Rethinking Namespaces

PSR-4, follows Directory, Autoloader
Autoloading ist mehr als Mapping Klasse zu Datei
Namespace könnte auch Package heißen,

namespace Vendor\Package\Subfolder

PSR-0 kam 2010 - mittlerweile deprecated
PSR-4 kam 2013 - Entry-Point statt komplettes Namespace-Directory-Mapping

Nachteil: NS diktiert durch das Autoloading
Vorteil: sieht bei gleichen Fw immer gleich aus

NS bildet meist technische Infrastruktur nach: Laravel Init, Symfony Demo, Rails App
erzählt: "What is made of?", aber nicht "What it does?"
Wo ist die Domain-Logik, NS unterstützt nur das Framework Catering

Package vs. NS
gruppiert eigentlich Dinge, NS bilden aber eher technische Details ab
import vendor.package.* funktioniert in PHP nicht, müsste alles laden um die Klassen zu kennen
vendor.package.subpackage -> Teileinheit eines Paketes, muss nicht exportiert werden, wird vom Paket verwendet
private class gibts in Java, ist außerhalb vom Package nicht sichbar - gibts in PHP nicht

Domain Driven Dev/Design
Code kann man Fachler zeigen, dieser kann sich aus Methoden grob herauslesen was die Klasse tut, weil die Begrifflichkeiten bekannt sind

namespace Vendor\Pack\MyDomain\Feature;
use Vendor\Pack\MyDomain\Feature\View
use Vendor\Pack\MyDomain\Feature\Repository
class Controller { .. }

Composer kam 2012
auf Production immer mit dump-autload mit --optimize --class-authorative

Composer kann auch Classmap -> scannt ein Verz. und merkt sich die Datei->Klasse(+NS) in einer Cache-Datei
phpab: A/B Tests
interessant wg. Classmap
snd aber statisch, müssen nach Änderungen neu generiert werden
kann nervig sein/werden, aber man nach NS von Verzeichnisstruktur trennen

dePHPend checkt Namespace Dependencys

Beispiel siehe Handyfoto
lowercase directories - weil es BS gibt die das nicht unterscheiden
alles gleicher Namespace "vendor\checkout".
Vorteil: einfach Verzeichnisstruktur ändern
benötigt eindeutige Klassen, müssen aber nicht "*Command" o. "*Event" o.ä. heißen