

Metodologías Ágiles de Producción

Guillermo Jiménez Díaz (gjimenez@ucm.es)

Curso 2019-2020

Prefacio

Estos son los apuntes de la asignatura Metodologías Ágiles de Producción, impartida en la Facultad de Informática de la Universidad Complutense de Madrid por el profesor Guillermo Jiménez Díaz, del Departamento de Ingeniería del Software e Inteligencia Artificial.

Este material ha sido desarrollado a partir de distintas fuentes, destacando como referencia principal el libro *Agile Game Development with Scrum* de Clinton Keith y los apuntes de asignaturas similares impartidas por Pedro Pablo Gómez Martín y Virginia Francisco Gilmartín.

Metodologías Ágiles de Producción (en dos semanas)

En el principio de los tiempos (del desarrollo de videojuegos), un único individuo (generalmente un ingeniero electrónico) era el encargado de definir las bases del juego, programarlo, crear el arte y el sonido para terminar construyendo un videojuego completo de principio a fin. Se desarrollaba un único juego para una única plataforma ya que el juego en sí se implementaba por hardware.

Con la llegada de los microprocesadores, los fabricantes se dieron cuenta de que podían reutilizar el hardware para múltiples juegos, programándolos en software, lo que dio lugar a las placas programables de las máquinas de arcade o a los cartuchos de consola. Los desarrolladores se convirtieron en programadores de videojuegos y seguían pudiendo desarrollar un videojuego solos en unos pocos meses.

El desarrollo de un videojuego para una máquina recreativa podía costar unos 3000\$, mientras que un buen videojuego podía generar unos 1000\$ semanales en monedas. Un buen juego daba margen para financiar muchos fracasos por lo que se podían desarrollar videojuegos de manera rápida dando poca importancia a si tenían éxito o no.

Sin embargo, con el paso del tiempo, el incremento de la complejidad de los sistemas para los que se desarrollan videojuegos (PCs, consolas, móviles...) y las expectativas de los consumidores ha hecho que el coste y el número de personas que han de trabajar en el desarrollo de videojuegos haya aumentado drásticamente en la última década (Figura 1.1).

A día de hoy, los videojuegos son desarrollados por cientos de personas y, algunos, con presupuestos incluso superiores a muchas superproducciones de cine. Por ejemplo, GTA IV fue un desarrollo de unas 150 personas durante 4 años y costó más de 100 millones de dólares, mientras que el coste de GTA V ha ascendido hasta los 265 millones (Black Panther tuvo un presupuesto de 200 millones de dólares). Esto, unido al enorme número

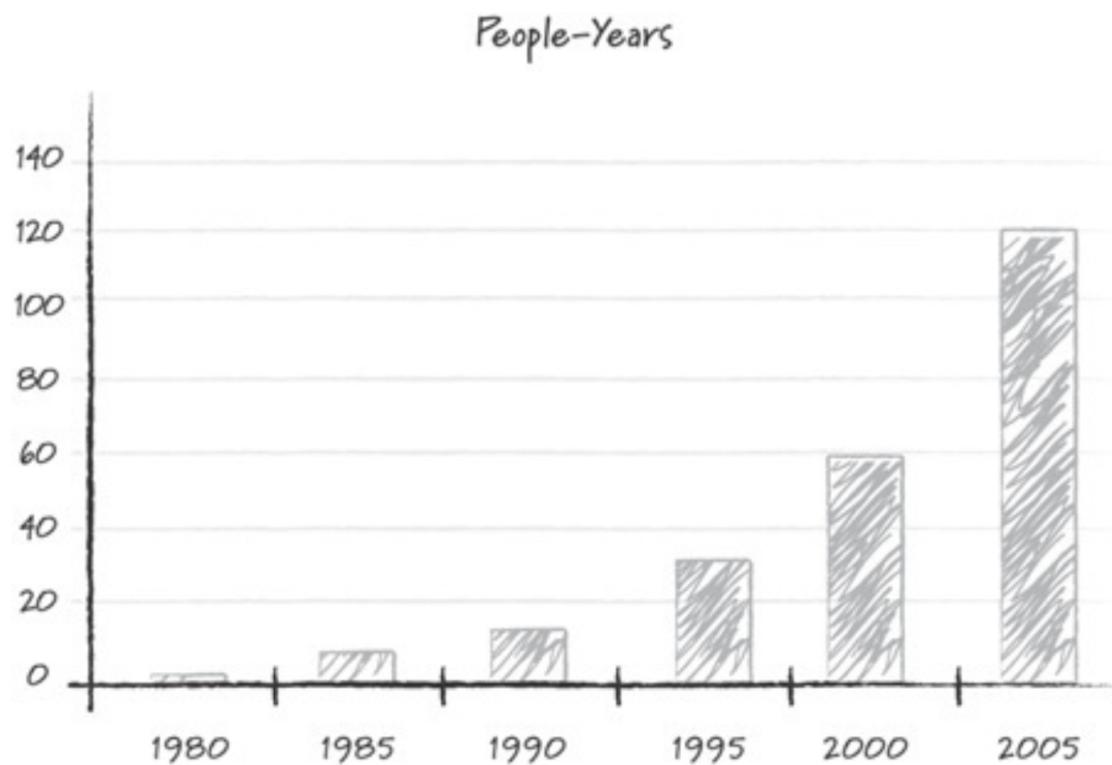


Figura 1.1: Coste en personas/año a lo largo de las últimas décadas

de juegos publicados anualmente, hace que el margen de error sea mucho más pequeño y que haya un mayor miedo al fracaso. Esto implica que:

- En general, hay menos innovación. Se suele apostar por las licencias y las secuelas.
- Los videojuegos son más cortos (menor tiempo de gameplay) ya que de esta forma se abaratan costes (ya que el precio del juego ha seguido siendo el mismo).
- Los grupos de desarrollo se van deteriorando si no se hace una gestión correcta, ya que las horas extras (*crunch*) se disparan, lo que hace que muchos desarrolladores terminen abandonando esta industria para tener una vida fuera de ella.

Todavía hay algunos casos de verdadero éxito creados de manera individual y por pequeños estudios. Por ejemplo, *Cave Story* fue desarrollado íntegramente por Daisuke Amaya (a.k.a. Pixel). Tardó 5 años en crear el juego completo y lo lanzó en 2004. Años más tarde, Nicalis hizo un remake en 3D para Nintendo 3Ds y para Switch. Aún se puede descargar su juego y acceder a los scripts, sprites y sonidos del juego. Sin embargo, no es lo más común a día de hoy y es necesario una organización y metodologías de trabajo en equipo (incluso en actividades más lúdicas como las Game Jams –desarrollos en muy pocos días con equipos de trabajo muy reducidos).

Para trabajar en grandes proyectos de desarrollo es necesario definir un proceso a seguir, especializar a los desarrolladores en distintos roles y definir una metodología que de soporte al proceso de desarrollo.

1.1. Proceso de desarrollo de videojuegos

El proceso de desarrollo de videojuegos se compone de las siguientes fases (Figura 1.2):

- **Concept:** Fase de generación de una idea de videojuego. Generalmente se realizan prototipos de concepto para validar que la idea a desarrollar es asumible por el equipo de desarrollo.
- **Preproducción:** Define qué juego vamos a desarrollar, en cuánto tiempo y con cuánta gente. Define el documento de concepto, las estimaciones de tiempos y costes e identifica los riesgos del juego. Durante este periodo también se pueden desarrollar bocetos de niveles, pruebas tecnológicas y assets.
- **Producción:** Es el proceso de desarrollo del juego en sí mismo, a partir del documento de concepto definido y siguiendo la planificación propuesta. Para ello es necesario controlar el proceso de desarrollo. Este proceso ha de ser flexible a los cambios que irán surgiendo a medida que desarrollemos el videojuego (nuevas características que aparecen durante el desarrollo, eliminación de características

que han dejado de resultar interesantes, cambios debido a reorganización de la planificación...)

- **Testing:** Es la fase de pruebas y equilibrado, donde se pulen el videojuego para su lanzamiento al mercado. En realidad, este proceso debe de ser desarrollado a lo largo de toda la producción, sobre todo en los desarrollos finales del juego. Una vez pasadas las pruebas (software y hardware) se realiza el lanzamiento del juego.
- **Postproducción:** Consiste en el cierre del desarrollo, incluyendo la creación de un kit del proyecto, junto con una revisión crítica del mismo, de modo que se pueda aprender de la experiencia. Esta revisión es lo que se conoce como *postmortem*.

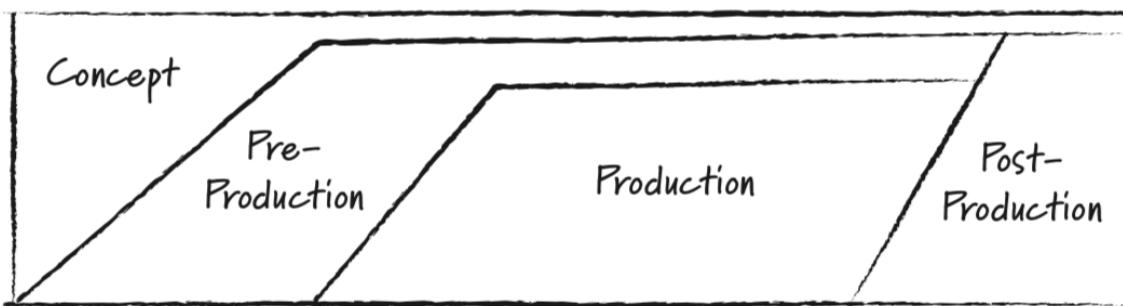


Figura 1.2: Fases del proceso de desarrollo en videojuegos (Fuente: *Agile Game Development with Scrum*)

1.2. Roles en los equipos de desarrollo

Los equipos de desarrollo de un videojuego se componen de varias personas divididas en roles o perfiles. Los más comunes son los siguientes:

- **Productores:** Negocian con agentes externos, monitorizan la planificación, gestionan el presupuesto, marketing ...
- **Arte:** Encargados de los recursos gráficos.
- **Ingeniería o programación:** Involucrados en todos los aspectos del juego, analizan el diseño para implementar las funcionalidades que hacen que el juego funcione.
- **Diseño:** Responsables de la experiencia de juego, toman decisiones de jugabilidad y, durante la producción, implementan el diseño (mapas, niveles...)
- **Audio:** Encargados de las bandas sonoras así como de los efectos de sonido y diálogos.

- QA (*Quality assurance testing*): diseñan los planes de pruebas y comprueban que el juego funciona correctamente y no tiene errores.

1.3. Metodologías de desarrollo

Las metodologías de desarrollo de software definen un conjunto de prácticas que sirven para mejorar el rendimiento de los equipos de trabajo, reducir los riesgos y obtener mejores resultados de desarrollo.

1.3.1. Metodologías basadas en procesos lineales

Las grandes empresas comenzaron a utilizar procesos de desarrollo clásicos, como el **desarrollo en cascada**, y lo adaptaron al proceso de desarrollo de videojuegos (Figura 1.3). El modelo en cascada **describe un *proceso lineal***, de modo que no se puede pasar a la siguiente fase hasta que no se ha completado la anterior. También existe la posibilidad de revisar una fase anterior. En este proceso **todo el diseño del videojuego se realizaría en las primeras fases**, mientras que las pruebas se realizan solo al final del proceso, lo que **implica que, en general, el trabajo realizado en una fase no se verifica hasta el final**. Además, se **pone un gran énfasis en la documentación del proyecto** (un GDD enorme y difícil de utilizar). Por último, las **metodologías lineales son muy poco flexibles a cambios**, algo que es muy común en el desarrollo de videojuegos.

El desarrollo de videojuegos es un proceso especialmente creativo y, como tal, necesita de lo que se conoce como *diseño iterativo*. Este proceso consiste en la realización de desarrollos cíclicos (iterativos) cortos que permiten probar y reflexionar sobre el producto conseguido en cada iteración para aprender sobre él y mejorar en la siguiente iteración. Esto se debe a un principio de diseño básico en los videojuegos:

La única forma de reconocer que un videojuego es divertido es jugando

El diseño iterativo tiene las siguientes características:

- Permanece abierto a cambios.
- Obliga a tener productos funcionales (demos) lo antes posible y de manera continua.
- Permite encontrar problemas rápidamente y ayuda a afrontarlos desde el primer momento.
- Las demos ayudan a probar y aprender.

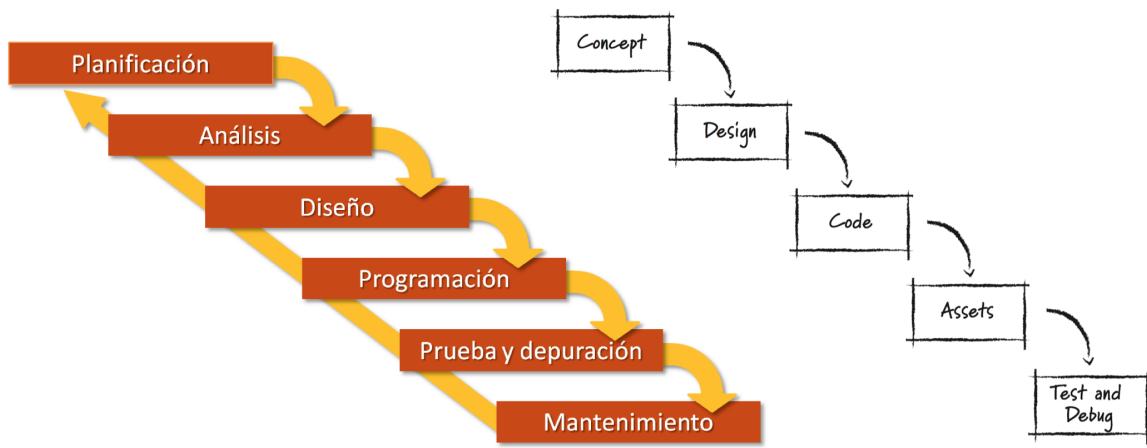


Figura 1.3: Modelo en cascada clásica (a la izquierda) y el mismo modelo adaptado a videojuegos

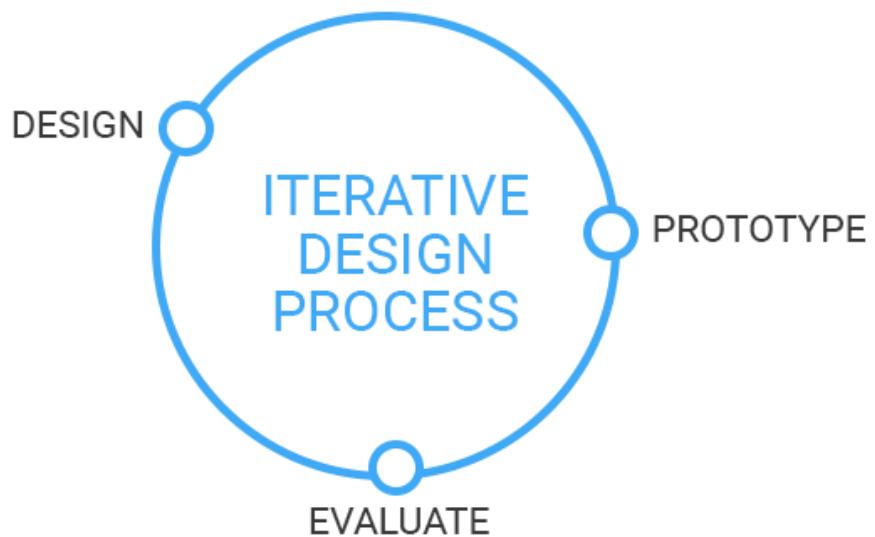


Figura 1.4: Diseño Iterativo (Fuente: JustinMind)

La filosofía de diseño iterativo es la que inspira el uso de metodologías ágiles.

1.3.2. Metodologías ágiles

Las metodologías ágiles o ligeras surgen como un medio para reducir la “burocracia” de las metodologías clásicas en proyectos de pequeña o mediana escala y con equipos de trabajos pequeños (no más de 10 personas). Además, reducen la probabilidad de fracaso por malas planificaciones y estimaciones. También se caracterizan por ser adaptativas ya que esperan que se produzcan cambios durante el desarrollo.

Las metodologías ágiles se describen mediante los valores descritos en el [Agile Manifesto](#): “Nuestra experiencia en el desarrollo de software nos ha enseñado a valorar”:

- Individuos e interacciones sobre procesos y herramientas: Las personas son el principal factor de éxito y es necesario crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.
- Software funcionando sobre documentación extensiva: no hay que producir documentos que no sean necesarios de manera inmediata para tomar decisiones importantes. Éstos han de ser cortos y centrados en lo fundamental. En su lugar, se prefiere que haya software funcionando, lo que ayuda a ver que el equipo va por buen camino.
- Colaboración con el cliente sobre negociación contractual: Es necesaria una interacción constante entre el cliente y el equipo de desarrollo.
- Respuesta ante el cambio sobre seguir un plan: hay que ser capaz de adaptarse a los cambios.

Estos valores se alinean con los [12 principios de las metodologías ágiles](#):

1. La prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Dar la bienvenida a los cambios.
3. Entregar frecuentemente software que funcione.
4. Los responsables del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.

8. Los procesos Ágiles promueven el desarrollo sostenible, manteniendo un ritmo constante de trabajo de forma indefinida.
9. La atención continua al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Las metodologías definen conjuntos de prácticas que son útiles para el desarrollo de software. Una práctica es una guía o conjunto de técnicas para definir cómo se ha de desarrollar software o cómo se ha de gestionar un proyecto.

1.4. Scrum

Scrum es una **metodología ágil con prácticas ligeras de gestión**. Se componen de **una serie de prácticas sencillas de comprender pero difíciles de dominar**. Se basa en los siguientes pilares:

- **Transparencia:** todos pueden ver cómo evoluciona el proyecto, tanto desde dentro como desde fuera del proyecto.
- **Inspección:** revisar constantemente el producto y el progreso del mismo, con el objetivo de comprobar que no nos estamos desviando de la planificación realizada.
- **Adaptación:** Las desviaciones de la planificación, así como los errores y problemas se corrigen rápidamente.

1.4.1. Proceso Scrum

El proceso comienza creando el **Product Backlog**, una **lista de las características del videojuego**. Cada **una de las características** se conoce como **Product Backlog Item (PBI)**. Generalmente **se crean a partir del documento de concepto** aunque pueden **incluir otros requisitos** (como herramientas o infraestructura). El **product backlog** está **ordenado por prioridad de los PBIs**.

A continuación **se planifica su desarrollo como una secuencia de sprints**. Cada sprint es **una iteración de desarrollo que tiene una duración fija** (generalmente, entre 1 y 4

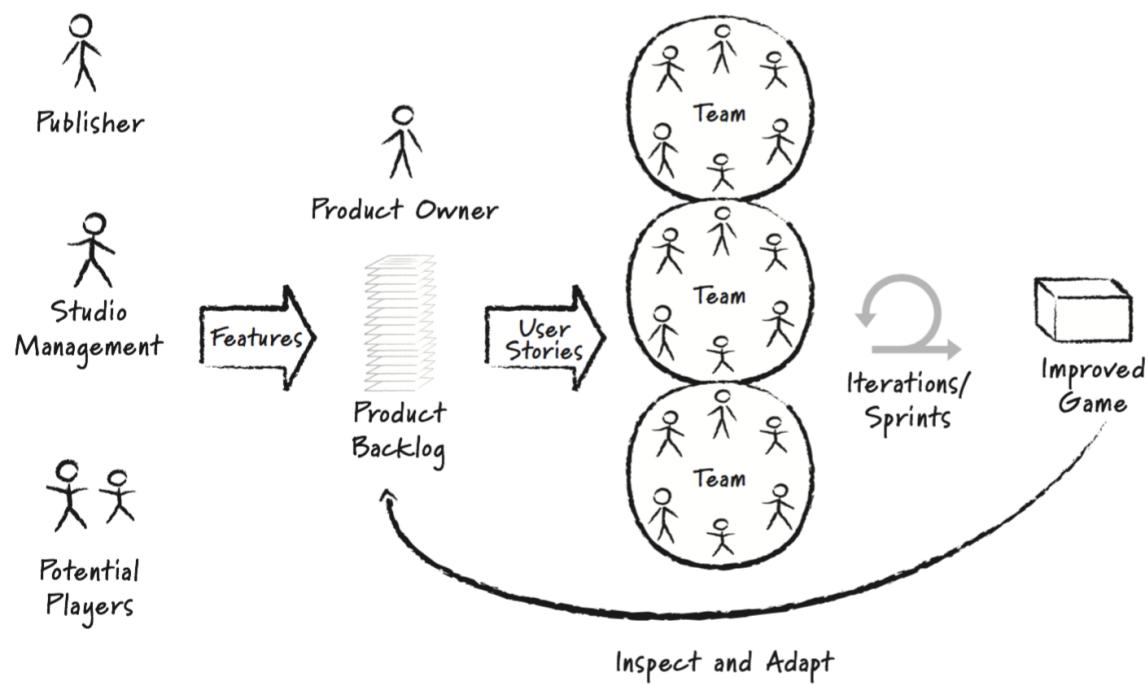


Figura 1.5: Proceso general de Scrum (Fuente: *Agile Game Development with Scrum*)

semanas).

Generalmente, a lo largo del desarrollo se realizan varias **releases**, hitos o **milestones**, de modo que se generan versiones “jugables” del producto cada poco tiempo.

A modo de ejemplo, en <https://polyghtgons.wordpress.com/page/2/> se puede ver muy claramente la evolución del juego Polyghtgons. Éste fue desarrollado siguiendo una planificación de 5 hitos con sprints semanales en el Master de Desarrollo de Videojuegos. Fue desarrollado, entre otros, por Carlos Fraguas (Mercury Steam - Raiders of the Broken Planet) y Francisco González Arribas (Halfbrick)

1.4.2. Fases de un sprint

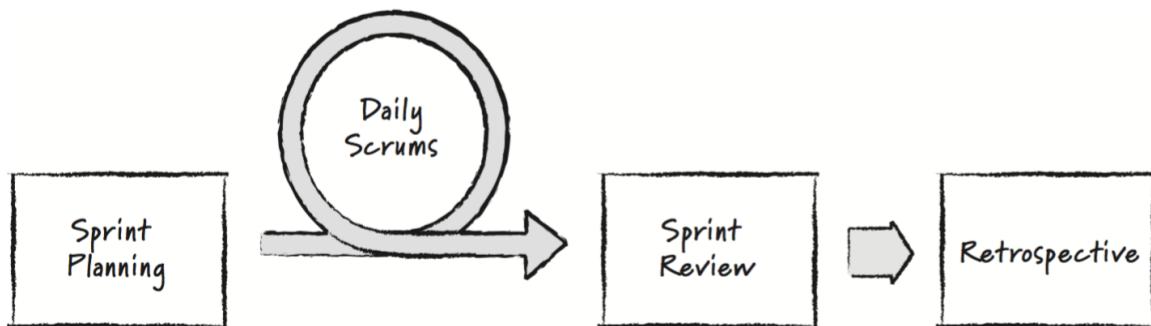


Figura 1.6: Desarrollo de un sprint (Fuente: *Agile Game Development with Scrum*)

1. Reunión de planificación del sprint (**Sprint planning meeting**): Se crea el **sprint backlog** usando el product backlog. Se discuten los PBI a incluir, se estima el coste de cada PBI, se priorizan y ordenan. Una vez definido, se dividen los PBI en tareas, si se considera necesario.
 2. Desarrollo del sprint (**sprint execution**): Es el día a día del sprint. El equipo de desarrollo va asignando PBIs a cada uno de sus miembros para que vayan siendo desarrollados y probados. Si se termina un PBI entonces se revisa el backlog y se asignan nuevas tareas. Diariamente se celebra un **daily scrum**, una reunión informal breve (10-15 min.) en el que cada persona explica:
 - ¿qué hice ayer?
 - ¿qué voy a hacer hoy?
 - ¿qué obstáculos me pueden impedir avanzar?
- Si todos los PBIs del sprint backlog se completan entonces se cierra el sprint.

3. **Sprint review:** Una vez concluido el sprint se realiza una presentación pública del proyecto en el estado en que se encuentra tras finalizar el sprint. La presentación es tanto para desarrolladores como para clientes. La presentación implica presentar una demo de software funcionando y que puede ser probada con usuarios para obtener feedback.
4. **Retrospectiva (retrospective):** Reunión en la que se hace una revisión del sprint. Es importante concluir con un detalle de:
 - ¿Qué fue bien?
 - ¿Qué fue mal / podemos mejorar?
 - ¿Qué podemos cambiar?

En general, los errores de programación o *bugs* se resuelven lo antes posible, añadiéndolos como tareas en el Sprint backlog o en el Product backlog (si se ve que tiene un impacto grande y se estima que no podrá ser resuelto en el sprint actual)

1.4.3. **Equipos de desarrollo y roles**

Los equipos de desarrollo en Scrum son pequeños (máximo 10 personas), multidisciplinares, sin subequipos y autogestionados.

Además de los roles previstos en el proceso de desarrollo de videojuegos, hay dos roles específicos de Scrum:

- **Product owner:** Persona responsable del producto y, por tanto, del product backlog. Crea los PBI, las prioriza y hace de filtro entre desarrolladores y clientes.
- **Scrum Master:** es el responsable de que el grupo se adhiera a las prácticas de Scrum (*a cop, more than a coach*), facilita los sprints y se encarga de que el equipo de desarrollo se organice y no se produzcan bloqueos. Además, ayuda al product owner a definir el product backlog y a priorizarlo.

1.4.4. **Seguimiento**

Scrum impone realizar un seguimiento continuo del desarrollo con el fin de detectar errores y resolverlos lo antes posible. Este seguimiento se realiza mediante distintas técnicas y herramientas, algunas de las cuales describimos a continuación:

1.4.4.1. Gráficos de seguimiento (*burndown charts*):

Son gráficos que miden la evolución del trabajo durante un sprint (Figura 1.7). Sirven para ver cuál es el progreso real y si nos estamos desviando de planificado. Son útiles si las estimaciones son buenas y frustrantes en caso contrario, lo que puede perjudicar al equipo de desarrollo.

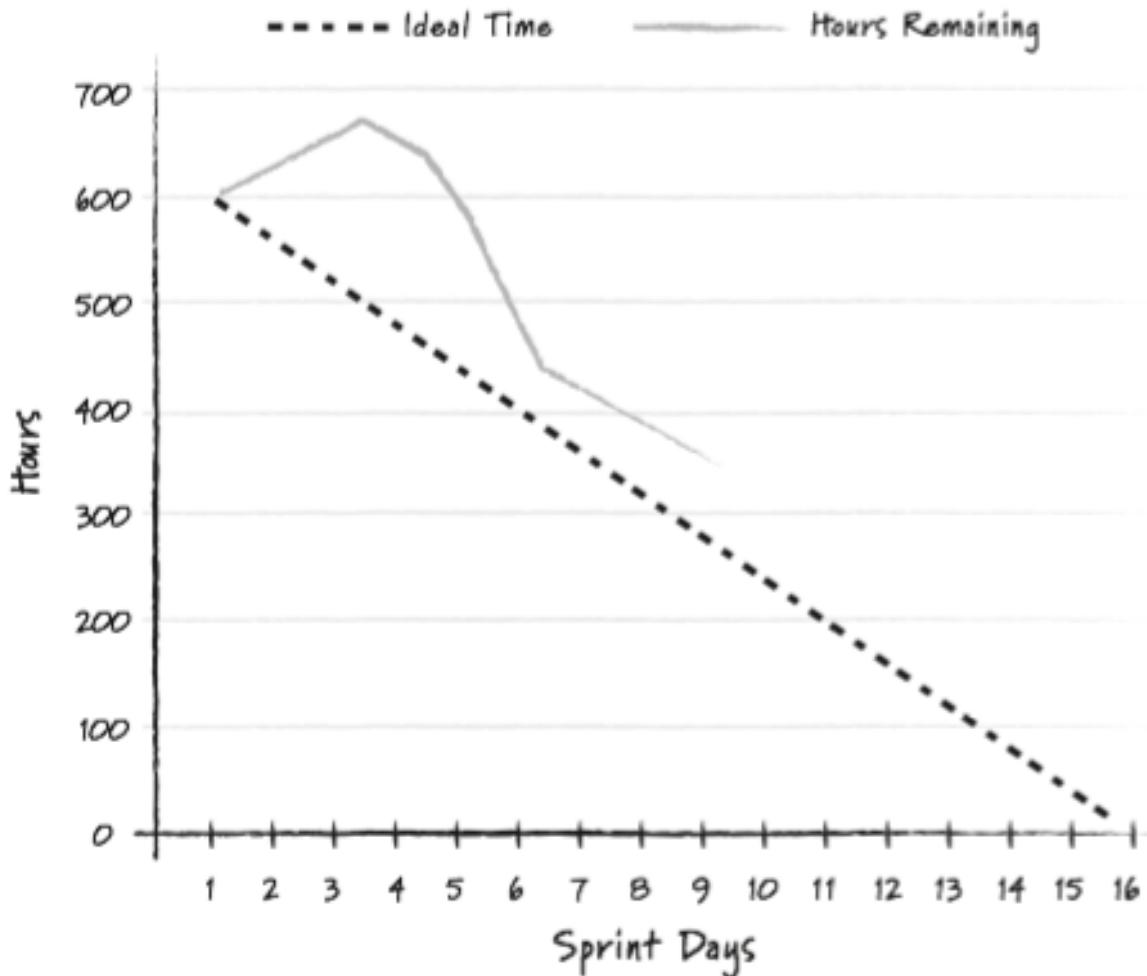


Figura 1.7: Gráficos de seguimiento (Fuente: *Agile Game Development with Scrum*)

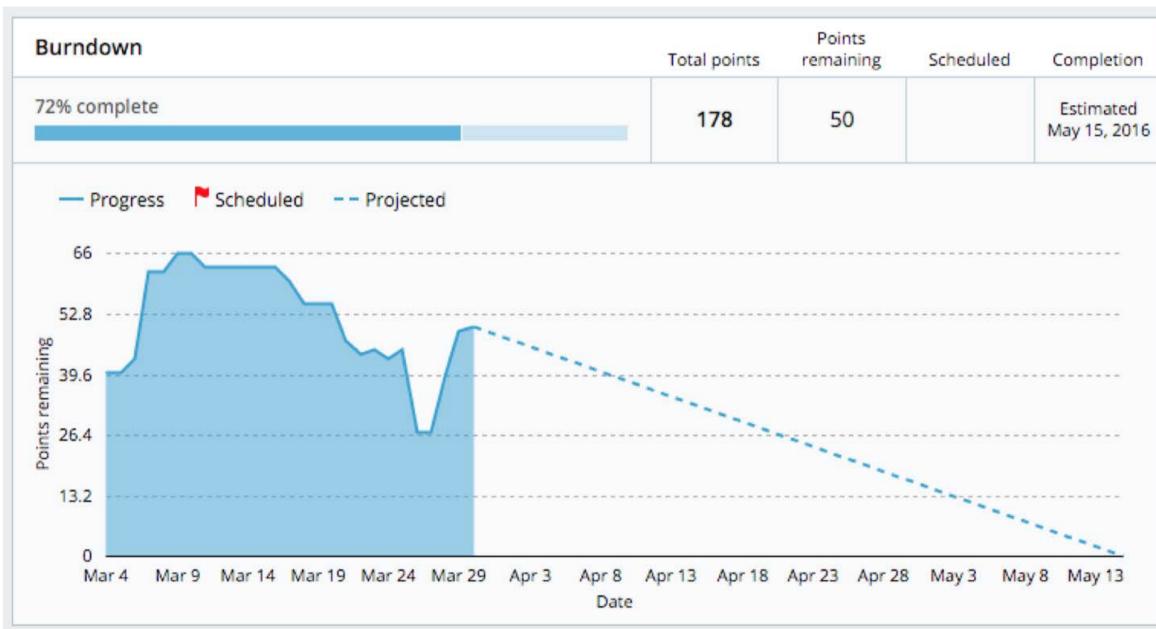


Figura 1.8: Gráficos de seguimiento en Pivotal

1.4.4.2. **Pizarra de tareas:**

Es el lugar en el que se muestra el progreso del equipo durante un sprint (Figura 1.9). En ella aparecen los PBIs del Sprint backlog, las tareas que se están haciendo y las que se han completado, generalmente acompañadas del gráfico de seguimiento.

Podemos crear pizarras “artesanales” (pizarra blanca, columnas, post-its) o utilizar herramientas como Trello o Pivotal Tracker (entre otras).

1.5. Extreme Programming

El **eXtreme Programming (XP)** es una metodología ágil dirigida especialmente a programadores. Destacan dos prácticas fundamentales:

- **Test-driven development (TDD):** Cada funcionalidad a implementar viene precedida de la creación de los test de unidad, es decir, de las pruebas que verifican su funcionalidad. No solo sirven para probar sino que también documentan. Además, la ejecución de los test está automatizada mediante lo que se conoce



Figura 1.9: Pizarra de tareas

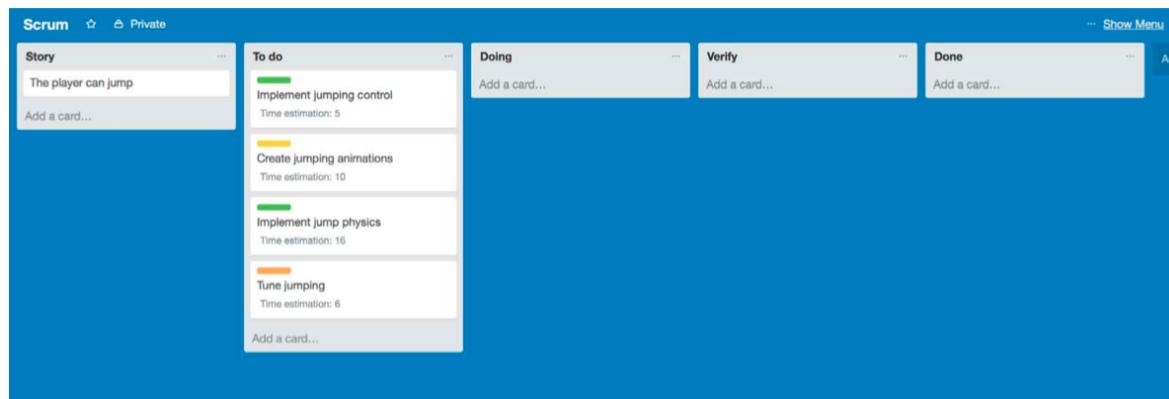


Figura 1.10: Pizarra de tareas en Trello

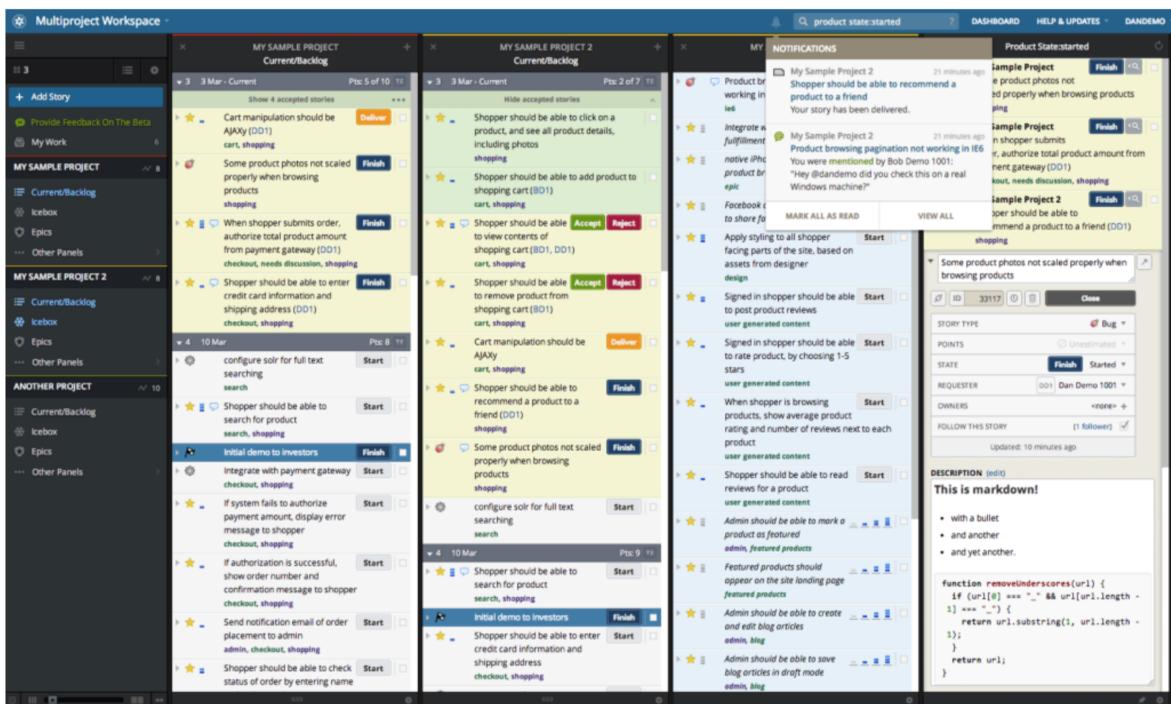


Figura 1.11: Pizarra de tareas en Pivotal

como *integración continua*. También se promueve la refactorización del código, es decir, modificar la arquitectura del mismo para que sea de mejor calidad y más sencillo de utilizar y de añadir nueva funcionalidad.

- **Pair programming:** Dos programadores sentados juntos frente a un único PC de desarrollo, de modo que uno programa mientras el otro observa y comenta lo que se está haciendo en busca de mejorar las decisiones de diseño (del programa que se está implementando). Cada cierto tiempo se invierten los roles.

1.6. Testing

El videojuego que estemos desarrollando ha de ser continuamente probado por dos motivos fundamentales:

1. Queremos que sea divertido
2. Queremos que funcione correctamente y no tenga errores

Con respecto al primer punto, no conoceremos de verdad cómo es nuestro juego hasta que un jugador (que no sea uno de los desarrolladores del juego) lo pruebe y dé *feedback*. Por tanto, es necesario *analizar cómo interactúa el jugador con nuestro juego* mediante evaluaciones con usuarios (*playtesting, focus groups, game user research...*).

Con respecto al segundo, es necesario que la búsqueda de errores en nuestro Videojuego sea *sistemática* y, en muchas ocasiones, *automatizable*, con el fin de realizarlos de la manera más rápida y repetitiva posible. De esta tarea se dedicará principalmente el departamento de **Quality Assurance (QA)**, que definirá los planes de pruebas y de validar el juego siguiendo dichos planes.

1.7. De la preproducción a la producción

Uno de los resultados de la fase de preproducción es el documento de diseño o GDD (*Game Design Document*), un documento generalmente extenso en el que se documenta y detalla de qué va el videojuego. Sin embargo, el trabajo en producción con este documento monolítico suele provocar dos problemas básicos:

- No comunica la prioridad que tiene cada uno de los requisitos / características / mecánicas de nuestro videojuego, de modo que cada miembro del equipo puede dar la prioridad que quiera a cada elemento.

- No comunica los cambios que se puedan producir en el diseño. Tendríamos que releer frecuentemente el documento para enterarnos de los cambios.

Necesitamos transformar este documento en un artefacto que:

- Permite priorizar y dar valor a cada una de las características.
- Permite añadir cambios y los comunique.
- Promueva la comunicación.
- Permite que los detalles de las características emergan al ser discutidos.
- Permite realizar refinamientos sucesivos de las características.

Por este motivo, el GDD se trocea y se convierte en los PBI que pueblan el Product Backlog. Estos PBIs se van a describir en forma de **historias de usuario**. Una historia de usuario es una descripción corta de una característica del juego, del pipeline o de una herramienta de desarrollo tiene un cierto valor **para un usuario**. No dan detalles completos de diseño sino que son una herramienta que fomenta la comunicación y que sirve para recordar cuáles son los requisitos del juego desde el punto de vista de los usuario, no de los desarrolladores.

Una historia de usuarios suele seguir la siguiente plantilla:

Como <rol de usuario>, quiero que <objetivo> [porque <razones>]

- El rol de usuario suele representar un “cliente” del videojuego o un usuario del pipeline de desarrollo que se beneficia de la historia.
- El objetivo describe brevemente la funcionalidad o característica del videojuego, de la herramienta o del pipeline.
- Las razones representan el beneficio que se tendrá de tener esta funcionalidad. Está entre corchetes porque es opcional.

Ejemplos:

Como jugador quiero ver mi nivel de salud

Como unidad enemiga, quiero poder robar objetos al jugador

Como jugador quiero ver animales corriendo por el mundo porque quiero que parezca más realista y vivo

Como ingeniero quiero poder poner minas

Como animador quiero poder cambiar las animaciones directamente en el juego sin tener que reiniciar porque de esta forma podré interaccionar más rápidamente con las animaciones.

Como se puede ver, los roles pueden ser el jugador como usuario final o uno de los roles en el contexto del juego. Así mismo, un rol de una historia de usuario también puede ser uno de los roles del equipo de desarrollo.

Una historia también deberá de tener unas pruebas que determinan cuándo está completa (también llamadas condiciones de éxito, *condition of success* o *CoS*). Las historias serán ordenadas por prioridad de cara al desarrollo del juego, teniendo en cuenta su coste, sus riesgos y dificultades.

En cada sprint se debería completar una o varias historias de usuario. Una historia de usuario demasiado grande puede ser dividida en historias más pequeñas, ordenadas por prioridad. Finalmente, en cada sprint, las historias de usuario se dividen en tareas realizadas por cada uno de los miembros individuales del equipo de desarrollo.

Nota final

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional

