Una Plataforma de Integración Continua Especializada en Desarrollo de Videojuegos

Iván Martínez-Mateu
Research & Development
Taiger
Madrid, España
ivan.martinez@taiger.com

Federico Peinado
Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
Madrid, España
email@federicopeinado.com

***Resumen*—Los estudios de desarrollo de videojuegos invierten mucho esfuerzo en mejorar la calidad de sus productos, beneficiándose significativamente de toda metodología y herramienta que les permita garantizar la integridad de sus procesos de trabajo con el menor coste posible. Desde hace años la industria del software cuenta con paradigmas y tecnologías que facilitan el trabajo a sus equipos de desarrollo, automatizando la mayoría de las tareas que anteriormente eran tediosas y originaban retrasos y errores. Este es el caso de la integración continua, una práctica de desarrollo en la que todos los miembros del equipo integran sus contribuciones como mínimo una vez al día, siendo cada integración verificada y probada exhaustivamente de forma automática para detectar lo antes posible cualquier conflicto. Tras detectar que esta práctica no ha calado lo suficiente entre los desarrolladores independientes de videojuegos, se propone cambiar la situación mediante el proyecto de construcción de una plataforma de integración continua orientada hacia el desarrollo, las pruebas y la publicación de videojuegos. Como resultado se obtiene la primera versión funcional de una herramienta basada en microservicios web que se presenta como libre, gratuita y de fácil manejo para los desarrolladores.**

***Palabras clave*—Desarrollo de Videojuegos, Informática del Entretenimiento, Metodologías Ágiles, Ingeniería del Software, Tecnologías Web.**

I. Introducción

En el ciclo de vida del software, todo proyecto pasa por diferentes procesos de análisis, diseño, implementación y pruebas. Lo habitual en un proyecto de alta calidad es tener que realizar miles de pruebas distintas en múltiples plataformas o configuraciones diferentes, todo ello para cada versión generada. Realizar manualmente todas estas tareas es costoso y perjudicial para la integridad y fiabilidad del proceso.

En la última década la industria del software ha promovido el uso de procedimientos y tecnologías, asociadas a las llamadas metodologías ágiles, que facilitan el trabajo a sus equipos de desarrollo, automatizando muchas tareas que antes debían realizarse a mano, originando retrasos, errores y conflictos de todo tipo.

Concretamente nos interesa el caso de la integración, la entrega y el despliegue continuo de software, un conjunto de prácticas de desarrollo en las que todos los miembros del equipo incorporan sus contribuciones al producto final como mínimo una vez al día, siendo cada integración, cada entrega y cada despliegue debidamente verificado y probado exhaustiva y automáticamente, con el fin de detectar cualquier problema lo antes posible.

El escenario típico comienza cuando el desarrollador sube los cambios del código fuente al repositorio común. Periódicamente la plataforma de integración, entrega y despliegue continuo está analizando el repositorio para detectar si se ha producido un cambio. Por ello, en cuanto se han remitido dichos cambios (*commit*) al repositorio, la plataforma los detecta y procede a descargar la última versión de todo el código fuente del proyecto. Una vez descargado este código, la plataforma ejecuta los *scripts* de construcción de proyecto. En estos *scripts* se indica qué pasos se deben seguir para compilar y ejecutar los tests del proyecto (por ejemplo, si es necesario descargar dependencias, utilizar alguna directiva de precompilación (*flag*), etc.). El resultado de la ejecución de estos *scripts* se enviará por correo electrónico o por otro método de comunicación a los integrantes del equipo de desarrollo para informarles del estado de la construcción automática, es decir, si se ha producido un error o si por el contrario todo marcha bien.

Aunque los estudios de desarrollo de videojuegos están a la última en tecnología e invierten mucho esfuerzo en mejorar la calidad de sus productos, hemos detectado (como se detallará más adelante en la Discusión) que la práctica de la integración continua es a menudo desconocida o confundida con otras prácticas populares por los desarrolladores independientes de videojuegos.

El ámbito del desarrollo de videojuegos puede verse como un subproblema del desarrollo de *software*, aunque muy peculiar [15] [11]. Entre lo específico de los proyectos de videojuegos está la variedad de talento humano que interviene en el proceso de creación y desarrollo, personas con diferente nivel de conocimiento que a menudo no suelen estar concienciados con los procesos de integración continúa. Un desconocimiento que puede suponer complicaciones ya que en un videojuego, un fallo no sólo se puede producir por un error en el código, sino también por un fichero de audio corrupto o un modelo 3D cuya calidad es excesivamente alta y provoca que el juego no gestione correctamente los recursos para representarlo visualmente por pantalla. Además, el desarrollador está llamado a optimizar recursos: el juego suele ocupar mucho espacio y consumir muchos recursos (lo

cual afecta claramente a su alojamiento y a la forma de trabajar con él) y dinámicamente, carga y descarga recursos muy pesados en tiempo real. Además de los controles de calidad y pruebas de *UX* normales, los videojuegos también requieren de *playtesting*, es decir, comprobar que son divertidos, no demasiado difíciles para superar ciertos niveles, con una dificultad ajustable gradualmente, etc. Una plataforma de integración, entrega y despliegue continuo para el desarrollo de videojuegos vendría a ser una herramienta que tratase de automatizar todas las pruebas y compilaciones que se realicen, de notificar de aquellos elementos del juego que consumen muchos recursos, etc. y, en definitiva, de lograr aumentar la calidad del producto sin incurrir en elevados costes temporales y económicos.

Consideramos importante trata de cambiar esta situación y para ello en este artículo proponemos una plataforma de integración continua especializada en el desarrollo, la prueba y la publicación de videojuegos, herramienta que es contribución principal de un Trabajo Fin de Máster de reciente aprobación [9].

A. Objetivos

Estos son los objetivos principales del proyecto:

- Simplificar la instalación, configuración y uso de la tecnología de integración continua para los desarrolladores de videojuegos.
- Especializar las tareas habituales del proceso de integración continua para que resulten más prácticas y conecten mejor con el flujo de desarrollo, prueba y publicación de un videojuego.
- Como consecuencia, reducir el tiempo que se tarda en tener listas nuevas versiones y mejorar la calidad de las mismas.

B. Estructura del artículo

El artículo presenta la siguiente estructura: tras esta primera sección de Introducción, se presenta una revisión del concepto y las herramientas de integración continua. A esta sección le sigue otra sobre la metodología utilizada, incluyendo lenguaje y tecnologías de desarrollo, y otra con los resultados obtenidos, la propia plataforma, y con la discusión sobre su enfoque y utilidad práctica. Finalmente en la sección de Conclusiones destacamos los aspectos más prometedores del proyecto y esbozamos las futuras líneas a seguir para mejorarlo.

II. Estado de la técnica

Las ventajas de usar integración continua hoy día, una práctica originalmente propuesta en 2001 con la creación de CruiseControl [12], son claras dentro de la disciplina de la Ingeniería del Software [3]. Una de las partes importantes de la automatización de este tipo de procesos es que ofrece uniformidad, seguridad y garantía de que todas las pruebas se van a ejecutar de la misma forma. Esto se refiere tanto a la automatización de las pruebas de unidad, que se ejecutan antes de construir la versión del software, como a la etapa de pruebas funcionales.

En el caso de los videojuegos es cierto que uno de los entornos de desarrollo más populares, Unity, cuenta con un sistema para la construcción automática de versiones llamado Unity Cloud Build [18], hoy día rebautizado como Unity Teams. Esta solución es gratuita en principio pero va resultando más costosa según el tamaño del equipo y las prestaciones de hardware y software necesarias.

Desgraciadamente no todos los entornos de desarrollo tienen un servicio de automatizacion como este. Cuando esto ocurre, se suelen usar sistemas externos populares y versátiles como puede ser Jenkins [7] o Travis [17], para la verificación de código. Esta es probablemente la herramienta más popular para automatizar procesos de construcción en la industria de los videojuegos, se trata de una plataforma sencilla donde es fácil definir trabajos y tiene además una comunidad muy activa. Además el código se construye automáticamente en un solo paso, con lo que usuarios -expertos, eso sí- pueden construir el videojuego para varias plataformas en un sólo paso. Jenkins además ha incorporado recientemente un editor visual de pipelines, Blue Ocean, que facilita su uso por miembros del equipo que no tengan conocimientos de programación.

Existen además otros sistemas como TeamCity [8], donde los proyectos C# se integran con mayor facilidad, lo cual es interesante para el desarrollador de Unity. En cualquier caso los sistemas de integración continua que existen actualmente o son gratuitos y relativamente complejos de administrar y mantener, o son de pago, y tienen licencias costosas que a menudo son difíciles de asumir por muchos desarrolladores independientes de videojuegos.

Los sistemas de integración continua se apoyan a su vez en sistemas de control de versiones como es el caso del clásico Concurrent Versions System (CVS) [2], que mantienen un registro minucioso del trabajo realizado y permiten que varios desarrolladores colaboren a la vez sobre un mismo proyecto. Su representante actual sería Subversion (SVN) [16], un sistema centralizado de control de versiones de código abierto, cuyo funcionamiento se asemeja al de un sistema de ficheros. Otro igual de popular es Git [5], diseñado por Linux Torvalds para proyectos con gran cantidad de código fuente. Finalmente Mercurial [10], por su parte, ofrece un diseño que proporciona una gran escalabilidad y rendimiento en la gestión de archivos tanto de texto como binarios, con capacidades avanzadas de ramificación e integración, a la par que manteniendo cierta sencillez conceptual.

III. Metodología y Desarrollo

Para el diseño de la plataforma propuesta se han tomado referencias de las características más destacadas de las herramientas del estado de la técnica que aparecen en la Tabla I.

La plataforma está desarrollada utilizando Java [6] como plataforma tecnológica, lo que permite desplegarla en los sistemas operativos más utilizados, y utiliza además el armazón Spring Boot [13] para ofrecer una arquitectura basada en microservicios. De esta forma, está asegurado que la plataforma pueda escalar horizontalmente de forma sencilla de acuerdo a la carga de trabajo y que tenga una alta disponibilidad.

	Jenkins	Circle	Travis	Unity Cloud	TFS	GameCraft
Soporta *macOS*	✓	✓	✓	✓		✓
Soporta *GNU/Linux*	✓	✓	✓	✓		✓
Soporta *Windows*	✓			✓	✓	✓
Gratuito	✓					✓
Código abierto	✓					✓
Soporte a plug-ins	✓	✓	✓		✓	
Soporte a *Unity*				✓		✓
Soporte a *Unreal*						✓
Soporte a *Monogame*						✓
Soporte a *Libgdx*						✓
Soporte a otros *motores*	✓	✓	✓		✓	✓
QA	✓	✓	✓		✓	

Tabla I
COMPARACIÓN DE LAS PLATAFORMAS DE INTEGRACIÓN CONTINUA ANALIZADAS

También se configura así como un sistema abierto que permite incluir funcionalidades desarrolladas por terceros.

La plataforma recibe el nombre de GameCraft y su arquitectura se representa de forma esquemática en la Fig. 1. A continuación, se explica cada microservicio del sistema:

- *Registry*: Tiene tres labores principales: Es un servidor *Eureka* [4], cuyo propósito es descubrir la dirección IP de los microservicios y ubicarlos en la red y es un servidor de *Spring Cloud Config* [14], cuyo propósito es centralizar y distribuir la configuración inicial para que cada microservicio pueda funcionar correctamente.
- *Gateway*: Sirve para hacer posible el encaminamiento HTTP y el balanceo de carga entre los diferentes microservicios, mantener la calidad de servicio, ofrecer seguridad y documentación API para todos los microservicios. Al igual que el microservicio de registro, el microservicio de enrutamiento tiene un panel de administración donde además se puede acceder a las bases de datos registradas por cada microservicio y manipularlas, gestionar los usuarios registrados en la plataforma, etc. Este microservicio utiliza el armazón *Zuul Proxy* [19] para posibilitar todas las funciones anteriormente descritas.
- *Notificador de e-mail*: Se encarga de conectarse con un servidor de correo electrónico para poder enviar *e-mails* acerca del estado de ejecución de un *pipeline*.
- *Notificador de Slack*: Se encarga de conectarse con un *bot* de *Slack* para poder enviar mensajes a través de este medio acerca del estado de ejecución de un *pipeline*.
- *Administrador de pipelines*: Permite crear y almacenar los *pipelines* del sistema, además de ejecutarlos.
- *Notificador de Telegram*: Se encarga de conectarse con un *bot* de *Telegram* para poder enviar mensajes a través de este medio acerca del estado de ejecución de un *pipeline*.
- *Notificador de IRC*: Se encarga de conectarse con un servidor de *IRC* para poder enviar mensajes acerca del estado de ejecución de un *pipeline*.
- *Notificador de Twitter*: Se encarga de conectarse con una cuenta de *Twitter* para poder publicar *tuits* acerca del estado de ejecución de un *pipeline*.
- *Administrador de motores*: Permite crear, almacenar e invocar los motores, es decir, los entornos integrados de desarrollo para cada proyecto, como por ejemplo, *Unity* o *gcc*.
- *Administrador de proyectos*: Permite crear y almacenar proyectos (es decir, los videojuegos que se quieren compilar y procesar) dentro del sistema.

El usuario accede a través de la interfaz de usuario de la plataforma, que es un microservicio más de la arquitectura. Para poder manejar los datos procedentes de otros microservicios desde la interfaz de usuario, las peticiones primero se encaminan hacia el *Gateway*, que se encarga de validar que la petición procede de un usuario autenticado, y redirige la petición al microservicio adecuado. De acuerdo con la arquitectura de microservicios, cada microservicio debe ser independiente, y por tanto, este es el motivo de que cada uno tenga bases de datos independientes. Siempre que se quiere obtener un dato, por ejemplo, de la base de datos de proyectos, hay que hacer uso de la *API Rest* diseñada en el microservicio de proyectos.

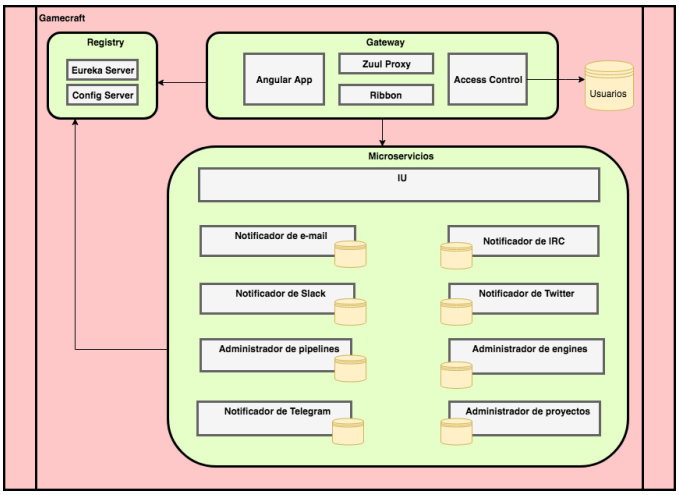

Fig. 1. Esquema de la arquitectura lógica de GameCraft.

El proceso de desarrollo seguido ha sido iterativo e incremental, para lo cual ha sido de gran ayuda la gran modularidad de la propuesta.

IV. Resultados y Discusión

Como parte del proceso de diseño de la plataforma, se realizó una consulta en línea a 17 desarrolladores de videojuegos

españoles con el fin de conocer su nivel de conocimiento e interés por una práctica del desarrollo industrial de proyectos como es la integración continua. Los resultados de la encuesta[1] se adjuntan a continuación, agrupando todas las respuestas obtenidas en cada una de las preguntas:

- **Q1 - Tengo experiencia usando herramientas de integración continua en el trabajo.**

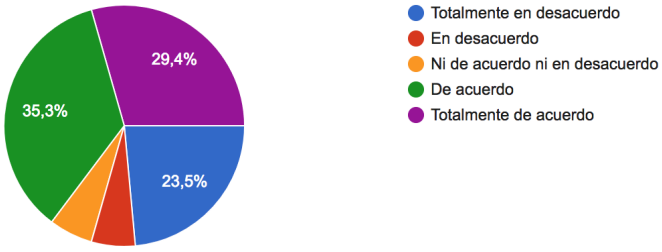

Fig. 2. Resumen de las respuestas obtenidas a la primera pregunta de la encuesta

- **Q2 - ¿Qué herramientas de integración continua has utilizado?**
  - SVN, Git. (2)
  - Mercurial, Git y Perforce.
  - En mi trabajo en el estudio de videojuegos ninguna, en el otro trabajo me han llegado a explicar el uso de Jenkins.
  - Jenkins pero solo para probar. Mi empresa no usa herramientas de este tipo por desgracia.
  - TFS.
  - MercuryEngine (motor propio de Mercury).
  - AgileCraft, Scrum, JIRA.
  - Bitbucket, con cliente TortoiseHG para mercurial y SourceTree para Git.
  - SVN, Git, Perforce, Mercurial.
  - Git, SVN, Mercurial.
  - Jenkins, Gitlab CI.
  - Perforce, tortoise svn.
  - Perforce.
  - Jenkins pipeline.
  - Ninguna.
  - AppVeyoy y TravisCI con GitHub.

- **Q3 - Considero que la integración continua es un asunto exclusivo de los programadores.**

[1]https://docs.google.com/forms/d/1UljfP9vSApvScdg8iJBqTTE7OkZXv-e_w25z9RbhQ4U

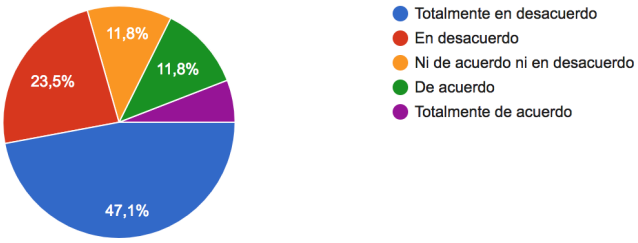

Fig. 3. Resumen de las respuestas obtenidas a la tercera pregunta de la encuesta

- **Q4 - Considero que mi empresa actual da importancia a la integración continua.**

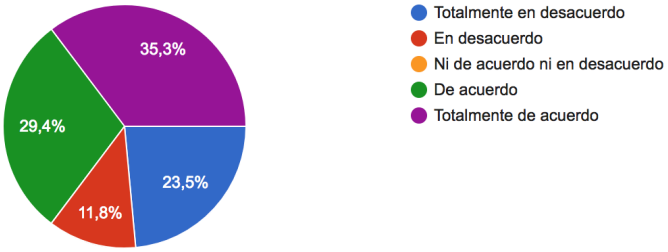

Fig. 4. Resumen de las respuestas obtenidas a la cuarta pregunta de la encuesta

- **Q5 - Considero que la integración continúa debería ser más importante en mi empresa actual.**

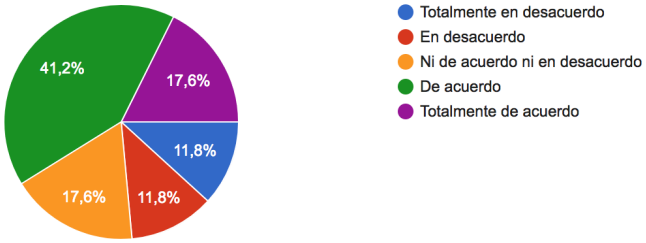

Fig. 5. Resumen de las respuestas obtenidas a la quinta pregunta de la encuesta

- **Q6 - En el uso profesional de las herramientas de integración continua, ¿cuales son los principales obstáculos y dificultades?**
  - El trabajo con archivos binarios y las herramientas, a veces, poco intuitivas.
  - Mentalizar a TODO el equipo de que cualquier cosa que se suba al repositorio debe ser probada, aún si no hay herramientas de integración continua. Entender bien cómo se puede aplicar la integración continua en videojuegos de cierto tamaño porque no lo veo nada claro más allá de que "compila, luego está OK" o, a lo sumo, pruebas unitarias tontorronas para métodos sueltos de cierta complejidad.

  - No querer aprender por parte de mis superiores.
  - Se requiere de un equipo potente -y dedicado exclusivamente a esta tarea- para que las pruebas se ejecuten con la rapidez necesaria. A medida que el proyecto crece, cada ciclo requiere más tiempo. Por lo que es útil que la herramienta acepte diferentes parámetros como hacer build o re-build del proyecto, poder precompilar scripts como los .lua, seleccionar solo ciertas secciones de código para la integración continua (por ejemplo X niveles de un juego en vez de todos), etc. Por la misma causa que el punto anterior, puede llegar a ser necesario tener 2 máquinas dedicadas a integración continua, cada una encargándose de una parte diferente (o incluso tener varias integraciones continuas en cada máquina). La integración continua puede llegar a ser una responsabilidad del equipo de *QA*. Pero esto requiere que el responsable tenga conocimientos técnicos para interpretar los errores y aumentar la eficiencia de este proceso.
  - Dificultad de Implantación, cuestionables ventajas.
  - Curva de aprendizaje para los no técnicos. Problemas de *merges*.
  - Integración de archivos binarios.
  - La adaptación de un usuario primerizo al uso de herramientas como estas. Normalmente suelen generar problemas durante cierto tiempo. A veces pueden generar caos por no realizar correctamente el protocolo exigido.
  - El aprendizaje de los novatos.
  - *Merging* y *branching*.
  - El uso de lenguajes *script* sin compilador causa muchas iteraciones hasta que el *script* funciona
  - Entender la filosofía y aplicarlo de forma continua.
  - En nuestra experiencia ha sido la instalación y configuración inicial del servidor.
  - My Spanish is not good, but I can read. The main difficulties is that developers do not understand the importance of CI, and not willing to write unit tests.
  - La comunicación.
  - Setup para multiples plataformas "no estandard".
- **Q7 - ¿Qué características y facilidades valorarías más en una hipotética herramienta de integración continua?**
  - Que sea gratis. (7 votos)
  - Que sea de código abierto. (4 votos)
  - Que se integre fácilmente con *Unity* o *Unreal*. (8 votos)
  - Que esté bien documentado. (9 votos)
  - **Que sea fácil de instalar y configurar. (13 votos)**
  - Que sea fácilmente escalable. (8 votos)
  - Que esté tanto en español como en inglés. (3 votos)
  - Que ofrezca servicios adicionales para la mejora de productividad como analizador de calidad de código. (4 votos)
  - Que su funcionalidad pueda ampliarse mediante *plug-ins.* (7 votos)
  - Otro: Multiplataforma. (1 voto)
  - Otro: Compatible con Docker. (1 voto)
- **Q8 - ¿Qué aspectos y funcionalidades valoras más en las herramientas de integración continua que ya has utilizado?**
  - Soporte para *macOS*.(7 votos)
  - Soporte para *GNU/Linux*. (4 votos)
  - **Soporte para *Windows*. (12 votos)**
  - Que sea gratuito. (7 votos)
  - Que sea de código abierto. (2 votos)
  - Que esté traducido en varios idiomas. (1 voto)
  - Que pueda integrarse con *plug-ins* de terceros. (3 votos)
  - Que tenga soporte nativo al motor *Unity*. (4 votos)
  - Que tenga soporte nativo al motor *Unreal*. (4 votos)
  - Que tenga soporte nativo al motor *Monogame*. (0 votos)
  - Que tenga soporte nativo al motor *Libgdx*. (0 votos)
  - Que cuente con un buen analizador de calidad de código. (4 votos)
  - Que tenga buena documentación. (5 votos)
  - Que sea fácil de instalar. (10 votos)
  - Que sea fácil de configurar. (13 votos)
  - Otro: Que sean muy adaptables al proyecto en desarrollo. (1 voto)
  - Otro: Poder excluir fácilmente archivos a sincronizar. (1 voto)
  - Otro: Que sea fácil de usar. (1 voto)

En el cuestionario había preguntas relativas a la experiencia con herramientas de integración continua en su trabajo, De los 17, 11 afirmaban tener experiencia, pero de estos sólo 3 contestaban correctamente cuando se les preguntaba por las herramientas que habían utilizado. El resto sorprendentemente confundían la integración continua con el mero hecho de usar sistemas de control de versiones como SVN o Git.

Esos resultados recuerdan, salvando las distancias, a los de Angioni et al. [1], quienes al describir una metodología ágil para el desarrollo distribuido compararon similitudes, diferencias y aplicabilidad de las distintas prácticas usadas en el desarrollo de software de código abierto, descubriendo que un 72% de los desarrolladores conocían estas prácticas, pero que menos del 10% usaban algunas de las más básicas como programar en pareja o realizar iteraciones cortas.

De los 17 encuestados sólo 3 opinaban que el asunto de la integración continua es un problema exclusivo de los programadores, y 8 consideraban que su empresa debería darle más importancia a esta práctica,

Del análisis de las respuesta a la encuesta realizada se pueden extraer varias ideas:

- Algunos usuarios no saben distinguir realmente lo que es una herramienta de integración continua, poniendo herramientas y tecnologías que no son tales como Git, SVN o Bitbucket. Esto se debe a que herramientas de este tipo no se aplican en muchas organizaciones y empresas, y por tanto, los usuarios no tienen claro qué responder ya que no han conocido o no han tenido la oportunidad de utilizar estas herramientas.
- Existe una mayoría que piensa que la integración continua debería ser más importante en el lugar de trabajo. Esto permite afirmar que la gente es consciente de algunas de las ventajas que el uso de estas herramientas puede ofrecer al equipo y posiblemente no dudarían en recomendar herramientas como Jenkins o la propia GameCraft a sus superiores si eso les ayuda en el trabajo.
- La mayoría pide que una herramienta de este tipo sea fácil de instalar y configurar, lo que confirma y justifica una de las prioridades de diseño de GameCraft.
- Algunas personas encuestadas no vieron las ventajas de GameCraft frente al resto de alternativas que existen en el mercado. Si bien es cierto que se pueden utilizar herramientas como Jenkins o Travis CI para poder tratar proyectos de videojuegos, su dificultad en la instalación y configuración, el hecho de que muchas de las soluciones

existentes en el mercado son de pago y que no haya una solución desarrollada a partir de una arquitectura de microservicios, hacen que GameCraft sea una alternativa interesante frente a herramientas ya asentadas. Además, GameCraft es un proyecto gratuito y de código abierto, por lo que puede ser modificado para cualquier entorno de trabajo sin mucha complejidad.

Los sistemas gratuitos como Jenkins, que es el más conocido por los desarrolladores que sí conocen lo que es la integración continua, tienen una curva de aprendizaje y un coste de mantenimiento elevados para el administrador por la gran cantidad de opciones que existen en la plataforma. Los videojuegos tienen algunas peculiaridades, como la multidisciplinaridad de sus miembros en el equipo de desarrollo, la complejidad de los entornos de desarrollo integrados que se suelen utilizar como base, el tamaño en disco de sus proyectos y la cantidad de ficheros con información binaria y muy pesada que estos manejan. Al especializar nuestra plataforma para este dominio, se reducen significativamente todas esas opciones, quedándonos con las estríctamente necesarias para el desarrollo de videojuegos.

La plataforma GameCraft está actualmente publicada bajo licencia de código abierto[2].

V. Conclusiones

Este proyecto tiene el propósito de encontrar maneras de fomentar buenas prácticas de desarrollo en las empresas de videojuegos, que les ayuden a mejorar la calidad de sus productos, aumentar la productividad y reducir el coste de sus procesos. Habiendo detectado que la integración continua es todavía una gran desconocida para muchos desarrolladores de videojuegos y que las necesidades en este ámbito no son realmente las mismas que cabría esperar de cualquier ámbito del software, se propone la creación de una plataforma de integración continua especializada para este tipo de creaciones.

Como resultado se obtiene la primera versión funcional de una herramienta basada en microservicios web que ha sido publicada libre, gratuita y que ofrece un interfaz web sencillo a sus usuarios. Con la plataforma GameCraft se pone a disposición de todos una herramienta gratuita, con una interfaz de usuario sencilla y adaptada a los procesos de desarrollo de videojuegos, escalable a cualquier tamaño de un proyecto y de código abierto, por lo que puede adaptarse a las necesidades de cualquier desarrollador.

A. *Trabajo futuro*

Entre los ambiciosos objetivos que se plantean para el proyecto a medio y largo plazo están:

- Realizar pruebas con usuarios y proyectos reales, a ser posible en entornos de desarrollo de producción.
- Ampliar el enfoque e ir más allá de la integración continua, camino hacia la automatización del proceso completo, no sólo de integración, sino también de entrega y despliegue de videojuegos.
- Incorporar aprendizaje automático para aprovechar los datos de realización de trabajos y poder así anticiparse a problemas que estén próximos a ocurrir.

[2]https://github.com/iMartinezMateu/gamecraft

Agradecimientos

Este trabajo se apoya parcialmente en el proyecto ComunicArte: Comunicación Efectiva a través de la Realidad Virtual y las Tecnologías Educativas, financiado por las Ayudas Fundación BBVA a Equipos de Investigación Científica 2017, y el proyecto NarraKit VR: Interfaces de Comunicación Narrativa para Aplicaciones de Realidad Virtual (PR41/17-21016), financiado por las Ayudas para la Financiación de Proyectos de Investigación Santander-UCM 2017.

Referencias

[1] Angioni, M., Sanna, R., Soro, A. (2005). Defining a Distributed Agile Methodology for an Open Source Scenario. In Scotto, M., Succi, G. (eds.): Proceedings of the First International Conference on Open Source Systems. Springer Verlag pp. 209-214.
[2] Concurrent Versions System (2018). Concurrent Versions System. http://www.nongnu.org/cvs/
[3] Duvall, P., Matyas, S. M., Glover, A. (2007). Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series). Addison-Wesley Professional.
[4] Eureka (2018). Eureka. https://github.com/Netflix/eureka
[5] Git (2018). Git. https://git-scm.com
[6] Java (2018). Java. https://www.java.com
[7] Jenkins (2018). Jenkins. https://jenkins.io
[8] JetBrains (2018). TeamCity. https://www.jetbrains.com/teamcity
[9] Martínez Mateu, I. (2018). GameCraft: Una plataforma de integración continua para videojuegos basada en microservicios. Trabajo Fin de Máster en Ingeniería Informática. Universidad Complutense de Madrid.
[10] Mercurial (2018). Mercurial. https://www.mercurial-scm.org
[11] Scacchi, W. (2018). Free and Open Source Development Practices in the Game Community.
[12] Sharma, A. (2018). A Brief History of DevOps, Part III: Automated Testing and Continuous Integration. https://circleci.com/blog/a-brief-history-of-devops-part-iii-automated-testing-and-continuous-integration/
[13] Spring Boot (2018). Spring Boot. https://spring.io/projects/spring-boot
[14] Spring Cloud Config (2018). Spring Cloud Config. https://cloud.spring.io/spring-cloud-config/
[15] Stacey, P., Nandhakumar J. (2004). Managing the Development Process in a Games Factory : A Temporal Perspective.
[16] Subversion (2018). Subversion. https://subversion.apache.org
[17] Travis CI (2018). Travis. https://travis-ci.org
[18] Unity (2018). Unity Cloud Build. https://unity3d.com/learn/tutorials/topics/cloud-build/
[19] Zuul Proxy (2018). Zuul Proxy. https://github.com/Netflix/zuul