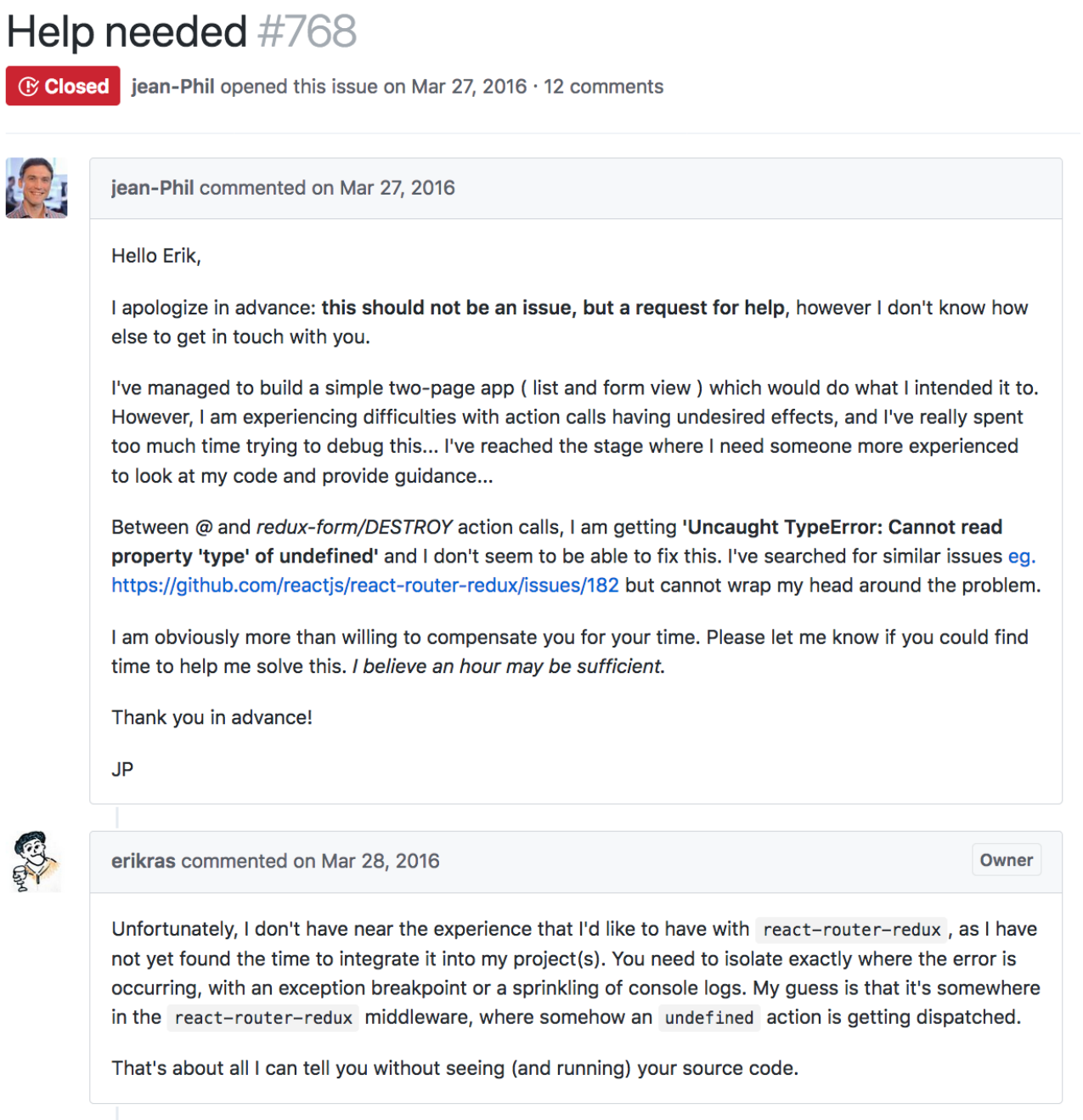


Figura 9: Ejemplo de GitHub como plataforma de colaboración. Fuente: GitHub Inc. (2016).

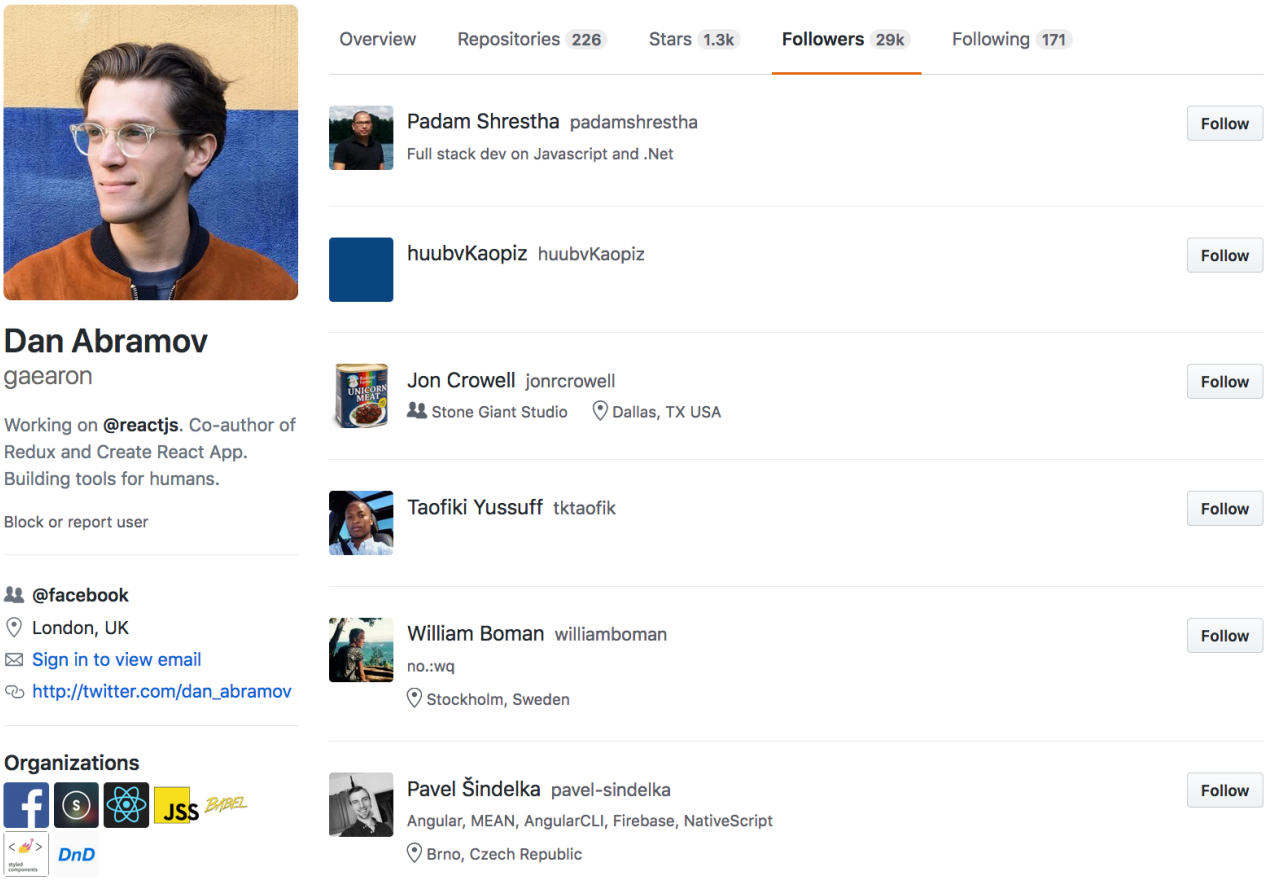


Figura 10: Ejemplo de GitHub como red social. Fuente: GitHub Inc. (2018e).

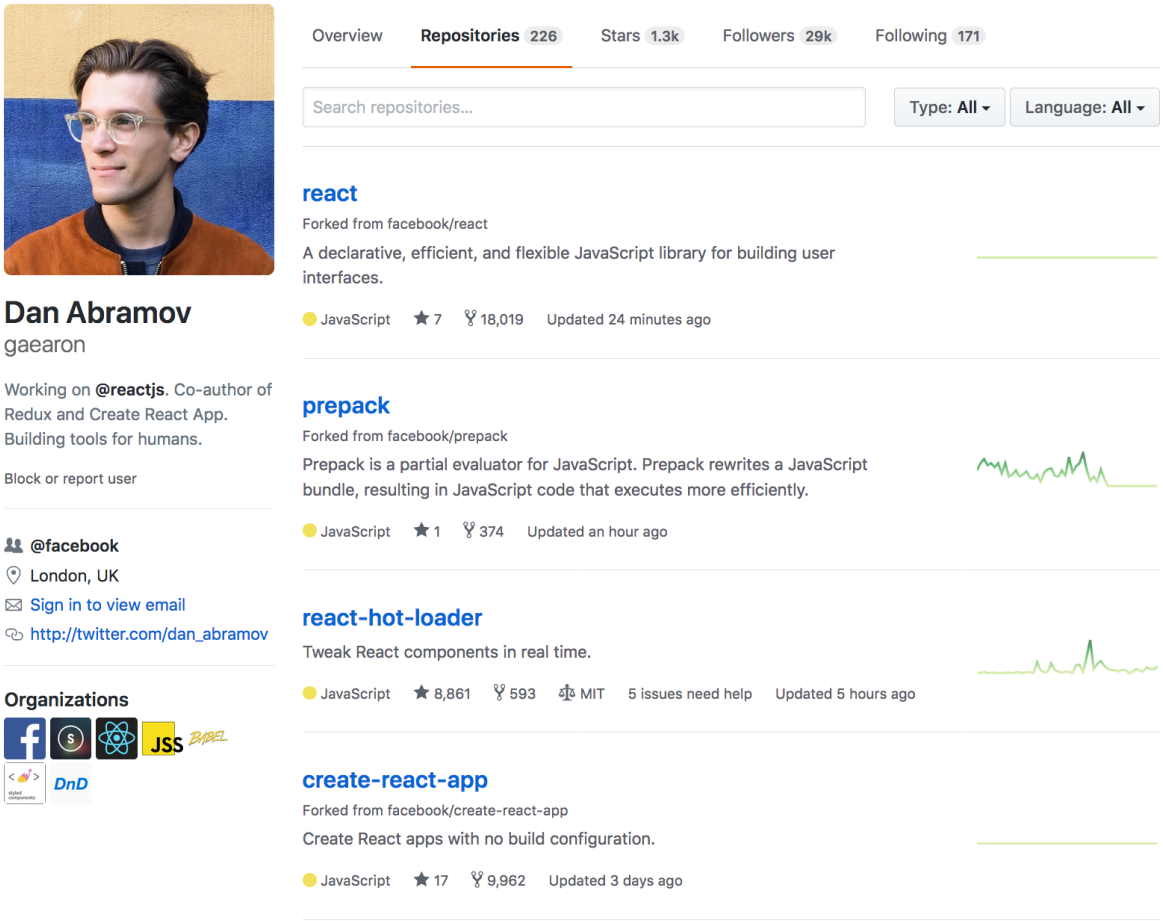


Figura 11: Ejemplo de GitHub como portafolio. Fuente: GitHub Inc. (2018f).

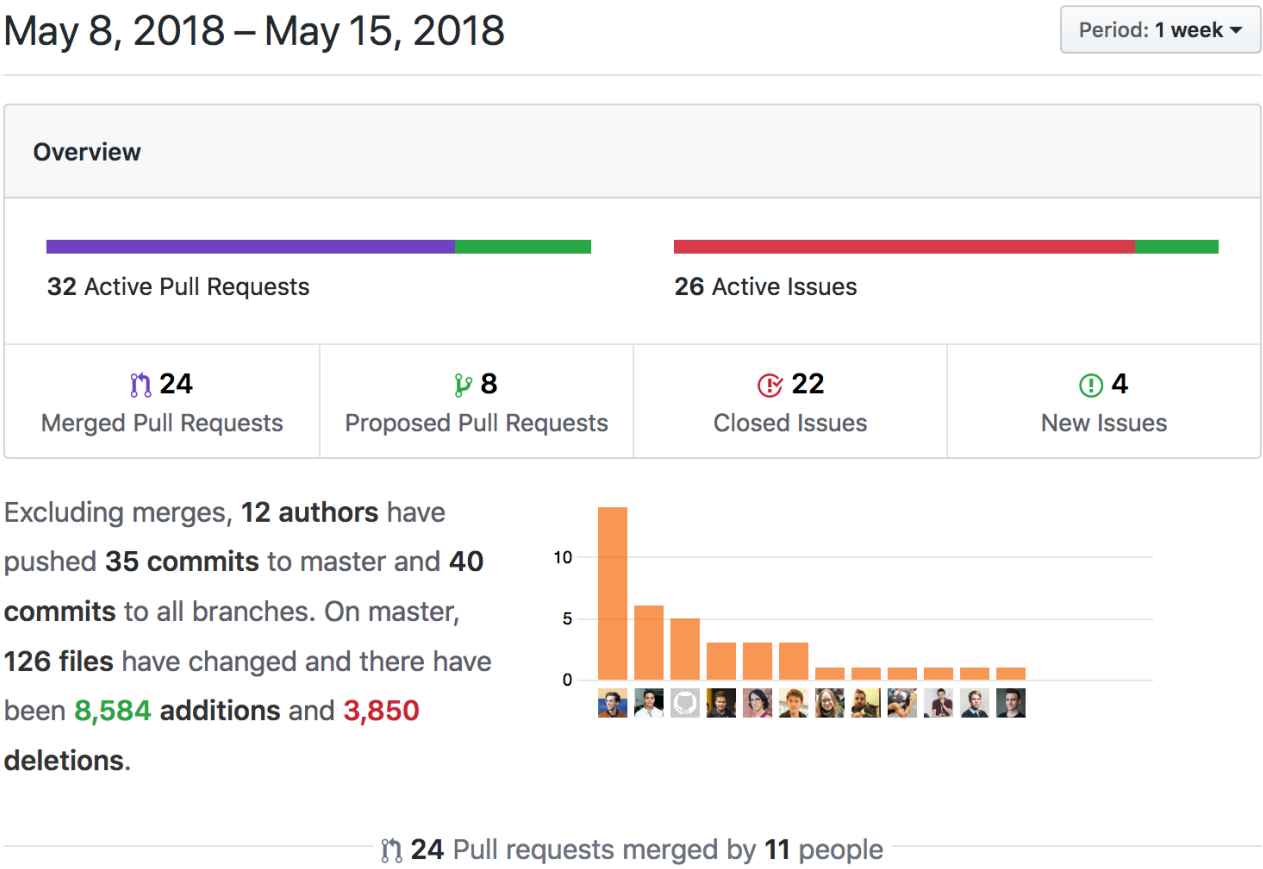


Figura 12: Ejemplo de GitHub como muestra de competencia. Fuente: GitHub Inc. (2018m).

7.3 Anexo C: Usuarios vs *Committers* en GitHub

En la **Actividad 1**, en aras de la claridad, se simplifica en exceso un concepto propio de GitHub, que hace necesaria una aclaración desde un punto de vista técnico. Siendo estrictos, el concepto de "***añadir* alumnos al *proyecto***" es muy difuso ya que, cualquier usuario de GitHub es libre de abrir una *Issue* para escribir lo que desee o un *Pull Request* con el código que le parezca. Sin embargo, sí hace falta alguien autorizado para *añadir* (*merge*) al proyecto el código asociado a los *Pull Requests*. Muchas veces se conoce este hecho como "**tener estatus de *committer***" o "tener derecho a *resolver* (*merge*) un *Pull Request*". Cuando en este TFM se habla de "*añadir* alumnos al *proyecto*", se hace referencia a otorgar ese derecho.

La razón por la que se hace esto, hay que buscarla en las dinámicas de los proyectos de *open source*. La idea que subyace en esta propuesta de innovación es, precisamente, replicar de la forma más fidedigna posible el contexto de desarrollo del *open source* y su metodología. En los proyectos reales, cuando alguien recibe el "derecho a *resolver* (*merge*) un *Pull Request*", implica que se le está otorgando una gran confianza, pero también una gran responsabilidad. Por lo tanto, **se considera muy importante** que los alumnos tengan ese "estatus de *committer*".

Todo esto tiene un peligro potencial ya que, cualquier alumno podría *corromper* el repositorio del proyecto. Pero teniendo en cuenta la naturaleza redundante de Git y GitHub, las posibilidades de pérdida real de información son casi inexistentes. Otro aspecto interesante, si se diese el caso antes citado, sería ver cómo reaccionan los alumnos. Podría esperarse que abriesen una *Issue* para pedir ayuda, de forma que otro alumno o el docente explicasen como restaurar el repositorio. Si así aconteciese, se consideraría un triunfo la actividad, ya que estarían replicando la mecánica habitual de los proyectos *open source* cuando alguien *corrompe* un repositorio y no sabe cómo *restaurarlo*.

7.4 Anexo D: Prácticas *agile*

Las siguientes prácticas de SCRUM (Schwaber y Sutherland, 2017) y eXtreme Programming (Beck, 2000) han servido de inspiración para el diseño de las **Actividades 3 y 5** y para justificar la agrupación por parejas de los alumnos:

- ***Daily Scrum***: Reunión diaria cronometrada, de como mucho 15 minutos, que sirve para planear el trabajo del día. Generalmente se celebra a primera hora de la mañana y deben acudir todos los desarrolladores del equipo. No hay sillas ni mesas, por lo que se suele llevar a cabo de pie. Es una buena forma de mantener a todo el equipo informado de los problemas y avances. También contribuye a no perder de vista el objetivo que se esté persiguiendo en ese ciclo de tiempo (*sprint*). Como contraste, decir que Erik Meijer (Reaktor, 2014; GOTO Conferences, 2015b) ha ridiculizado en diversas ocasiones estas reuniones.

- ***Sprint Retrospective***: Reunión que se celebra al finalizar el ciclo de tiempo (*sprint*) actual. Puede variar en duración, pero rara vez se extiende más de 3 horas. Acude todo el equipo SCRUM (desarrolladores, y algunos miembros más del equipo que, desde el punto de vista de este TFM, no resultan importantes). Debe servir para reflexionar sobre lo aprendido en el ciclo de tiempo (*sprint*) que acaba de finalizar.

- ***Pair Programming***: Práctica consistente en tener a dos personas por ordenador trabajando en equipo sobre un mismo problema. Supuso una pequeña revolución en su momento (aun a día de hoy no está exenta de polémica). La razón que justifica tal agrupamiento es que, en principio, la cantidad de trabajo realizada es muy similar pero la calidad es superior. Siendo tremendamente sarcásticos (habilidad sublime en manos de Erik Meijer), se podría establecer que la práctica aquí descrita, en otras profesiones, se llama "*hablar con el compañero*", pero que debido a la especial idiosincrasia de los programadores ha sido necesario renombrarla y establecer su obligado cumplimiento para todos aquellos que practiquen el eXtreme Programming.

7.5 Anexo E: Resultados de aprendizaje y criterios de evaluación

RA1. Selecciona las arquitecturas y las tecnologías de programación sobre clientes web, para lo que identifica y analiza las capacidades y las características de cada una.

- CE1.1. Se han caracterizado y se han diferenciado los modelos de ejecución de código en el servidor y en el cliente web.
- CE1.2. Se han identificado las capacidades y los mecanismos de ejecución de código de los navegadores web.
- CE1.3. Se han identificado y se han caracterizado los principales lenguajes relacionados con la programación de clientes web.
- CE1.4. Se han reconocido las particularidades de la programación de guiones, y sus ventajas y desventajas sobre la programación tradicional.
- CE1.5. Se han verificado los mecanismos de integración de los lenguajes de marcas con los lenguajes de programación de clientes web.
- CE1.6. Se han reconocido y se han evaluado las herramientas de programación sobre clientes web.

RA2. Escribe sentencias simples aplicando la sintaxis del lenguaje, y verifica su ejecución sobre navegadores web.

- CE2.1. Se ha seleccionado un lenguaje de programación de clientes web en función de sus posibilidades.
- CE2.2. Se han utilizado diversos tipos de variables y operadores disponibles en el lenguaje.
- CE2.3. Se han identificado los ámbitos de uso de las variables.
- CE2.4. Se han reconocido y se han comprobado las peculiaridades del lenguaje respecto a las conversiones entre tipos de datos.
- CE2.5. Se han utilizado mecanismos de decisión en la creación de bloques de sentencias.
- CE2.6. Se han utilizado bucles y se ha verificado su funcionamiento.
- CE2.7. Se le han añadido comentarios al código.
- CE2.8. Se han utilizado herramientas y entornos para facilitar la programación, la prueba y la depuración del código.

RA3. Escribe código, para lo que identifica y aplica las funcionalidades aportadas por los objetos predefinidos del lenguaje.

- CE3.1. Se han identificado los objetos predefinidos del lenguaje.
- CE3.2. Se han analizado los objetos referentes a las ventanas del navegador y los documentos web que contengan.
- CE3.3. Se han escrito sentencias que utilicen los objetos predefinidos del lenguaje para cambiar el aspecto del navegador y el documento que contenga.
- CE3.4. Se han generado textos y etiquetas como resultado de la ejecución de código en el navegador.
- CE3.5. Se han escrito sentencias que utilicen los objetos predefinidos del lenguaje para interactuar con el usuario.
- CE3.6. Se han utilizado las características propias del lenguaje en documentos compuestos por varias ventanas y marcos.
- CE3.7. Se han utilizado cookies para almacenar información y recuperar su contenido.
- CE3.8. Se ha depurado y se ha documentado el código.

RA4. Programa código para clientes web, para lo que analiza y utiliza estructuras definidas por el usuario.

- CE4.1. Se han clasificado y se han utilizado las funciones predefinidas del lenguaje.
- CE4.2. Se han creado y se han utilizado funciones definidas por el usuario.
- CE4.3. Se han reconocido las características del lenguaje relativas a la creación y al uso de *arrays*.
- CE4.4. Se han creado y se han utilizado *arrays*.
- CE4.5. Se han reconocido las características de orientación a objetos del lenguaje.
- CE4.6. Se ha creado código para definir la estructura de objetos.
- CE4.7. Se han creado métodos y propiedades.
- CE4.8. Se ha creado código que haga uso de objetos definidos por el usuario.
- CE4.9. Se ha depurado y se ha documentado el código.

RA5. Desarrolla aplicaciones web interactivas integrando mecanismos de manejo de eventos.

- CE5.1. Se han reconocido las posibilidades del lenguaje de marcas relativas a la captura de los eventos producidos.
- CE5.2. Se han identificado las características del lenguaje de programación relativas a la gestión de los eventos.
- CE5.3. Se han diferenciado los tipos de eventos que se pueden manejar.
- CE5.4. Se ha creado un código que capture y utilice eventos.
- CE5.5. Se han reconocido las capacidades del lenguaje relativas a la gestión de formularios web.
- CE5.6. Se han validado formularios web utilizando eventos.
- CE5.7. Se han utilizado expresiones regulares para facilitar los procedimientos de validación.
- CE5.8. Se ha probado y se ha documentado el código.

RA6. Desarrolla aplicaciones web, para lo que analiza y aplica las características del modelo de objetos del documento.

- CE6.1. Se ha reconocido el modelo de objetos del documento de una página web.
- CE6.2. Se han identificado los objetos del modelo, sus propiedades y sus métodos.
- CE6.3. Se ha creado y se ha verificado un código que acceda a la estructura del documento.
- CE6.4. Se han creado nuevos elementos de la estructura y se han modificado elementos ya existentes.
- CE6.5. Se han asociado acciones a los eventos del modelo.
- CE6.6. Se han identificado las diferencias que presenta el modelo en función de los navegadores.
- CE6.7. Se han programado aplicaciones web de modo que funcionen en navegadores con diferentes implementaciones del modelo.
- CE6.8. Se han independizado el contenido, el aspecto y el comportamiento en aplicaciones web.

RA7. Desarrolla aplicaciones web dinámicas, para lo que reconoce y aplica mecanismos de comunicación asíncrona entre cliente y servidor.

- CE7.1. Se han evaluado las ventajas y los inconvenientes de utilizar mecanismos de comunicación asíncrona entre cliente y servidor web.
- CE7.2. Se han analizado los mecanismos disponibles para el establecimiento de la comunicación asíncrona.
- CE7.3. Se han utilizado los objetos relacionados.
- CE7.4. Se han identificado las propiedades y los métodos de los objetos relacionados.
- CE7.5. Se ha utilizado comunicación asíncrona en la actualización dinámica del documento web.
- CE7.6. Se han utilizado distintos formatos en el envío y en la recepción de información.
- CE7.7. Se han programado aplicaciones web asíncronas de modo que funcionen en diferentes navegadores.
- CE7.8. Se han clasificado y se han analizado librerías que faciliten la incorporación de las tecnologías de actualización dinámica a la programación de páginas web.
- CE7.9. Se han creado y se han depurado programas que utilicen estas librerías.

7.6 Anexo F: Bloques de contenido

BC1. Selección de arquitecturas y herramientas de programación

- Modelos de programación en entornos cliente-servidor.
- Mecanismos de ejecución de código en un navegador web.
- Capacidades y limitaciones de ejecución de código en los navegadores web.
- Lenguajes de programación en entorno cliente.
- Tecnologías y lenguajes asociados.
- Integración del código con las etiquetas HTML.

BC2. Manejo de la sintaxis del lenguaje

- Variables y constantes: ámbito de uso.
- Tipos de datos: conversión entre tipos.
- Asignaciones.
- Operadores.
- Expresiones.
- Comentarios al código.
- Sentencias y bloques de sentencias.
- Decisiones.
- Bucles.
- Uso de entornos de desarrollo integrados.
- Depuración y documentación del código.

BC3. Uso de los objetos predefinidos del lenguaje

- Uso de objetos. Objetos nativos del lenguaje.
- Interacción con el navegador. Objetos predefinidos asociados.
- Generación de texto y elementos HTML desde código.
- Aplicaciones prácticas de los marcos.
- Gestión de la apariencia de la ventana.
- Creación de nuevas ventanas y comunicación entre ventanas.
- Uso de cookies.

BC4. Programación con *arrays*: funciones y objetos definidos por el usuario

- Funciones predefinidas del lenguaje.
- Llamadas a funciones. Definición de funciones.
- *Arrays*.
- Creación y utilización de objetos.
- Definición de métodos y propiedades.

BC5. Interacción con el usuario: eventos y formularios

- Modelo de gestión de eventos.
- Uso de formularios desde código.
- Modificación de apariencia y comportamiento.
- Validación y envío.
- Expresiones regulares en los procedimientos de validación de formularios.

BC6. Uso del modelo de objetos del documento

- Modelo de objetos del documento.
- Objetos del modelo: propiedades y métodos.
- Acceso al documento desde código.
- Creación y modificación de elementos del documento.
- Programación de eventos.
- Diferencias en las implementaciones del modelo.
- Independencia del contenido, aspecto y comportamiento de las aplicaciones web.

BC7. Uso de mecanismos de comunicación asíncrona

- Mecanismos de comunicación asíncrona.
- Objetos relacionados: propiedades y métodos.
- Modificación dinámica del documento utilizando comunicación asíncrona.
- Formatos para el envío y la recepción de información.
- Programación de aplicaciones con comunicación asíncrona.
- Librerías de actualización dinámica.

7.7 Anexo G: Dimensiones e indicadores para la evaluación

Dimensión	Indicador
MOS	Se fomenta la búsqueda de soluciones creativas
	Se fomenta el intercambio de conocimiento en red
	Se desmitifica la idea del programador Ninja, Rockstar o Jedi
	Existe espacio para el aprendizaje autónomo
	Existe espacio para el aprendizaje en red
	Se exponen suficientes ejemplos que ilustran los conceptos básicos
	Se secuencian los contenidos de forma lógica
	Se establecen referencias cruzadas con la materia del resto de ciclo
	Las clases sobre Git y GitHub han sido suficientes
	He mejorado mi capacidad para buscar soluciones de forma autónoma
	He aprendido a trabajar sin disponer, de antemano, de toda la información necesaria
MOS WORKFLOW	La conversación informal diaria es útil para expresar los problemas y limitaciones enfrentadas
	La conversación informal diaria es útil para informarse de los problemas del resto de los compañeros
	Las clases expositivas en el REPL ayudan a comprender los conceptos básicos
	La retrospectiva evaluable es útil para perder el miedo a hablar en público
	La retrospectiva evaluable es útil para aprender de los demás
	La retrospectiva evaluable refleja fielmente mis conocimientos
	La prueba evaluable sobre el REPL refleja fielmente mis conocimientos

Dimensión	Indicador
	Los apuntes colaborativos me han ayudado a reflexionar sobre mi proceso de aprendizaje
	Los apuntes colaborativos me han resultado de utilidad y los he consultado habitualmente
	Los apuntes colaborativos evaluables reflejan fielmente mis conocimientos
	El MOS *Workflow* es satisfactorio
ACTITUD DOCENTE	El docente ha permitido a los alumnos trabajar con suficiente autonomía
	El docente ha estado atento para resolver los inconvenientes que han ido surgiendo en el día a día
	El docente ha ayudado a que los alumnos valoren su proceso de aprendizaje
	El docente no lo sabía todo y su actitud en estos casos ha sido positiva favoreciendo la búsqueda de soluciones en conjunto
	El docente ha ejercido satisfactoriamente su labor como mentor
	El docente ha ejercido satisfactoriamente su labor como facilitador
EVALUACIÓN	La evaluación ha sido suficientemente diversa
	La evaluación refleja el desempeño del alumnado
	La evaluación ayuda a reflexionar y orienta a la mejora
	La evaluación ha sido suficientemente consensuada con el alumnado

Tabla 8: *Relación de dimensiones e indicadores para la evaluación de la propuesta.* Fuente: Elaboración propia.

La mayor parte de los indicadores de la *tabla 8* son auto explicativos y su utilidad evidente. Sin embargo, es interesante hacer una pequeña reflexión sobre:

- Se desmitifica la idea del programador Ninja, Rockstar o Jedi.
- He aprendido a trabajar sin disponer, de antemano, de toda la información necesaria.

- El docente no lo sabía todo y su actitud en estos casos ha sido positiva favoreciendo la búsqueda de soluciones en conjunto.

En realidad, se trata de tres caras del mismo cubo. Lo que se pretende con ellos es tener una muestra real sobre cómo se ha trabajado la prevención del *Impostor Syndrome*. Si la puntuación conjunta es buena se podría concluir que efectivamente se ha trabajado satisfactoriamente.

Antes de que los alumnos ofrezcan su valoración, deberían ser informados de que la escala que deben usar (Likert, 1932) va de 1 a 5 con el siguiente significado:

1. En total desacuerdo.
2. En desacuerdo.
3. Ni de acuerdo ni en desacuerdo.
4. De acuerdo
5. Totalmente de acuerdo.

7.8 Anexo H: Ejemplos de rúbricas para Temporalización 2

Evidencia	Muy alto (10-9)	Alto (8-7)	Medio (6-5)	Bajo (4-0)
Retrospectiva (5 %)	Han presentado de forma correcta	Han presentado, pero no han sido capaces de aclarar las dudas planteadas	Han presentado sólo en parte	No han hecho la presentación
Wiki de apuntes (15 %)	Han contribuido información relevante	Han contribuido información no relevante	Han contribuido información incompleta	No han contribuido
Prueba REPL (15 %)	Ha comprendido y maneja con soltura los conceptos básicos	Ha comprendido, pero le cuesta manejar con soltura algunos conceptos básicos	No ha comprendido y le cuesta manejar algunos conceptos básicos	No ha comprendido los conceptos básicos
MOS Worflow (65 %)	Muy alto (10-9)	Alto (8-7)	Medio (6-5)	Bajo (4-0)

Tabla 9: *Ejemplo de rúbrica para Temporalización 2*. Fuente: Elaboración propia.

Evidencia MOS Workflow	Muy alto (10-9)	Alto (8-7)	Medio (6-5)	Bajo (4-0)
Pull Requests (60 %)	Han solucionado correctamente una parte significativa de la *Issue* planteada	Han solucionado correctamente una parte no muy significativa de la *Issue* planteada	Han solucionado parcialmente aspectos de la *Issue* planteada	No han solucionado aspectos de la *Issue* planteada
Issues (30 %)	Han contribuido de forma correcta al análisis del problema	Han contribuido de forma parcialmente correcta al análisis del problema	Han contribuido de forma incorrecta correcta al análisis del problema	No han contribuido al análisis del problema
PRs e Issues de otros grupos (10 %)	Han contribuido significativamente a *Issues* o PRs de otros grupos	Han contribuido poco a *Issues* o PRs de otros grupos	Han interaccionado de forma informal con los otros grupos	No han interaccionado con los otros grupos

Tabla 10: *Ejemplo de rúbrica para evaluar MOS Workflow en Temporalización 2.* Fuente: Elaboración propia.