Teoría del color

Se conoce como Teoría del color a un conjunto de reglas básicas que rigen la mezcla de colores para conseguir efectos deseados, mediante la combinación de colores o pigmentos. Es un principio de gran importancia en el diseño gráfico, la pintura, la fotografía, la imprenta y la televisión, entre otras áreas visuales.

- Saturación. Básicamente se refiere a la pureza del color, o sea, la concentración de gris presente en un color en un momento determinado. Mientras más gris posea, menos puro será y menor será su saturación, viéndose como si estuviera sucio, opaco.

El modelo de color RGB

El modelo de color RGB se llama así debido a sus colores primarios: rojo, verde y azul (*Red*, *Green*, *Blue*, en inglés), a partir de los cuales se compone el resto. Es un sistema de color aditivo, en el que los colores deben sumarse para producir uno nuevo.

Las excepciones son el negro, que se produce en ausencia de luz (y por ende, de color) y el blanco que se produce en presencia de todos los colores, recomponiendo el espectro. Este sistema es el empleado en la mayoría de los televisores, monitores de computador, proyectores de video, etc.

El modelo de color CMYK

El modelo CMYK es distinto al anterior, pero su nombre también es la unión de las iniciales de los colores que toma como referencia: cyan, magenta, amarillo (en inglés: *yellow*), con el añadido del negro (llamado *Key* en inglés para evitar la confusión con la *B* del *blue* del RGB).

Este modelo comprende el color a partir de la absorción de la luz, de modo que a diferencia del RGB, es de tipo sustractivo, de resta de luz: la mezcla de todos los colores puros (azul, rojo, amarillo) da negro, la ausencia total de luz.

Aparte, los diversos colores secundarios pueden formarse de esta matriz, variando las combinaciones posibles de los tres: cian y magenta construye púrpura, cian y amarillo construye el verde, amarillo y magenta construye el rojo.

Este modelo del color es empleado en las diversas técnicas de impresión en tinta, ya que el papel carece de las propiedades lumínicas de los monitores o proyectores. Por esta razón, cuando se trabaja en un programa digital de diseño, se debe convertir el RGB a CMYK a la hora de preparar el diseño para impresión.

Fases de desarrollo de software

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con grandes posibilidades de éxito. Esta sistematización

indica cómo se divide un proyecto en módulos más pequeños para normalizar cómo se administra el mismo.

Así, una metodología para el desarrollo de software son los procesos a seguir sistemáticamente para idear, implementar y mantener un producto de software desde que surge la necesidad del producto hasta que se cumple el objetivo por el cual fue creado.

De esta forma, las etapas del desarrollo de software son las siguientes:

Planificación

Antes de empezar un proyecto de desarrollo de un sistema de información, es necesario hacer ciertas tareas que influirán decisivamente en el éxito del mismo. Dichas tareas son conocidas como el *fuzzy front-end* del proyecto, puesto que no están sujetas a plazos.

Algunas de las tareas de esta fase incluyen actividades como la determinación del ámbito del proyecto, la realización de un estudio de viabilidad, el análisis de los riesgos asociados, la estimación del coste del proyecto, su planificación temporal y la asignación de recursos a las diferentes etapas del proyecto.

Análisis

Por supuesto, hay que averiguar qué es exactamente lo que tiene que hacer el software. Por eso, la etapa de análisis en el ciclo de vida del software corresponde al proceso a través del cual se intenta descubrir qué es lo que realmente se necesita y se llega a una comprensión adecuada de los requerimientos del sistema (las características que el sistema debe poseer).

Diseño

En esta fase se estudian posibles opciones de implementación para el software que hay que construir, así como decidir la estructura general del mismo. El diseño es una etapa compleja y su proceso debe realizarse de manera iterativa.

Es posible que la solución inicial no sea la más adecuada, por lo que en tal caso hay que refinarla. No obstante, hay catálogos de patrones de diseño muy útiles que recogen errores que otros han cometido para no caer en la misma trampa.

Implementación

En esta fase hay que elegir las herramientas adecuadas, un entorno de desarrollo que facilite el trabajo y un lenguaje de programación apropiado para el tipo de software a construir. Esta elección dependerá tanto de las decisiones de diseño tomadas como del entorno en el que el software deba funcionar.

Al programar, hay que intentar que el código no sea indescifrable siguiendo distintas pautas como las siguientes:

- Evitar bloques de control no estructurados.
- Identificar correctamente las variables y su alcance.
- Elegir algoritmos y estructuras de datos adecuadas para el problema.
- Mantener la lógica de la aplicación lo más sencilla posible.

- Documentar y comentar adecuadamente el código de los programas.
- Facilitar la interpretación visual del código utilizando reglas de formato de código previamente consensuadas en el equipo de desarrollo.

También hay que tener en cuenta la adquisición de recursos necesarios para que el software funcione, además de desarrollar casos de prueba para comprobar el funcionamiento del mismo según se vaya programando.

Pruebas

Como errar es humano, la fase de pruebas del ciclo de vida del software busca detectar los fallos cometidos en las etapas anteriores para corregirlos. Por supuesto, lo ideal es hacerlo antes de que el usuario final se los encuentre. Se dice que una prueba es un éxito si se detecta algún error.

Instalación o despliegue

La siguiente fase es poner el software en funcionamiento, por lo que hay que planificar el entorno teniendo en cuenta las dependencias existentes entre los diferentes componentes del mismo.

Es posible que haya componentes que funcionen correctamente por separado, pero que al combinarlos provoquen problemas. Por ello, hay que usar combinaciones conocidas que no causen problemas de compatibilidad.

Uso y mantenimiento

Esta es una de las fases más importantes del ciclo de vida de desarrollo del software. Puesto que el software ni se rompe ni se desgasta con el uso, su mantenimiento incluye tres puntos diferenciados:

- Eliminar los defectos detectados durante su vida útil (mantenimiento correctivo).
- Adaptarlo a nuevas necesidades (mantenimiento adaptativo).
- Añadirle nuevas funcionalidades (mantenimiento perfectivo).

Aunque suene contradictorio, cuanto mejor es el software más tiempo hay que invertir en su mantenimiento. La principal razón es que se usará más (incluso de formas que no se habían previsto) y, por ende, habrá más propuestas de mejoras.

Herramientas de desarrollo de software

La plataforma de desarrollo de software líder en el mundo. GitHub es un servicio que permite almacenar repositorios Git en la nube. En otras palabras, es como un Google Drive para proyectos de software donde puedes encontrar código. Cuando subes tu proyecto puedes elegir entre hacerlo público o privado. La comunidad es enorme y la base de proyectos es aún más grande. GitHub también es un sitio excelente para el desarrollo colaborativo, en el que desarrolladores pueden descubrir, compartir y construir el mejor software.

2. Git

Git es un sistema de control de versiones libre y de código abierto diseñado para gestionar todo tipo de proyectos, desde pequeños hasta muy grandes, con rapidez y eficiencia. Cuando los desarrolladores trabajan en algo, a menudo tienen que hacer cambios en el código hasta que llegan a la última versión.

Lo que hace un sistema de control de versiones es guardar cada cambio realizado, permitiendo que otros puedan colaborar, hacer cambios y contribuir. También encontrarás una copia del código del trabajo de cada desarrollador.

3. GitLab

GitLab es el producto integrado líder para el desarrollo moderno de software. Une la gestión de problemas, el control de versiones, la revisión de código, CI, CD y la supervisión en una única aplicación de fácil instalación que ayuda a los equipos a moverse más rápidamente de la fase de planificación a la de supervisión.

4. IntelliJ IDEA

Intellij es un entorno de desarrollo integrado. Un IDE (por sus siglas en inglés) integra todas las herramientas que necesitas para desarrollar una plataforma, de modo que éste cuenta con un editor de código, un compilador, un depurador, etc. Después de indexar tu código fuente, IntelliJ IDEA te ofrece una experiencia rápida e inteligente con sugerencias relevantes en cada contexto: finalización de código inteligente y al instante, análisis de código y herramientas de refactorización fiables.

5. Stack Overflow

Stack Overflow es la comunidad online de programadores más grande del mundo. Cada mes, la visitan más de 50 millones de desarrolladores. Este es el lugar donde aprenden, comparten conocimientos e impulsan su carrera. A nuestros desarrolladores les encanta este foro y lo visitan casi cada día porque en él se puede encontrar la respuesta a casi todas las preguntas y dudas que tengas.

6. Docker

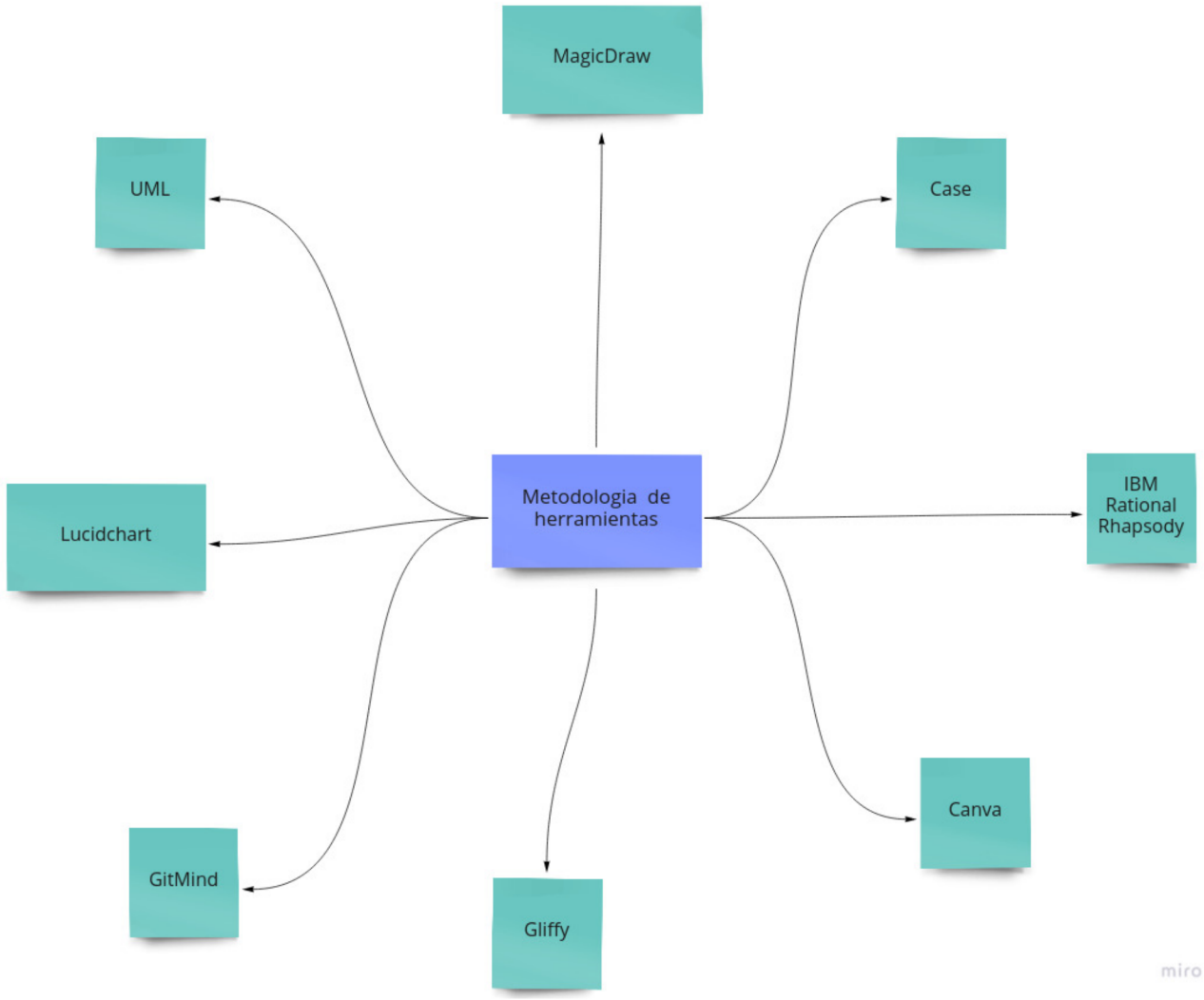
Quizás no es 100% correcto decir que Docker es una herramienta, ya que más bien se trata de un contenedor, pero creemos que no podía faltar en esta lista. Docker es una plataforma abierta para desarrolladores y administradores de sistemas para construir, enviar y utilizar aplicaciones distribuidas. Lo que hace Docker es proporcionar una plataforma de ejecución de contenedores de software que te permite contener tu software en un sistema de archivos.

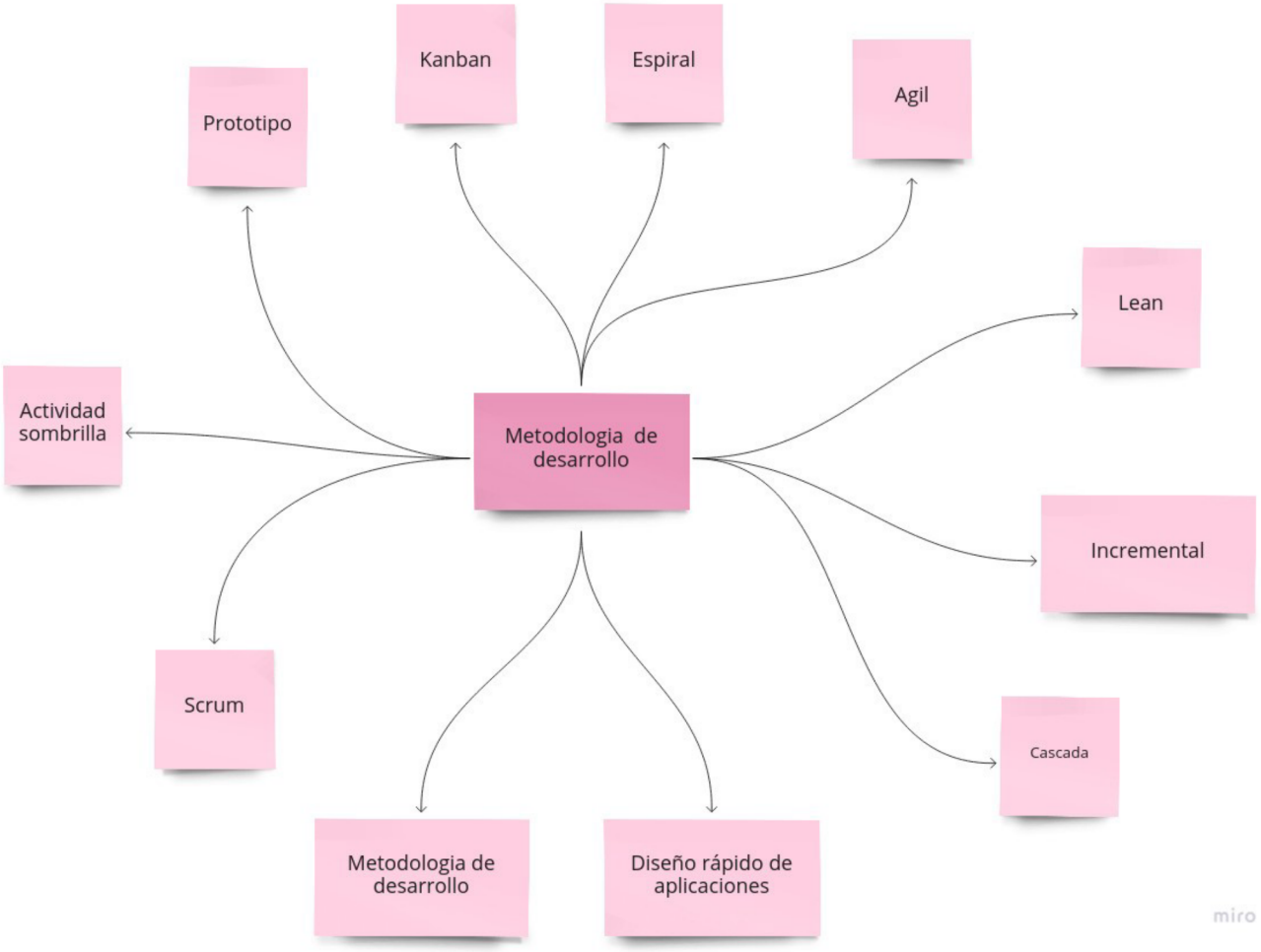
7. Jira

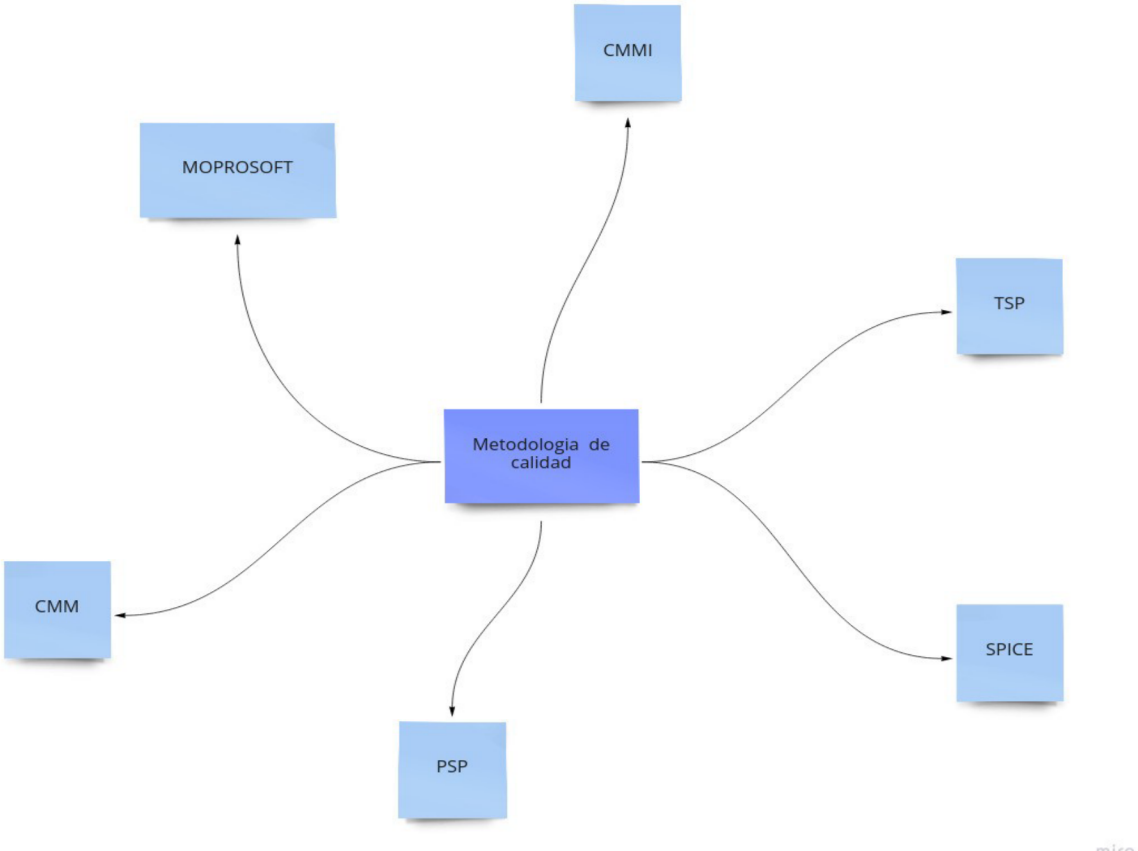
Jira es la herramienta de desarrollo de software número uno entre los equipos ágiles. Ayuda a planear, supervisar y gestionar proyectos de desarrollo de software. Además, ayuda a personalizar el flujo de trabajo, colaborar y lanzar buen software. Es de gran ayuda en la gestión de proyectos. Te permite gestionar el trabajo pendiente del equipo, visualizar el trabajo en curso y generar informes.

8. Jenkins

Jenkins es un servidor de automatización de código abierto, en concreto, un servidor de integración continua.




Kanban
Espiral
Prototipo
Agil
Lean
Actividad sombrilla
Metodologia de desarrollo
Incremental
Scrum
Cascada
Metodologia de desarrollo
Diseño rápido de aplicaciones
miro


CMMI
MOPROSOFT
TSP
Metodologia de calidad
CMM
SPICE
PSP
miro