OSCON 2009

Jonathan Lloyd (majrmovies)
Who am I?

• I have been programming Perl for ~ 4 years

• I work for a small business (~ 30 employees) in Irvine, CA that does e-commerce and distribution.

• I do primarily web programming with Perl, mod_perl2, CGI and JavaScript. Including lots of web service communications like SOAP, XML, and JSON.
The Presentations

• Distributed Applications with CouchDB - J Chris Anderson
• Open Source Language Roundtable
• Transparent Sharing of Complex Data with YAML - Ingy döt Net
• Zen and the Art of Abstraction Maintenance - Alex Martelli
• Gearman: Building a Distributed Platform - Eric Day and Brian Aker
• 7 Principles of Better API Design - Damian Conway
• Situation Normal, Everything Must Change - Simon Wardley
The Presentations

- **Distributed Applications with CouchDB** - J Chris Anderson
- **Open Source Language Roundtable**
- **Transparent Sharing of Complex Data with YAML** - Ingy döt Net
- **Zen and the Art of Abstraction Maintenance** - Alex Martelli
- **Gearman: Building a Distributed Platform** - Eric Day and Brian Aker
- **7 Principles of Better API Design** - Damian Conway
- **Situation Normal, Everything Must Change** - Simon Wardley
The Presentations

- Distributed Applications with CouchDB - J Chris Anderson
- Open Source Language Roundtable
- Transparent Sharing of Complex Data with YAML - Ingy döt Net
- Zen and the Art of Abstraction Maintenance - Alex Martelli
- Gearman: Building a Distributed Platform - Eric Day and Brian Aker
- 7 Principles of Better API Design - Damian Conway
- Situation Normal, Everything Must Change - Simon Wardley

Disclaimer: I am not these people. I just listened to these people.
The Presentations

- Distributed Applications with CouchDB - J Chris Anderson
- Open Source Language Roundtable
- Transparent Sharing of Complex Data with YAML - Ingy döt Net
- Zen and the Art of Abstraction Maintenance - Alex Martelli
- Gearman: Building a Distributed Platform - Eric Day and Brian Aker
- 7 Principles of Better API Design - Damian Conway
- Situation Normal, Everything Must Change - Simon Wardley

Disclaimer: I am not these people. I just listened to these people. And stole some of their materials for this presentation.
The Presentations

- **Distributed Applications with CouchDB** - J Chris Anderson
- **Open Source Language Roundtable**
- **Transparent Sharing of Complex Data with YAML** - Ingy döt Net
- **Zen and the Art of Abstraction Maintenance** - Alex Martelli
- **Prism, Bringing Web Applications to the Desktop** - Matthew Gertner
- **Gearman: Building a Distributed Platform** - Eric Day and Brian Aker
- **7 Principles of Better API Design** - Damian Conway
- **Situation Normal, Everything Must Change** - Simon Wardley
Perl: The Metrics

- Perl is 21 year’s old.
Perl: The Metrics

- Perl is 21 year’s old.
- Perl 5 is 14 years old.
Perl: The Metrics

- Perl is 21 year’s old.
- Perl 5 is 14 years old.
- Larry Wall is 55 years old.
Perl: The Metrics

• Perl is 21 year’s old.

• Perl 5 is 14 years old.

• Larry Wall is 55 years old.

• The idea of Perl 6 was introduced to the community on October 24th, 2000.
Perl 6: Why?

- We have 20 years of experience with the language.
Perl 6: Why?

• We have 20 years of experience with the language.

• We have a much better Larry.
Perl 6: Why?

• We have 20 years of experience with the language.

• We have a much better Larry.

• We have Damian Conway
Perl 6: Why?

- We have 20 years of experience with the language.
- We have a much better Larry.
- We have Damian Conway
- “It’s time to steal all the good ideas from other languages.”
Perl 6: Why?

- We have 20 years of experience with the language.
- We have a much better Larry.
- We have Damian Conway
- “It’s time to steal all the good ideas from other languages.”
Perl 6: Why?

- We have 20 years of experience with the language.
- We have a much better Larry.
- We have Damian Conway
- “It’s time to steal all the good ideas from other languages.”
Perl 6: Seriously?

Comments are inline-able

use v5;

my $x

# This is a comment to the end of the line
    = 1;

use v6;

my $y #{ Need a better var name!} = 2;
Perl 6: Seriously?

String lists

code

use v5;

    # throw some strings in to an array
    my @names = qw(Jonathan David Lloyd);

    # throw variables and strings -- no more qw!
    my @meals = ($breakfast, 'Lunch', 'Dinner');

code

use v6;

    # The qw list constructor gets prettier
    my @names = <Jonathan David Lloyd>;

    # Interpolates variables or strings
    my @meals = «$breakfast Lunch Dinner»;

    my @names = «Jonathan # This is my first name»

    »
Perl 6: Seriously?

Everything is an object

use v5;
say keys %hash;
say values %hash;

join(‘-‘, $year, $month, $day);
for (sort keys %hash) { say; }

use v6;

%hash.keys.say;

%hash.keys.sort.join(‘ | ‘);
%hash.keys.reverse.join(‘-‘).say;

.say for %hash.keys.sort;
Perl 6: Seriously?

Variable declarations

use v5;

    my $variable = "String";
    my $variable = 10;
    my $variable = new CGI;

    my @array = ('String', 10, $object);

use v6;

    my Str $variable = 'a scalar';
    my Int $variable = 10;

    my Str @array = < Jonathan David Lloyd >;
    my Int @array = 1..10;
Perl 6: Seriously?

Junctions

use v5;

    my @odds = qw(1 3 5 7 9);
    my @nums = qw(0 1 2 3 4 5 6 7 8 9);

    for my $num (@nums) {
        if (grep $_ eq $num, @odds) {
            say "$_ is odd"; ...
        }
    }

use v6;

    for (@nums) {
        say "$_ is odd" if $_ == any (@odds);
        say "$_ in even" if $_ == none (@odds);
    }

    # The comparisons are performed in parallel!
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Strictures and warnings on by default</td>
</tr>
<tr>
<td>2.</td>
<td>Comments are inline-able</td>
</tr>
<tr>
<td>3.</td>
<td>Big revamp of POD</td>
</tr>
<tr>
<td>4.</td>
<td>Identifiers</td>
</tr>
<tr>
<td>5.</td>
<td>String lists</td>
</tr>
<tr>
<td>6.</td>
<td>Sigils sanitized</td>
</tr>
<tr>
<td>7.</td>
<td>Everything is an object</td>
</tr>
<tr>
<td>8.</td>
<td>Variable declarations</td>
</tr>
<tr>
<td>9.</td>
<td>State variables</td>
</tr>
<tr>
<td>10.</td>
<td>Constants</td>
</tr>
<tr>
<td>11.</td>
<td>Lists</td>
</tr>
<tr>
<td>12.</td>
<td>Generators</td>
</tr>
<tr>
<td>13.</td>
<td>Pairs</td>
</tr>
<tr>
<td>14.</td>
<td>Smarter string interpolations</td>
</tr>
<tr>
<td>15.</td>
<td>Heredocs fixed</td>
</tr>
<tr>
<td>16.</td>
<td>Junctions</td>
</tr>
<tr>
<td>17.</td>
<td>Array indexing</td>
</tr>
<tr>
<td>18.</td>
<td>Multidimensional arrays</td>
</tr>
<tr>
<td>19.</td>
<td>Hash features</td>
</tr>
<tr>
<td>20.</td>
<td>Data-preserving hash transformations</td>
</tr>
<tr>
<td>21.</td>
<td>Operator revamp</td>
</tr>
<tr>
<td>22.</td>
<td>DWIMier comparisons</td>
</tr>
<tr>
<td>23.</td>
<td>DWIMier matching</td>
</tr>
<tr>
<td>24.</td>
<td>Switch statements and switch loops</td>
</tr>
<tr>
<td>25.</td>
<td>Defaulting operators</td>
</tr>
<tr>
<td>26.</td>
<td>IO</td>
</tr>
<tr>
<td>27.</td>
<td>Sort has been fixed</td>
</tr>
<tr>
<td>28.</td>
<td>Revamped loops</td>
</tr>
<tr>
<td>29.</td>
<td>Nested postfix control statements</td>
</tr>
<tr>
<td>30.</td>
<td>Error variables</td>
</tr>
<tr>
<td>31.</td>
<td>Subroutines</td>
</tr>
<tr>
<td>32.</td>
<td>Named parameters</td>
</tr>
<tr>
<td>33.</td>
<td>Parameter types and return types</td>
</tr>
<tr>
<td>34.</td>
<td>Captures</td>
</tr>
<tr>
<td>35.</td>
<td>“Slurpy” parameters</td>
</tr>
<tr>
<td>36.</td>
<td>The MAIN subroutine</td>
</tr>
<tr>
<td>37.</td>
<td>Classes</td>
</tr>
<tr>
<td>38.</td>
<td>Inheritance</td>
</tr>
<tr>
<td>39.</td>
<td>Constructors and destructors</td>
</tr>
<tr>
<td>40.</td>
<td>Multiple dispatch</td>
</tr>
<tr>
<td>41.</td>
<td>Roles</td>
</tr>
<tr>
<td>42.</td>
<td>Regular Expressions</td>
</tr>
<tr>
<td>43.</td>
<td>Named regexes</td>
</tr>
<tr>
<td>44.</td>
<td>Match-time variable interpolation</td>
</tr>
<tr>
<td>45.</td>
<td>Named regexes and grammars</td>
</tr>
</tbody>
</table>
Perl 6: How?

- Download “Rakudo”
- It won’t hurt your current distribution/system
- Use modules built for Perl5 that are similar to Perl6::*
The Presentations

- Distributed Applications with CouchDB - J Chris Anderson
- Open Source Language Roundtable
- Transparent Sharing of Complex Data with YAML - Ingy döt Net
- Zen and the Art of Abstraction Maintenance - Alex Martelli
- Prism, Bringing Web Applications to the Desktop - Matthew Gertner
- Gearman: Building a Distributed Platform - Eric Day and Brian Aker
- 7 Principles of Better API Design - Damian Conway
- Situation Normal, Everything Must Change - Simon Wardley
Distributed Applications with CouchDB

- **Document-oriented,** not *relational* database.

- Schema-Free (JSON)

- RESTful HTTP API

- JavaScript Powered Map/Reduce Views

- N-Master Replication, Highly Concurrent, Robust Storage, Buzz word, Buzz word, Buzz word.
Distributed Applications with CouchDB

- **Document-oriented**, not relational database.

- **Documents in the Real World**
  - Bills, letters, tax forms ..
  - Same type != same structure
  - Can be out of date
  - No references

- **Natural Data Behavior**
Distributed Applications with CouchDB

- **Schema-Free (JSON)**

  ```json
  {
    "_id": "BCCD12CBB",
    "_rev": "3-AB764C",
    "type": "person",
    "name": "Darth Vader",
    "age": 63,
    "headware": [
      "Helmet",
      "Sombrero"
    ],
    "dark_side": true
  }
  ```

- Unique ID for each document
- Data structure can change on a per-document basis
- Limited only by the data structures available in JSON
Distributed Applications with CouchDB

- RESTful HTTP API
  - Create HTTP PUT /db/mydocid
  - Read HTTP GET /db/mydocid
  - Update HTTP PUT /db/mydocid
  - Delete HTTP DELETE /db/mydocid

```javascript
function(doc, req) {
  // Ijson templates.post
  // Ijson blog
  // Icode helpers.template
  // Icode helpers.couchapp
  // log(req.headers.Accept);

  // we only show html
  return template(templates.post, {
    title : doc.title,
    blogName : blog.title,
    post : doc.html,
    date : doc.created_at,
    author : doc.author,
    assets : assetPath(),
    editPostPath : showPath('edit', doc...id),
    index : listPath('index', 'recent-posts', {descending:true, limit: 8})
  });
}
```

Hello World For Real This Time

And a bag of chips.

- by J Chris A, 5 weeks ago
- by Jason Davies, 5 weeks ago
- by Jason Watkins, 5 weeks ago
- by J Chris A, 5 weeks ago
Distributed Applications with CouchDB

- RESTful HTTP API

```perl
use JSON;

require LWP::UserAgent;
my $ua = LWP::UserAgent->new;
$ua->timeout(10);
$ua->env_proxy;

my $response = $ua->get('http://localhost/db/mydocid');
if ($response->is_success) {
    my $document = from_json( $response->content );
} else {
    die $response->status_line;
}

use DBI;
my $dbh = DBI->connect or die $DBI::errstr;

my $sth = $dbh->prepare('SELECT * FROM db WHERE id = ?');
$sth->execute(1);
my $item = $sth->fetchrow_hashref;
```
Distributed Applications with CouchDB

- Javascript-Powered Map/Reduce Functions

### Documents

```
{"user": "Chris",  
 "points": 3 }  
{"user": "Joe",  
 "points": 10 }  
{"user": "Alice",  
 "points": 5 }  
{"user": "Mary",  
 "points": 9}  
{"user": "Bob",  
 "points": 7}  
```

### Map

```
function(doc) {
    if (doc.user && doc.points) {
        emit(doc.user, doc.points);
    }
}
```

### Reduce

```
function(keys, values, rereduce) {
    return sum(values);
}
```

```
{"key": "Alice", "value": 5}  
{"key": "Bob", "value": 7}  
{"key": "Chris", "value": 3}  
{"key": "Joe", "value": 10}  
{"key": "Mary", "value": 9}  
```

```
Alice ... Chris: 15  
Everyone: 34  
```
Distributed Applications with CouchDB

- N-Master Replication, Highly Concurrent, Robust Storage..

Pull
```
-d '{
  "source":"http://server/db",
  "target":"db-replica"
}'
```

Push
```
-d '{
  "source":"db-replica",
  "target":"http://server/db"
}'
```

Remote
```
-d '{
  "source":"http://server-one/db",
  "target":"http://server-two/db"
}'
```
Distributed Applications with

Pros

• Terrific Idea

• Leverage Apache for its strength -- distributing documents

• Use client-side JavaScript to manage and display documents

• Replication across multiple servers, or being downloadable to offline applications is very simple

Cons

• Security (HTTP DELETE /db -- Oops!)

• Using Perl would require a DBD::CouchDB plugin for sanity

• Writing queries/views is not practical in a small shop

• No direct interface -- runs as a daemon that is simply killed

• Very JavaScript oriented
The Presentations

- Distributed Applications with CouchDB - J Chris Anderson
- Open Source Language Roundtable
- Transparent Sharing of Complex Data with YAML - Ingy döt Net
- Zen and the Art of Abstraction Maintenance - Alex Martelli
- Prism, Bringing Web Applications to the Desktop - Matthew Gertner
- Gearman: Building a Distributed Platform - Eric Day and Brian Aker
- 7 Principles of Better API Design - Damian Conway
- Situation Normal, Everything Must Change - Simon Wardley
Open Source Language Roundtable

Java: Rod Johnson (SpringSource)
Perl: Jim Brandt (Perl Foundation)
PHP: Laura Thomason (Mozilla)
Python: Alex Martelli (Google)
Ruby: Brian Ford (Engine Yard)

- Most dynamic programming languages are inherently the same. **Don’t hate.**

- **Perl** is the best for shell scripting

- **JavaScript** is a dynamic language completely undervalued, but hugely important in web development (i.e. Google)
  - Runs on the client-side
  - AJAX has enabled more dynamic communication with the server
  - Frameworks like Prototype, Dojo, Moo Tools, and jQuery make it easy
The Presentations

- **Distributed Applications with CouchDB** - J Chris Anderson
- **Open Source Language Roundtable**
- **Transparent Sharing of Complex Data with YAML** - Ingy döt Net
- **Zen and the Art of Abstraction Maintenance** - Alex Martelli
- **Prism, Bringing Web Applications to the Desktop** - Matthew Gertner
- **Gearman: Building a Distributed Platform** - Eric Day and Brian Aker
- **7 Principles of Better API Design** - Damian Conway
- **Situation Normal, Everything Must Change** - Simon Wardley
Transparent Sharing of Complex Data with YAML  
by Ingy döt Net (Hackers, Inc)

- YAML (YAML Ain't Markup Language)
- JSON == YAML
- YAML != JSON
- YAML can store objects
- YAML can be streamed
- YAML has implementations in 8 different languages -- more to come ...

```yaml
---
name: ingy
age: old
weight: heavy
# I should comment that I also
# like pink, but don't tell anybody.
favorite colors:
  - red
  - green
  - blue
---
- Clark Evans
- Oren Ben-Kiki
- Ingy döt Net
...
The Presentations

- **Distributed Applications with CouchDB** - J Chris Anderson
- **Open Source Language Roundtable**
- **Transparent Sharing of Complex Data with YAML** - Ingy döt Net
- **Zen and the Art of Abstraction Maintenance** - Alex Martelli
- **Prism, Bringing Web Applications to the Desktop** - Matthew Gertner
- **Gearman: Building a Distributed Platform** - Eric Day and Brian Aker
- **7 Principles of Better API Design** - Damian Conway
- **Situation Normal, Everything Must Change** - Simon Wardley
Zen and the Art of Abstraction Maintenance
by Alex Martelli (Google)

- Everything is built on something.
- You build layers of abstraction (Perl modules)
- All layers of abstraction leak.
- Understand the layers surrounding your code.

![Chart showing most concerning threats](image)

*Figure 4: Most Concerning Threats*
Source: Arbor Networks, Inc.
The Presentations

- Distributed Applications with CouchDB - J Chris Anderson
- Open Source Language Roundtable
- Transparent Sharing of Complex Data with YAML - Ingy döt Net
- Zen and the Art of Abstraction Maintenance - Alex Martelli
- Prism, Bringing Web Applications to the Desktop - Matthew Gertner
- Gearman: Building a Distributed Platform - Eric Day and Brian Aker
- 7 Principles of Better API Design - Damian Conway
- Situation Normal, Everything Must Change - Simon Wardley
Bringing Web Applications to the Desktop
by Matthew Gertner

- The browser wasn’t designed for running applications -- but it is being used that way

- **HTML5** is furthering this effort
  - Offline Operation
  - Local Data
  - Worker Threads

- **Prism** allows you to spin a process (separate from the browser) and interact with the OS using JavaScript calls to the API
The Presentations

- Distributed Applications with CouchDB - J Chris Anderson
- Open Source Language Roundtable
- Transparent Sharing of Complex Data with YAML - Ingy döt Net
- Zen and the Art of Abstraction Maintenance - Alex Martelli
- Prism, Bringing Web Applications to the Desktop - Matthew Gertner
- Gearman: Building a Distributed Platform - Eric Day and Brian Aker
- 7 Principles of Better API Design - Damian Conway
- Situation Normal, Everything Must Change - Simon Wardley
• Gearman provides a distributed application **framework**
  
  • **Clients** - Create jobs to be run and sends them to a job server.
  
  • **Workers** - Register with a job server and grab jobs to run.
  
  • **Job Server** - Coordinate the assignment from the client to the works, handle restarts.
  
  • “Gearman, like managers, assign the tasks but do none of the work.”
Not everything needs immediate attention

- E-mail notifications
- Certain DB updates
- RSS indexing
- Search indexing

Image Processing
Build Your Own Distributed Platform in 3 Hours

- Background tasks
- Foreground tasks
- Asynchronous tasks
- No single point of failure (multiple job servers, multiple workers)
- Workers can be specific to certain jobs

Diagram:

- Apache
  - PHP
  - Gearman Job Server
  - Resize Worker
- Apache
  - PHP
  - Gearman Job Server
  - Resize Worker
- Apache
  - PHP
  - Storage
    - NFS, MogileFS
  - Resize Worker
Build Your Own Distributed Platform in 3 Hours

**Pros**

- Written in C
- Perl API on CPAN (**Gearman::XS**)
- Command line tool
- Multi-language - mix client and workers
- Synchronous and Asynchronous queues
- Runs as a daemon (gearmand)
- Developing improved monitoring (statistics, configuration management, etc.)

**Cons**

- Only accepts a single string / file handle from Client to Worker
- Failure by worker -- not enough configuration (would rather it be function specific)
The Presentations

- **Distributed Applications with CouchDB** - J Chris Anderson
- **Open Source Language Roundtable**
- **Transparent Sharing of Complex Data with YAML** - Ingy döt Net
- **Zen and the Art of Abstraction Maintenance** - Alex Martelli
- **Prism, Bringing Web Applications to the Desktop** - Matthew Gertner
- **Gearman: Building a Distributed Platform** - Eric Day and Brian Aker
- **7 Principles of Better API Design** - Damian Conway
- **Situation Normal, Everything Must Change** - Simon Wardley
7 Principles of Better API Design
by Damian Conway

1. Do one thing **really well**

```perl
# read a file in to a variable
my $text = do { local (@ARGV, $/) = filename; <> };

use Perl6::Slurp;

my $text = slurp $fh;
my $text2 = slurp 'filename';
```
2. Design by **coding** *(work backwards)*

```perl
# regex for floating point integer
my $input =~
    /([+-]?(?:\d+\.|\.[\d]*)\d*|\.[\d]+(?:[eE][+-]?\d+))*/;

use Regexp::Common;
my $input =~ /($RE{num}{real})/;
my $input2 =~ /($RE{num}{int})/;
my $input3 =~ /($RE{num})/;
```
3. Evolve by Subtraction

use IO::Prompt;
while (prompt "next: ", -bool, -chomped) {
    print "You said \'$_\'\n";
}

while (prompt "next: ") { # autodetect, autochomp
    print "You said \'$_\'\n";
}
7 Principles of Better API Design
by Damian Conway

4. Declarative beats imperative

use Getopt::Euclid;

for my $x (0 .. $ARGV->{size}{h} - 1) {
    for my $y (0 .. $ARGV->{size}{w} - 1) {
        do_something_with($x, $y);
    }
}

__END__

=item -s[ize]=<h>x<w>

Specify size of simulation

=for Euclid:
  h.type:  int > 0
  h.default:  24
  w.type:  int >= 10
  w.default:  80
7 Principles of Better API Design
by Damian Conway

5. Preserve the metadata

```perl
use Config::Std;

read_config $file_name => my %config;
# update %config here
write_config %config => $file_name;

read_config $file_name => my %config;
# update %config here
write_config %config;

bash> cd /home/jlloyd
bash> ls
bash> cd docs/
bash> ls
bash> cd modules/
bash> ls
You have typed the commands "cd [path] ls" 3 times, would you like to create an alias? [y|n]
```
6. Leverage the familiar

```
use Log::Log4perl;
Log::Log4perl->init($log_config_file);

my $logger = Log::Log4perl->get_logger(__PACKAGE__);

$logger->debug('this is a debug message');
$logger->info('this is an info message');
$logger->warn('this is a warning message');
$logger->error('this is an error message');
$logger->fatal('this is a fatal message');

use Log::StdLog { file => $log_file }
print STDLOG debug => 'this is a debug message';
print STDLOG info => 'this is an info message';
print STDLOG warn => 'this is a warning message';
print STDLOG error => 'this is an error message';
print STDLOG fatal => 'this is a fatal message';
```
7 Principles of Better API Design
by Damian Conway

7. The best code is no code at all

    my $obj = MyClass->new('data');
    print $obj;

    MyClass=HASH[0x12a8f2];

    my $obj = MyClass->new('data');

    use Object::Dumper;
    print $obj;
The Presentations

- **Distributed Applications with CouchDB** - J Chris Anderson
- **Open Source Language Roundtable**
- **Transparent Sharing of Complex Data with YAML** - Ingy döt Net
- **Zen and the Art of Abstraction Maintenance** - Alex Martelli
- **Prism, Bringing Web Applications to the Desktop** - Matthew Gertner
- **Gearman: Building a Distributed Platform** - Eric Day and Brian Aker
- **7 Principles of Better API Design** - Damian Conway
- **Situation Normal, Everything Must Change** - Simon Wardley
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization
Situation Normal, Everything Must Change

by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
Situation Normal, Everything Must Change

by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
- Competitive disadvantage to companies that fail to follow the market
Situation Normal, Everything Must Change
by Simon Wardley

- All good innovations undergo a process of commoditization (i.e. Electricity)
- This trend can be mapped and seen in the market (i.e. CRM)
- Competitive disadvantage to companies that fail to follow the market
Situation Normal, Everything Must Change
by Simon Wardley

Pros
- Economies of scale
- Pay per use
- Speed to market
- Focus on core
- Price competition
- Not “locked-in”
- Secondary sourcing

Cons
- Management of data/applications (different way of thinking/designing)
- Trust / Security
Thanks!

Questions?