Arquitectura de proyectos Drupal

Ramon Vilar Gavaldà

QUIÉN SOY

Ramon Vilar Gavaldà

http://ymbra.com/blogs/ramon

http://twitter.com/rvilar

http://drupal.org/user/293298

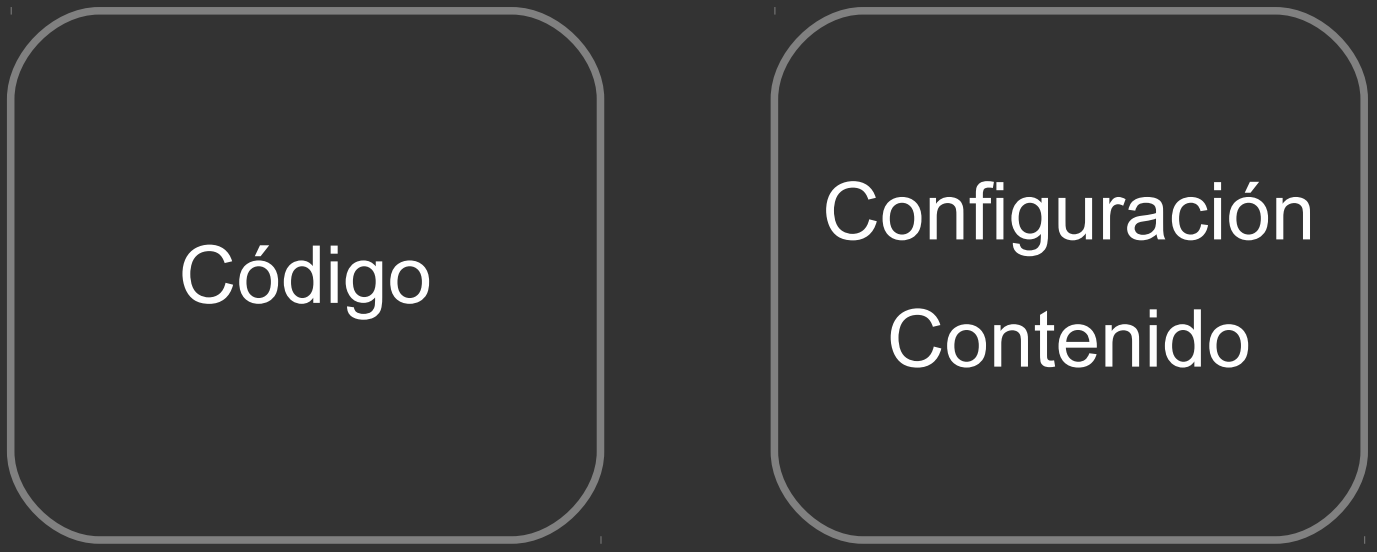
- Socio fundador de Ymbra
- Desarrollador Drupal
- Miembro activo de la comunidad drupalera:
  - Presidente de Drupal.cat
  - Administrador de la traducción catalana
  - Eventos, parches...

ÍNDICE

DESARROLLO TRADICIONAL EN DRUPAL

DESARROLLO TRADICIONAL EN DRUPAL (I)

- Aunque estemos enamorados de él, debemos aceptarlo: Drupal, *a día de hoy*, tiene un problema!



DESARROLLO TRADICIONAL EN DRUPAL (II)

- Cómo nos lo hacemos para hacer un proyecto en grupo?
- Cómo podemos mantener el proyecto versionado en un sistema de control de versiones?
- Cómo hacemos los pasos entre entornos? Y el paso a producción?
- Cómo la gente, y los proyectos, podían sobrevivir hasta ahora?

DESARROLLO TRADICIONAL EN DRUPAL (III)

- Posibles soluciones:
  - El tradicional papel y lápiz... grrrr!!
  - Algo un poco menos caótico: desarrollo de módulo
    - Hacer un módulo que cree su tipo de contenido, con sus campos, sus características, sus funciones de actualización,...
    - Ah, y exportemos sus vistas...
    - Ah, y cree sus taxonomías y sus menús...
    - Y si tiene más chicha, pues también se la ponemos...
    - Ufff!

DESARROLLO TRADICIONAL EN DRUPAL (y IV)

- Seamos claros: el lápiz y el papel no se puede considerar herramienta tecnológica.
- Miremos qué herramientas nos ofrece la comunidad y cuáles se están convirtiendo en un estándar *de facto*.
- Si no era así, a partir de ahora, el horizonte de Drupal se os va a colocar un poco más lejos.

HERRAMIENTAS PARA DESARROLLO PROFESIONAL

HERRAMIENTAS PARA DESARROLLO PROFESIONAL

- Cosas que comentaremos aquí:
  - Drush
  - Features
  - Profiles
  - Drush.make
  - Git

DRUSH:
DRUPAL SHELL

DRUSH

- Esto va a ser rápido... Drush es OBLIGADO! Y punto!
- Para más información:
  - http://drupal.org/project/drush
  - Drush User's Guide http://ves.cat/boPI

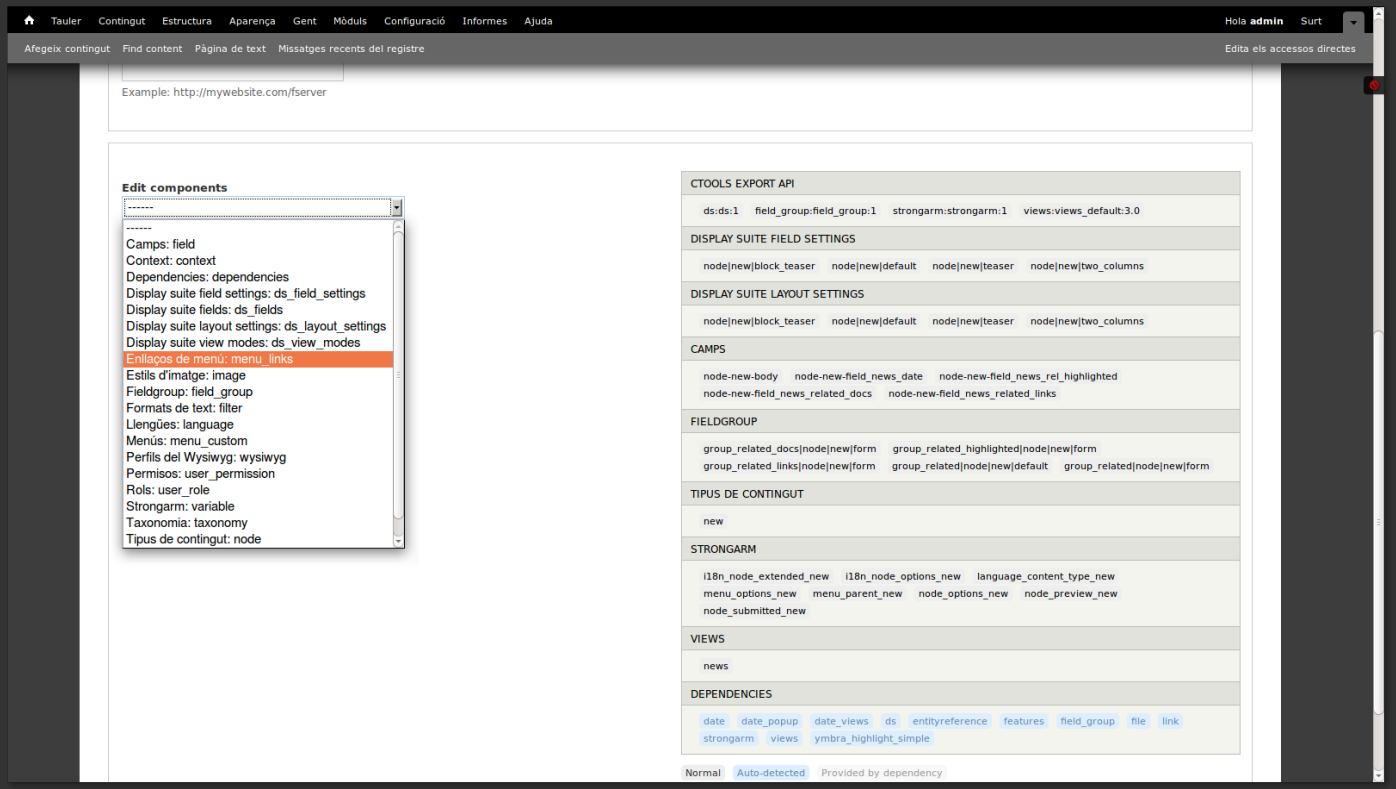
FEATURES:
TODO A CÓDIGO

FEATURES: TODO A CÓDIGO (I)

- Lo que tenemos claro es que tenemos que pasar toda la configuración y sus definiciones a código.
- Y qué ganamos?
  - Podemos versionar el código
  - Podemos resolver los conflictos (trabajo en grupo)
  - Separamos contenido de configuración
  - Facilitamos el paso entre entornos
- Módulos obligatorios: **Strongarm** y **Diff**

FEATURES: TODO A CÓDIGO (II)

- Crear un feature no tiene secreto: dispone de una UI muy intuitiva



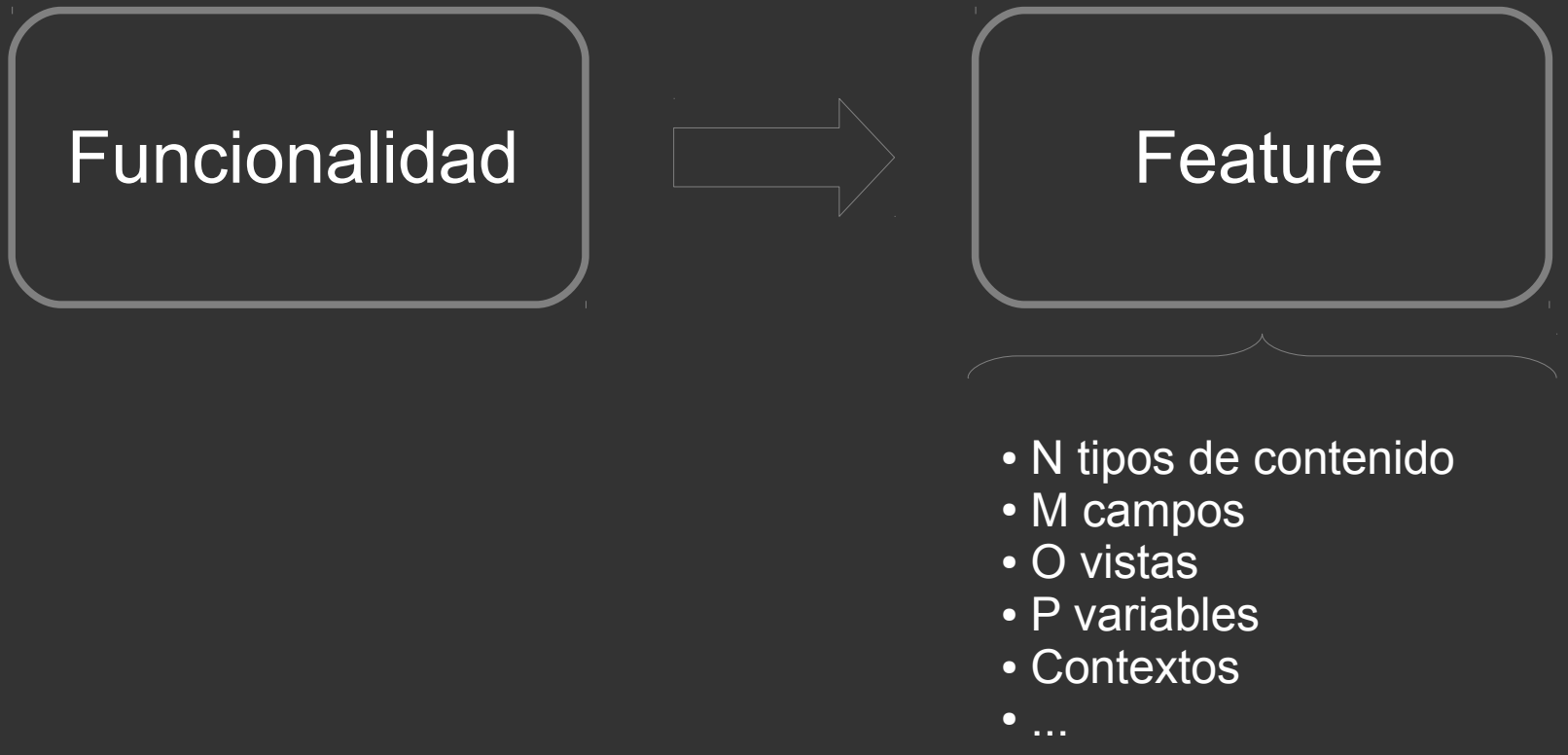
- Y también de integración con `drush`

FEATURES: TODO A CÓDIGO (III)

- Antes de empezar, hagamos un pequeño paso para el programador, pero un gran paso para el mantenedor: organicemos los directorios!
  - `/contrib`
  - `/custom`
  - `/features`

FEATURES: TODO A CÓDIGO (IV)

- Al empezar un proyecto, tenemos que tener claro qué funcionalidades tendrá



FEATURES: TODO A CÓDIGO (V)

- Un feature lo podemos hacer tan genérico cómo queramos y luego crear otros que lo complementen.
- Por ejemplo, podemos crear un feature que sea una noticia básica y luego crear otro feature que tenga cómo dependencia este y sólo le añada, por ejemplo, la capacidad de tener comentarios.
- No tengamos miedo en hacer features pequeños y jugar con las dependencias... pero no nos pasemos!

FEATURES: TODO A CÓDIGO (VI)

- Es normal tener algún feature que no tenga ningún tipo de funcionalidad, sino que sólo nos sirva para exportar configuración y/o parametrización del sistema
- Es el llamado **feature_sitewide** o **sitewide_config**
- Este nos puede servir, por ejemplo, para exportar nuestros formatos de entrada, menús,...
- Otro concepto es el **Controller Feature**: *"One feature to rule them all"* (Nuvole)

FEATURES: TODO A CÓDIGO (VII)

- Funcionalidad = Feature = ... = Módulo? Por qué no?
- Normalmente cuando queremos encapsular una funcionalidad, no sólo queremos encapsular su configuración, sino también algún comportamiento JS asociado, estilos, plantillas, etc.
- Cuando creamos un feature nos crea un archivo llamado `feature-name.module`
- Ese fichero es de libre modificación (sólo debemos respetar el `include`)
- Podemos crear unidades de desarrollo reutilizables en otros proyectos a partir de nuestros features
- Somos libres de implementar nuestros `hooks`

FEATURES: TODO A CÓDIGO (y VIII)

- Y hasta podemos añadir su propia plantilla `node-slideshow.tpl.php` en el módulo
- Sólo tenemos que añadir nuestro módulo al *theme registry* de Drupal: http://ves.cat/bazN
- Y con todo esto conseguimos tener módulos reutilizables para otros proyectos.
- Mola, no?
- Para más información, presentación de DrupalDay 2012 http://ves.cat/boTD

PROFILES: UNA HERRAMIENTA DE DESARROLLO

PROFILES: UNA HERRAMIENTA DE DESARROLLO (I)

- En Drupal 7 los profiles pasan a ser “módulos” con esteroides.
- Si son módulos, podemos añadirles funciones y hooks, sin problemas, aprovechando la potencia que esto nos permite
- Por qué no aprovechar esto y utilizar los profiles para guiar nuestros desarrollos?

PROFILES: UNA HERRAMIENTA DE DESARROLLO (y II)

- Un proyecto = un profile
- En el fichero `.info` definimos los módulos (y features) que se deben activar para nuestro proyecto
- Usar profiles facilita el despliegue en entornos y la posibilidad de integrar con un sistema de integración continua
- Podemos usar funciones `hook_upate()` para, por ejemplo, automatizar tareas en actualizaciones del proyecto: habilitar/deshabilitar módulos, etc.

DRUSH MAKE: EL ÍNDICE DE NUESTRO PROYECTO

DRUSH MAKE: EL ÍNDICE (I)

- Un desarrollo Drupal no consiste sólo en descargar módulos, activarlos y usarlos.
- Es común usar versiones en “desarrollo” (vía commit de git por favor!), además de aplicar parches en estos...
- Y además usar también temas contribuidos cómo base...
- Y además, usar librerías que complementan algunos módulos.
- Cómo saber de qué está formado tu proyecto al cabo de un tiempo?

DRUSH MAKE: EL ÍNDICE (II)

```
; Drush Make file
core = 7.x
api = 2
projects[drupal][type] = core

; MODULE
Sprojects[entityreference][version] = 1.0-rc5
projects[entityreference][subdir] = contrib

projects[i18nviews][subdir] = contrib
projects[i18nviews][download][type] = git
projects[i18nviews][download][url] = http://git.drupal.org/project/i18nviews.git
projects[i18nviews][download][revision] = 059e772ae25e925c33c0697bf37241a1e41b1a16

projects[l10n_update][version] = 1.0-beta3
projects[l10n_update][subdir] = contrib

projects[menu_block][version] = 2.3
projects[menu_block][subdir] = contrib
projects[menu_block][patch][1461254] = http://drupal.org/files/menu-block-language-1461254-1.patch

; THEMES
projects[omega][version] = 3.1
projects[omega][subdir] = contrib

; LIBRARIES
libraries[jquery.cycle][download][type] = file
libraries[jquery.cycle][download][url] = http://malsup.github.com/jquery.cycle.all.js
libraries[jquery.cycle][destination] = libraries
```

DRUSH MAKE: EL ÍNDICE (III)

- Si seguimos este enfoque, un proyecto está formado por:
  - 1 perfil de instalación
  - 1-N ficheros make con la definición de los módulos, librerías, etc del proyecto
  - Una carpeta `/modules/features` con las funcionalidades y la configuración del proyecto
  - Una carpeta `/modules/custom` con los módulos a medida
  - Una carpeta `/themes/custom` con los temas a medida

DRUSH MAKE: EL ÍNDICE (y IV)

- Si queremos instalar un nuevo módulo, lo añadimos al fichero make y ya está...¿?
- Debemos volver a ejecutar el makefile! No hay problema...

```
#!/bin/bash
rm -rf modules/contrib
rm -rf themes/contrib
drush make --working-copy --no-core
  --contrib-destination=. profile.make .
drush updatedb -y && drush cc all
```

- Todo esto se puede automatizar vía CI

GIT:
CONTROL, CONTROL Y MÁS CONTROL

GIT: CONTROL, CONTROL Y MÁS CONTROL (I)

- Utilizar un CVS se ha convertido en un imprescindible hasta para proyectos de una sola persona
- Git, cómo ya sabéis, es el CVS que se usa para mantener/gestionar el código de Drupal y sus módulos
- Si no conocéis Git, os animo/obligo a que vayáis a la sesión de juampy: Git y Drupal http://ves.cat/boTF

GIT: CONTROL, CONTROL Y MÁS CONTROL (II)

- Con lo aquí propuesto, en Git sólo tenemos la carpeta del perfil del proyecto.
- Está está formada por:
  - Archivos própios del perfil
  - Un archivo <profile>.make
  - Los features del proyecto
  - Los módulos personalizados
  - El tema

GIT: CONTROL, CONTROL Y MÁS CONTROL (III)

- El .gitignore del perfil es:

```
modules/*
!modules/custom
!modules/features
themes/*
!themes/custom
libraries/*
```

- Qué sentido tendría tener los módulos o temas contribuidos o las librerías en Git si ya está la información en d.o?

GIT: CONTROL, CONTROL Y MÁS CONTROL (y IV)

- Recomendaciones de un programador que ha sufrido...
  - Haz commits de cada unidad significativa. Si debes volver a atrás te será más fácil y menos impactante.
  - Usa/abusa de las ramas. Si están es por algo!
  - Tagea los estados en Git. Tarde o temprano vas a querer saber cuál era el estado del código el día que subiste a producción algo
  - Y mucho más!

LAST BUT NOT LEAST

PASO ENTRE ENTORNOS (I)

- Quién de aquí ha hecho alguna vez un paso entre entornos con lápiz y papel?
- Venga no engañéis...
- Venga...
- ...
- Lo acepto, yo también lo he hecho!
- Pero hoy en día ya “no” hay excusa...

PASO ENTRE ENTORNOS (y II)

- Si tenemos toda nuestra configuración en código, un paso entre entornos se puede volver en una cosa “trivial”
- “Simplemente” haciendo un `git pull` (de la rama que queramos) `+ drush updb + drush cc all` podemos desplegar en otro entorno
- Usarlo, veréis cómo hay un antes y un después y dormiréis más a gusto!

INTEGRACIÓN CONTINUA

- Si tenemos todo a código, y además en Git, ya no tenemos escusas para no usar un sistema de integración continua
- Hacer un sistema de despliegue automático es mucho más fácil con esta arquitectura
- Se nos terminan las escusas para no empezar a usar Jenkins, por ejemplo
- Hay una sesión por ahí que trata estas palabrotas http://ves.cat/boTG

CONTACTO

- Twitter: @rvilar
- Correo: ramon@ymbra.com
- Blog: http://ymbra.com/blogs/ramon
- Web: http://ymbra.com

Grácias a todos(as). ¿Preguntas?