imagine 2012

Magento® Conference

Hail or Fail: The Right Way™ to Override Core

Mark Shust
eCommerce Developer
Metrics Marketing

imagine 2012
Magento Conference

About Me

- Over 10 years of experience working with custom and open source applications, including Drupal, osCommerce, and Magento
- eCommerce Developer at Metrics Marketing Group located in Cleveland, Ohio
- Zend Certified Engineer (PHP5)
- Magento Certified Developer Plus

About Metrics Marketing Group

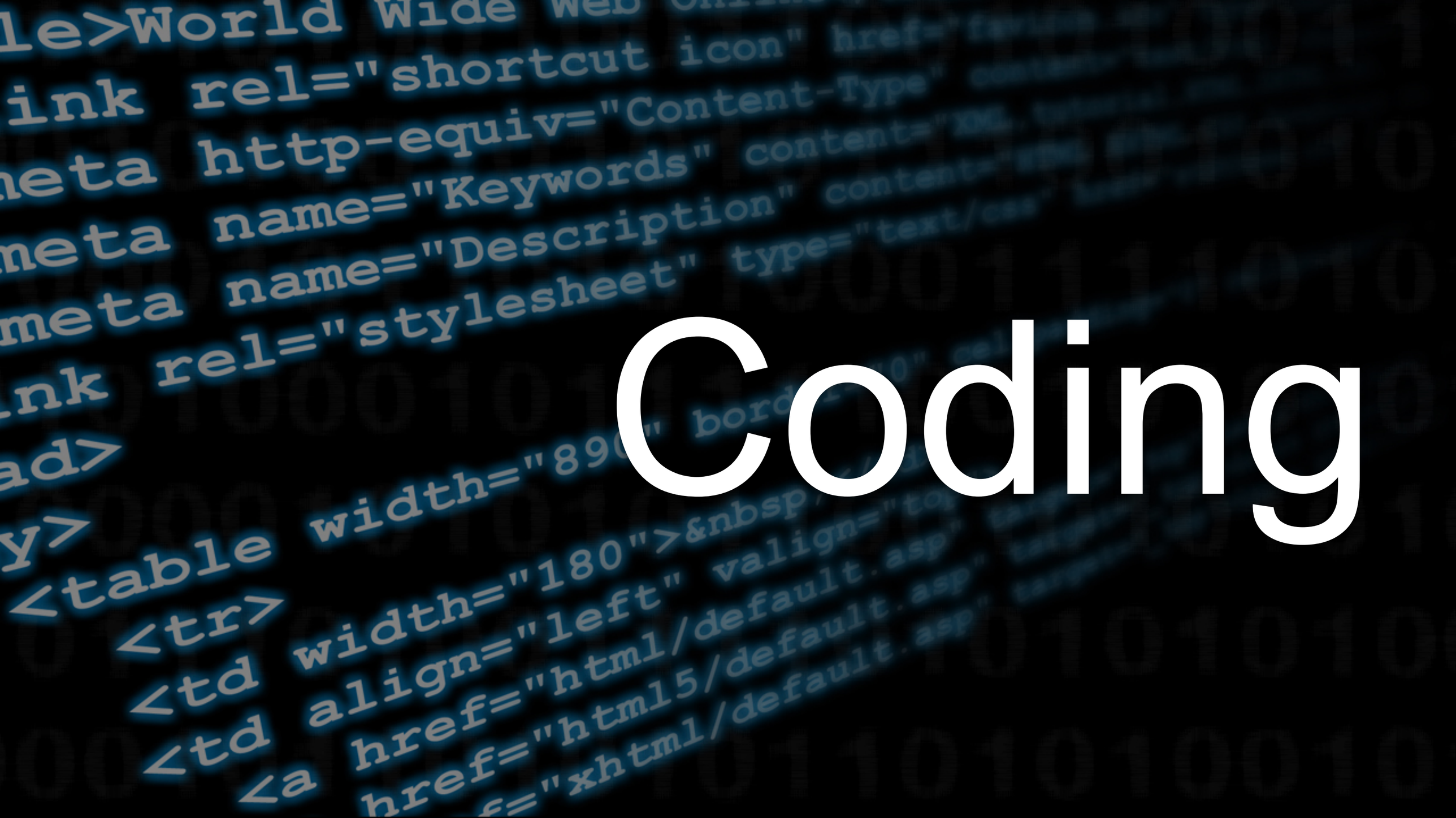
- Analytics-driven database marketing & interactive services
- Firm headquartered in Cleveland, Ohio with 70+ employees
- Only interactive marketing firm in the Midwest with a full solution of both owned in-lab and mobile eye tracking capabilities with MetricsLab[SM], a state-of-the-art usability and market research lab
- Only Magento Gold Solution Partner to offer out-of-the-box:
  - Proprietary Magento-ExactTarget Integration
  - Search analytics
  - Usability best practices

MetricsLab

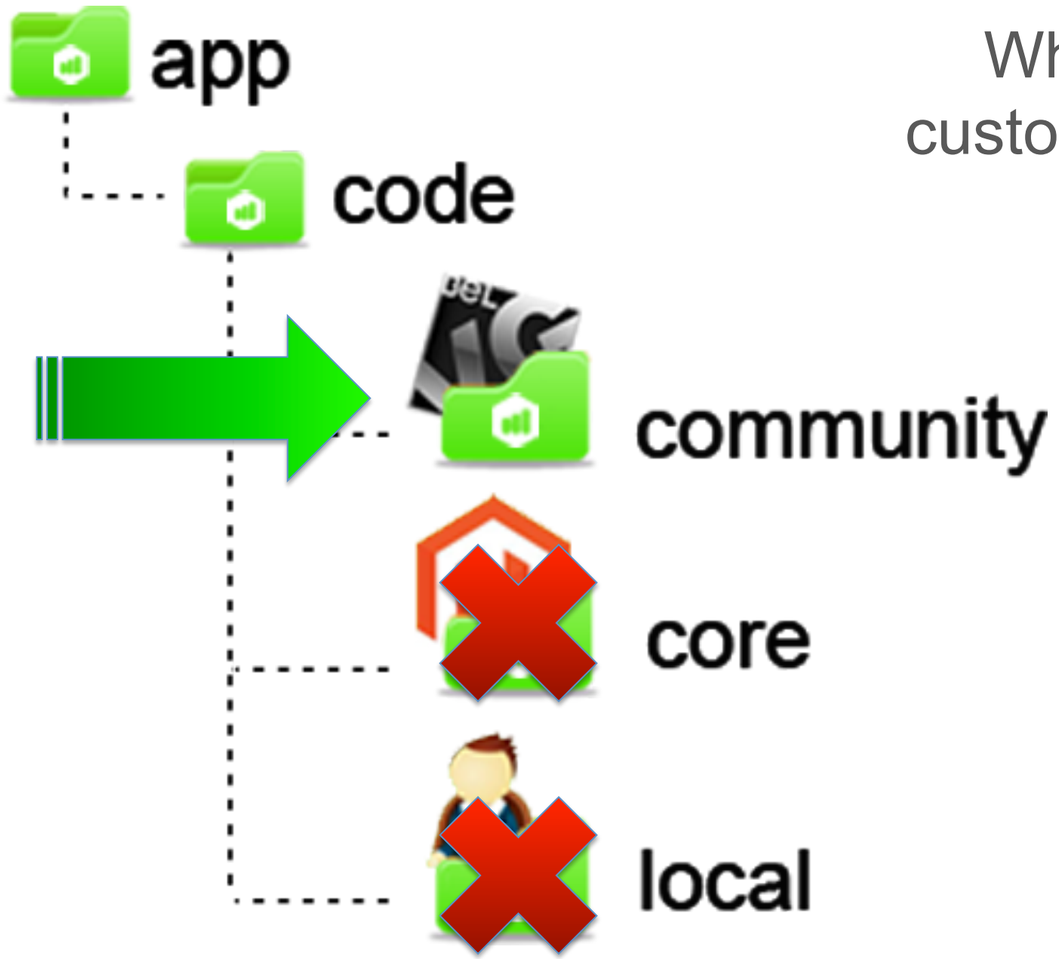
MAGENTO GOLD PARTNER

ExactTarget

imagine 2012
Magento Conference


Coding

Code Pool Usage



Where should your custom code be placed?

Allows for overrides in local code pool

Allows for module conflict resolution in local

Contributes to code portability

app/code/local/Mage

- Only use during development, **NEVER PRODUCTION!**
- Not upgrade proof
  - Must copy entire file over instead of only changed methods
  - Possible exceptions include overriding methods of abstract classes, but better practice is to create a helper, and override methods using that helper
- Using app/code/local/Mage is not PHP5
  - Copy/Paste/Edit is not OOP
  - Learn OOP
- No advantage over extending and overriding

Event Observers

- Code that executes on a specific event
- Use whenever and wherever possible, huge list of built-in event observers!
- Event observer doesn't exist for your event?
  - Create one!
- Allows multiple modules to override/extend the same functionality
- Avoid conflicts by not needing to "override an override"

Rewrite Best Practices

- When overriding a class, don't include the entire file, just the necessary methods which contain changes
- Don't copy the entire core method into your overridden functions
    - Instead, write your custom code then call `parent::functionName()` to call core code in parent method
    - Makes module more upgradable as you don't have to diff core methods on upgrades
- Use `[area]/routers/[route-name]/args/modules` instead of `global/routers/[route-name]/rewrite`

Overriding an Override

- Until Magento 2 is released (which contains new module naming conventions), use **`<depends>`** tags in your module definition files
    - Ensures proper load ordering takes place
    - Also ensures one module's code is loaded before another
    - Let's take a look at how the load order is processed…

Overriding an Override

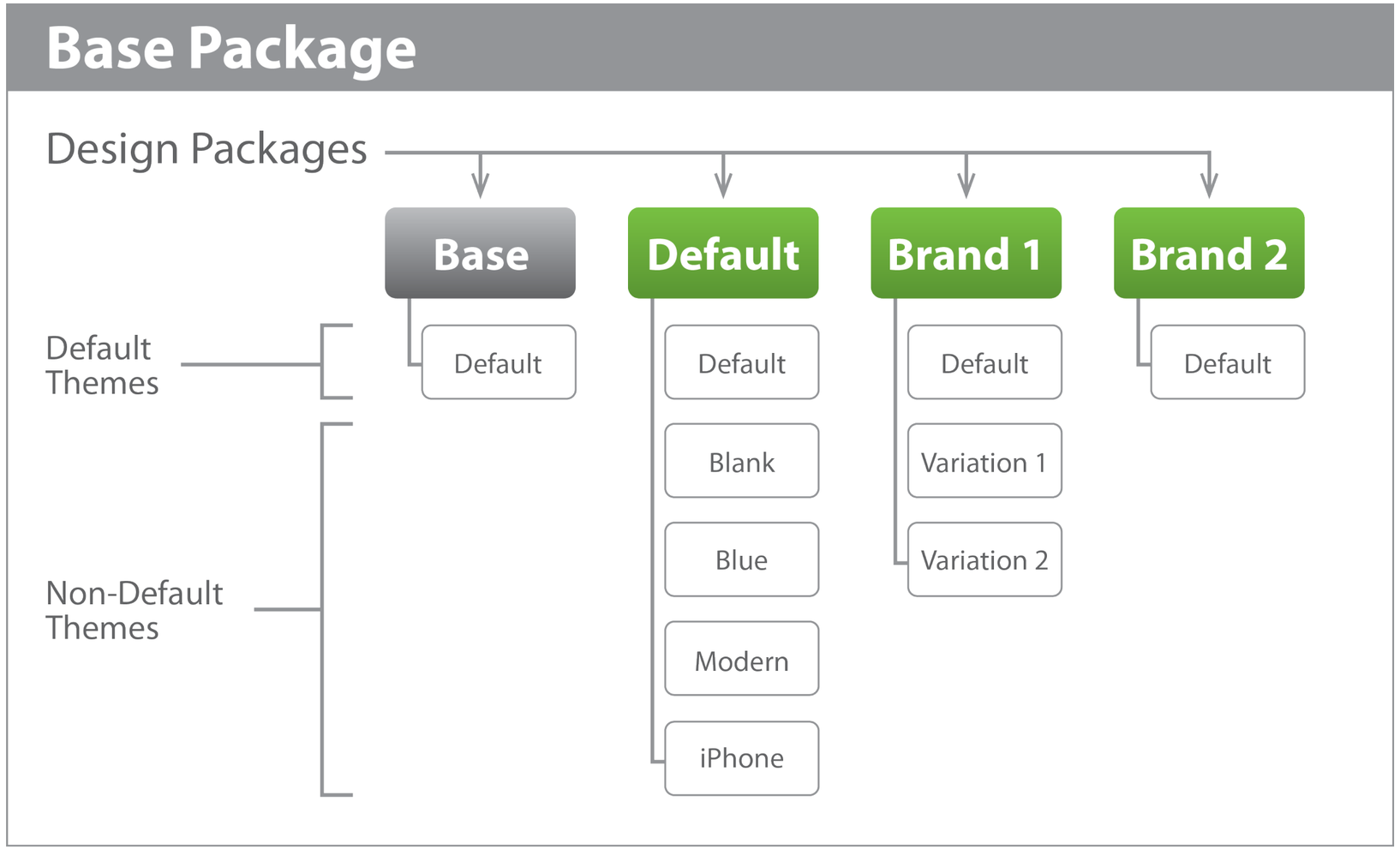
Aa_Aa community	Aa_Aa.xml	vs.	Aa_Zz community	Aa_Zz.xml
Aa_Aa community	Aa_Aa.xml	vs.	Zz_Aa community	Zz_Aa.xml
Aa_Zz community	Aa_Zz.xml	vs.	Zz_Zz community	Zz_Zz.xml
Zz_Aa community	Zz_Aa.xml	vs.	Zz_Zz community	Zz_Zz.xml
Aa_Aa community	Aa_all.xml	vs.	Aa_Zz community	Aa_all.xml
Aa_Aa community	aa_aa.xml	vs.	Zz_Zz community	Zz_Zz.xml
Aa_Aa community	aa_aa.xml	vs.	Zz_Zz community	zz_zz.xml
Aa_Aa community	Aa_Aa.xml	vs.	Mage_Aa core	Mage_Aa.xml
Aa_Aa core	Aa_Aa.xml	vs.	Zz_Zz community	Zz_Zz.xml
Aa_Aa local	Aa_Aa.xml	vs.	Zz_Zz community	Zz_Zz.xml
Aa_Aa local	Aa_all.xml (mentioned 1st)	vs.	Aa_Aa community	Aa_all.xml (mentioned 2nd)
Aa_Aa local	Aa_all.xml (mentioned 2nd)	vs.	Aa_Aa community	Aa_all.xml (mentioned 1st)

65	41	101	&#65;	A	97	61	141	&#97;	a
66	42	102	&#66;	B	98	62	142	&#98;	b
67	43	103	&#67;	C	99	63	143	&#99;	c
68	44	104	&#68;	D	100	64	144	&#100;	d
69	45	105	&#69;	E	101	65	145	&#101;	e
70	46	106	&#70;	F	102	66	146	&#102;	f
71	47	107	&#71;	G	103	67	147	&#103;	g
72	48	110	&#72;	H	104	68	150	&#104;	h
73	49	111	&#73;	I	105	69	151	&#105;	i
74	4A	112	&#74;	J	106	6A	152	&#106;	j
75	4B	113	&#75;	K	107	6B	153	&#107;	k
76	4C	114	&#76;	L	108	6C	154	&#108;	l
77	4D	115	&#77;	M	109	6D	155	&#109;	m
78	4E	116	&#78;	N	110	6E	156	&#110;	n
79	4F	117	&#79;	O	111	6F	157	&#111;	o
80	50	120	&#80;	P	112	70	160	&#112;	p
81	51	121	&#81;	Q	113	71	161	&#113;	q
82	52	122	&#82;	R	114	72	162	&#114;	r
83	53	123	&#83;	S	115	73	163	&#115;	s
84	54	124	&#84;	T	116	74	164	&#116;	t
85	55	125	&#85;	U	117	75	165	&#117;	u
86	56	126	&#86;	V	118	76	166	&#118;	v
87	57	127	&#87;	W	119	77	167	&#119;	w
88	58	130	&#88;	X	120	78	170	&#120;	x
89	59	131	&#89;	Y	121	79	171	&#121;	y
90	5A	132	&#90;	Z	122	7A	172	&#122;	z

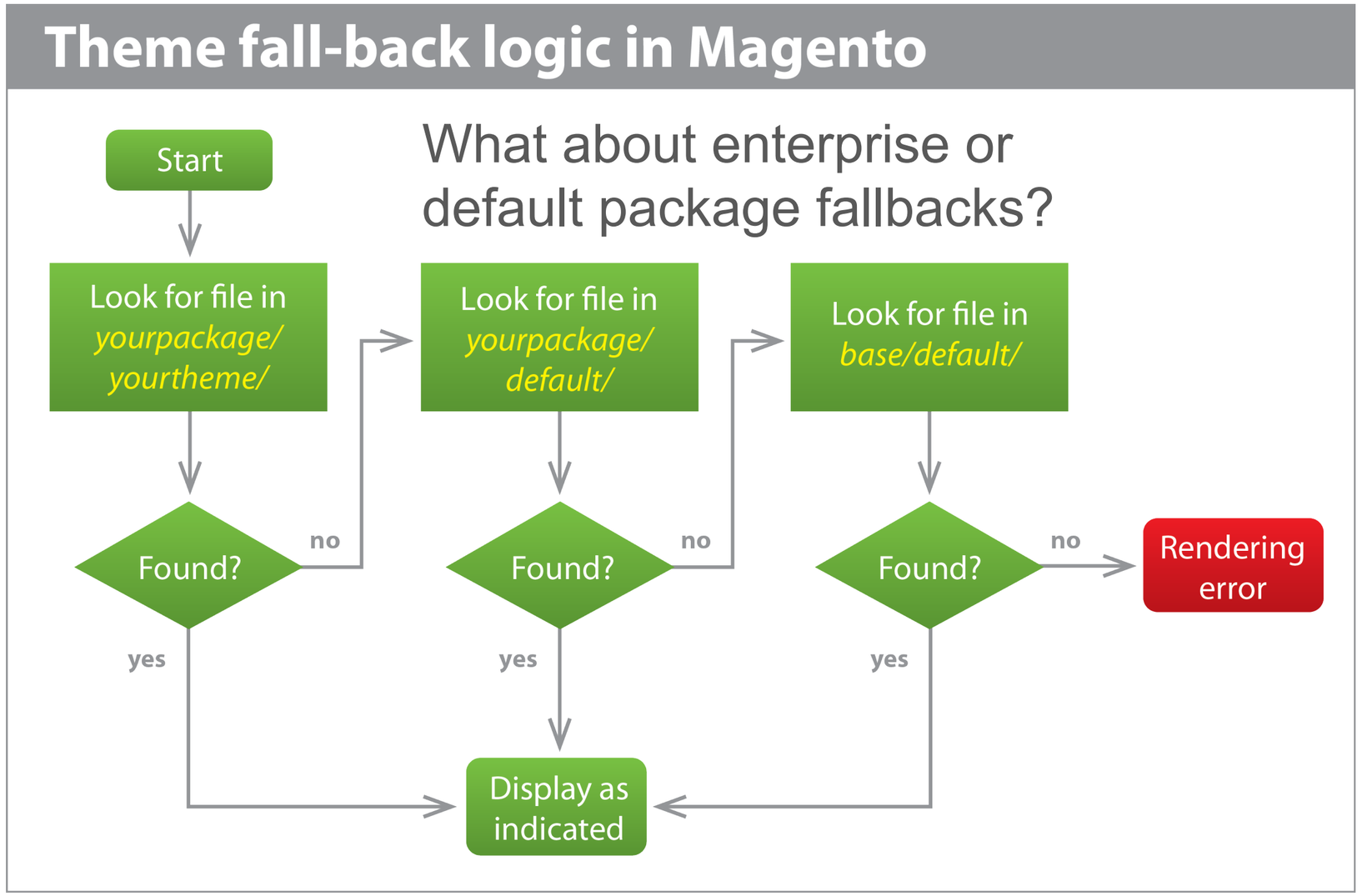
Courtesy of @foomanNZ


Theming

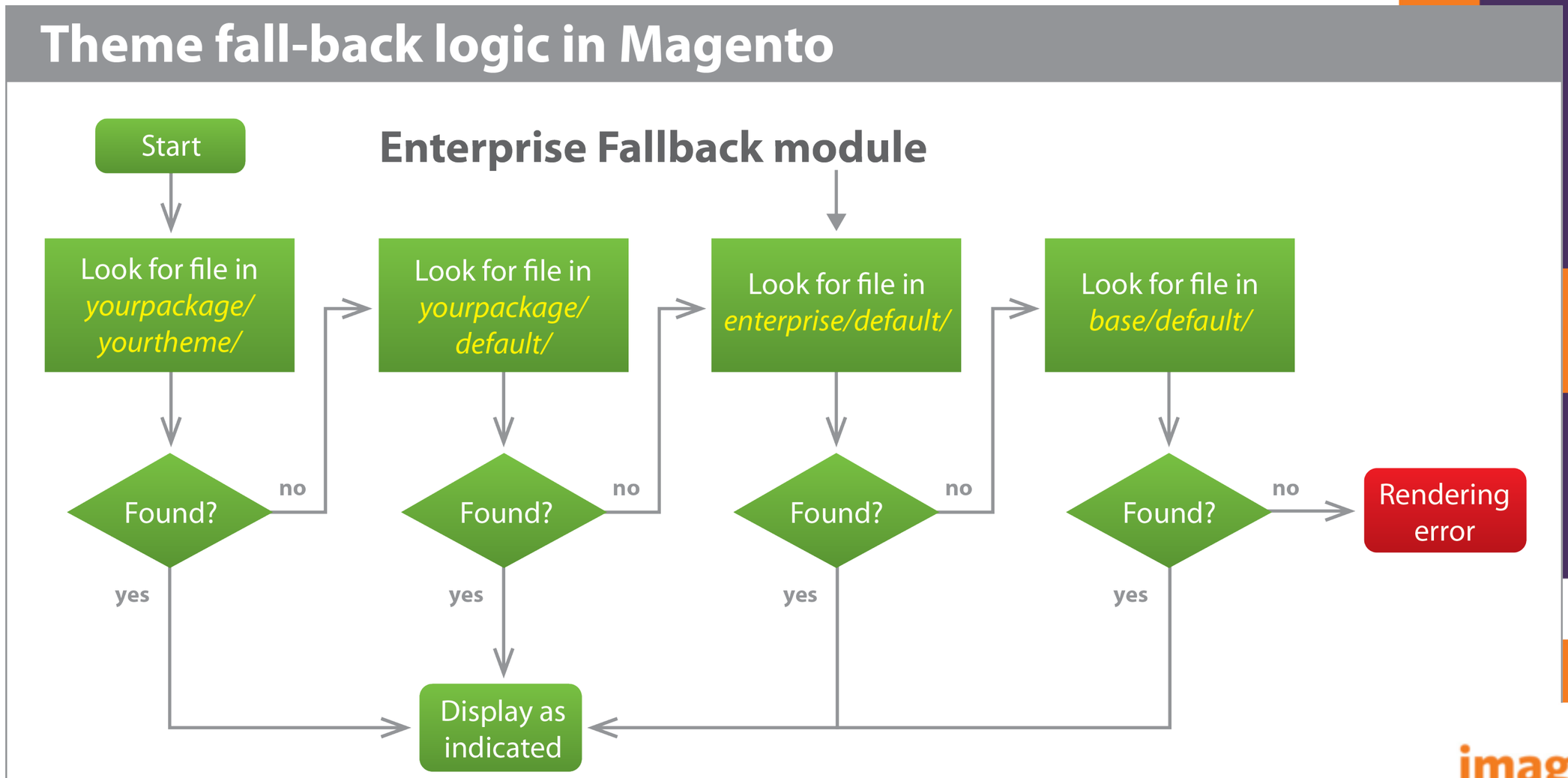
Fallback Layer



Fallback Layer



Fallback Layer



More Fallback Layer & CSS

- Copying default/default to default/yourtheme?
  - You're doing it **all** wrong!

- To inherit enterprise or default package functionality, use the Enterprise Fallback module
  - Adds these packages to the fallback layer between your package and base package
  - Copy only files with modifications
  - Make changes in layout XML whenever possible to avoid having to diff out phtml files on upgrades
  - Contributes to more maintainable codebase by having less files to maintain

- Use LESS CSS or SASS for more maintainable CSS

Design local.xml

- The local.xml file should contain XML updates that apply to just your store
- XML updates specific to modules (or distributed themes) should take place in separate layout definition files (which are respectively ‘owned’ by those modules)
- Let’s other developers that join project at a later date know exactly where to find your changes
- Do NOT copy entire XML files from base package
  - Don’t “own” others’ logic you don’t wish to modify
  - Could potentially cause conflicts during upgrades

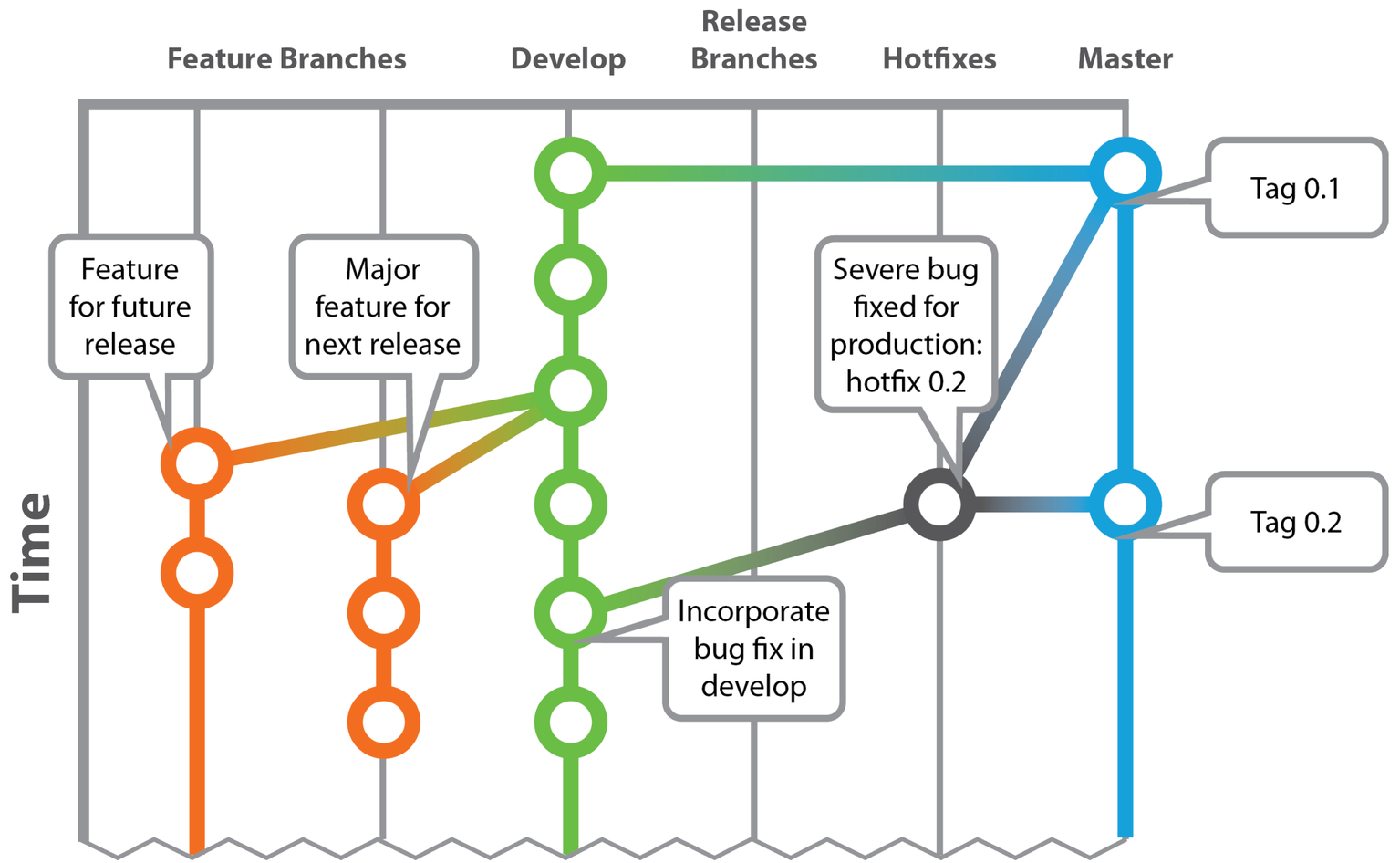
VCS

Version Control & Git

- Use with **every** Magento project
- Use a strong .gitignore file to exclude certain files/dirs
  - https://bitbucket.org/markoshust/magento_gitignore
- Not using Git? (you should be)
  - Let's you track directory and file permissions
    - `find app/code/core -type d -exec chmod 0555 {} \;`
    - `find app/code/core -type f -exec chmod 0444 {} \;`
    - & commit!
  - Use "git assume-unchanged" for local file changes

Git Flow

- Develop multiple features simultaneously
- Controls all of the branch management features, including:
  - merging
  - branching
  - rebasing
- Allows for hotfixes and tag management for releases



Deployment

- Code review a necessity for every commit
    - Built into many hosted repository solutions (bitbucket, github, etc.) in the form of pull requests
- Utilize deployment strategy
    - Never login to a production box (security, lockdown file changes, ensure file integrity, etc.)
    - Allows easy rollback of code on production
    - Utilize deployment strategy: Springloops, Beanstalk, or custom in the form of post-commit hooks

Database

Database Updates & Changes

- Don’t **add/modify/delete** any core tables or fields
- Better Practice: create custom tables and fields, and join on core tables if relation is needed

	A	B	C	D
1	yourmodule_id	name	status	customer_id
2	1	Foo	1	42
3	2	Bar	1	11
4	3	Baz	2	63
5	4	Qux	1	23
6	5	Corge	2	77

- Keeps database changes completely upgradable and easy to uninstall
  - Include uninstall scripts with your custom modules

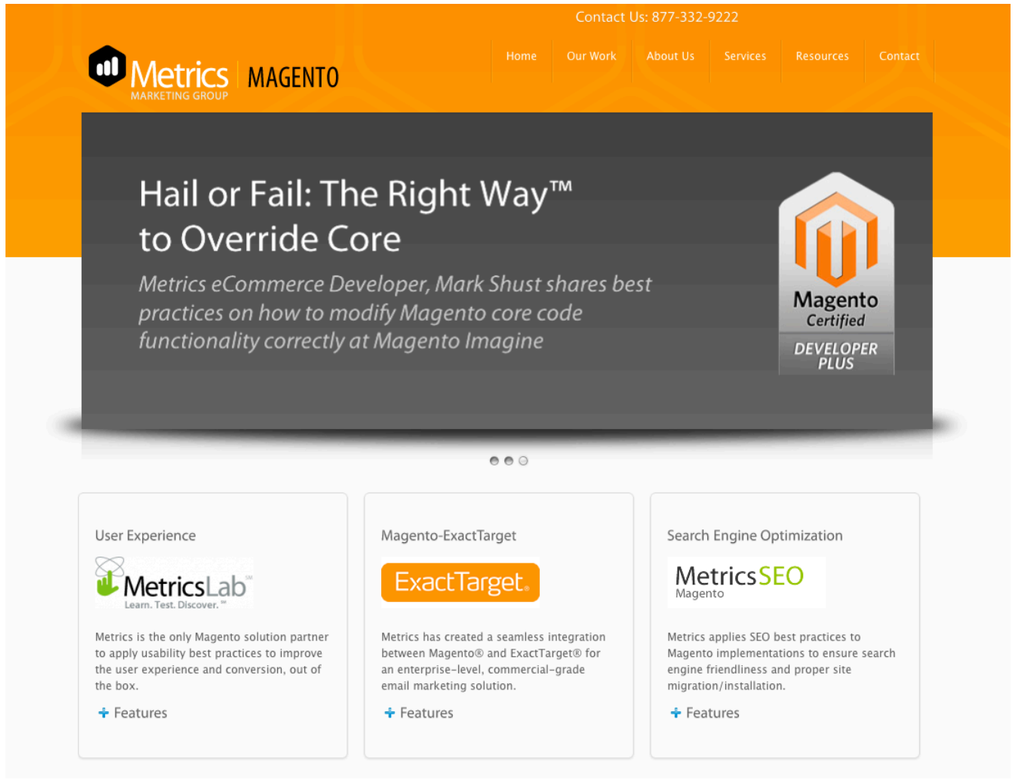
Testing

Unit Testing

- Magento Testing Automation Framework
  - https://github.com/magento/taf
  - http://www.magentocommerce.com/blog/comments/automate-your-testing-with-the-magento-test-automation-framework/
- Use Xdebug for more verbose error messages
- Write unit tests for custom modules to ensure module reliability with Magento version upgrades

Thank you!

- Visit http://ecommerce.metricsmarketing.com for links & references, including additional content and information



- Questions/Comments

imagine 2012
Magento Conference

@MetricsMrktg
#MagentoImagine

GET SEEN WEARING
THE GREEN