DevOps: API Management and Application Development

ca World '15

Implementing Adaptable Microservices; A Methodology for Loosely-Coupled Components

Mike Amundsen
CA Technologies
Dir. API Architecture, API Academy
@mamund

#CAWorld

Session: DO3X96S

For Informational Purposes Only

Terms of this Presentation

The content provided in this CA World 2015 presentation is intended for informational purposes only and does not form any type of warranty. The information provided by a CA partner and/or CA customer has not been reviewed for accuracy by CA.

Agenda

1. ADAPTABILITY
2. LOOSELY-COUPLED
3. D.O.R.R.
4. CONWAY'S LAW
5. SUMMARY
6. Q & A

Recent Tech Headlines…

- “How Etsy Deploys More Than 50 Times a Day”
  - Joao Miranda - InfoQ, March 2014
- “Netflix … deploys a hundred times per day”
  - Zef Hemel – InfoQ, June 2013
- “How We Deploy 300 Times a Day”
  - Zack Bloom, Hubspot blog, November 2013

What's going on here?

“Software at the Speed of DevOps”

CIO Journal, March 2014

Yep – DevOps

Yep – DevOps
But for CODE

“The Three Ways: The Principles Underpinning DevOps”

By Gene Kim

- The First Way: Systems Thinking
- The Second Way: Amplify Feedback Loops
- The Third Way: Culture of Continual Experimentation and Learning

Adaptability

Adaptability

"To make suitable to requirements or conditions; adjust or modify fittingly"

Adaptability

*"To make suitable to **requirements** or **conditions**; adjust or modify fittingly"*

Adaptability

*"To make **suitable** to requirements or conditions; adjust or modify **fittingly**"*

Evolution

“The gradual development of something, especially from simple to a more complex form”

Refactoring

"The process of restructuring existing computer code without changing its external behavior"

Refactoring

"The process of restructuring existing computer code ***without changing its external behavior****"*

Adaptability
is a
System Property

Adaptability is a System Property

Desired Property

Loosely-Coupled

Loosely-Coupled

"A system in which each of its components has little or no knowledge of the [internal] definitions of other separate components"

Loosely-Coupled

"Program to an INTERFACE, not an implementation." – GoF, 1994

Loosely-Coupled

Your system is NOT loosely-coupled if deploying Component-A means you MUST also deploy Component-B.

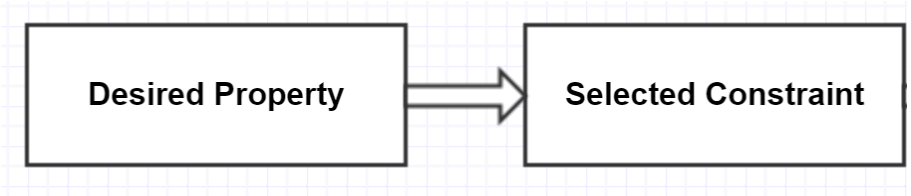
Loosely-Coupled

“Embrace independent evolvability.”

- Darrel Miller, Microsoft

Loosely-Coupled
is a
Constraint

Loosely-Coupled is a Constraint



D. O. R. R.

D. O. R. R.

- Data Model (storage)
- Object Model (functionality)
- Resource Model (interface)
- Representation Model (message)

D. O. R. R.

Data Model

```
function main(object, action, arg1, arg2, arg3) {
  var rtn;

  switch(action) {
    case 'list':
      rtn = getList(object);
      break;
    case 'filter':
      rtn = getList(object, arg1);
      break;
    case 'item':
      rtn = getItem(object, arg1);
      break;
    case 'add':
      rtn = addItem(object, arg1, arg2);
      break;
    case 'update':
      rtn = updateItem(object, arg1, arg2, arg3);
      break;
    case 'remove':
      rtn = removeItem(object, arg1);
      break;
    default:
      rtn = null;
      break;
  }
  return rtn;
```

D. O. R. R.

Object Model

```
exports.task = function(action, args1, args2, args
  var object, rtn;

  object = 'task';
  rtn = null;

  switch(action) {
    case 'list':
      rtn = loadList(storage(object, 'list'), obje
      break;
    case 'read':
      rtn = loadList(storage(object, 'item', args1
      break;
    case 'filter':
      rtn = loadList(storage(object, 'filter', arg
      break;
    case 'add':
      rtn = loadList(storage(object, 'add', args1)
      break;
    case 'update':
      rtn = loadList(storage(object, 'update', arg
    default:
      rtn = null;
  }

  return rtn;
}
```

D. O. R. R.

Resource Model

```
switch(req.method) {
  case 'GET':
    if(parts[1] && parts[1].indexOf('?')===-1) {
      switch (parts[1]) {
        case "complete":
          sendCompleteForm(req, res, parts[2], res
          break;
        case "assign":
          sendAssignUserForm(req, res, parts[2], r
          break;
        case "add":
          sendAddTaskForm(req, res, respond);
          break;
        case "all":
        case "bycategory":
        case "bytitle":
        case "bycomplete":
          sendList(req, res, respond, parts[1]);
          break;
        default:
          sendItem(req, res, parts[1], respond);
          break;
      }
    }
    else {
```

D. O. R. R.

Representation Model

```
27   // dispatch to requested represen
28   switch(mimeType.toLowerCase()) {
29     case "application/json":
30       doc = json(object, root);
31       break;
32     case "application/vnd.collectio
33       doc = cj(object, root);
34       break;
35     case "application/vnd.amundsen.
36       doc = uberjson(object, root);
37       break;
38     case "text/html":
39     case "application/html":
40     default:
41       doc = html(object, root);
42       break;
43   }
44
45   return doc;
```

D. O. R. R.

“Embrace Independent Evolvability”

```
exports.task = function(action, args1, args2, args3) {
  var object, rtn;

  object = 'task';
  rtn = null;

  switch(action) {
    case 'list':
      rtn = loadList(storage(object, 'list'), object);
      break;
    case 'read':
      rtn = loadList(storage(object, 'item', args1), object)
      break;
    case 'filter':
      rtn = loadList(storage(object, 'filter', args1), objec
      break;
    case 'add':
      rtn = loadList(storage(object, 'add', args1), object);
      break;
    case 'update':
      rtn = loadList(storage(object, 'update', args1, args2,
    default:
      rtn = null;
  }

  return rtn;
```

D. O. R. R.

“Embrace Independent Evolvability”

```
exports.task = function(action, args1, args2, args3) {
  var object, rtn;

  object = 'task';
  rtn = null;

  switch(action) {
    case 'list':
      rtn = loadList(storage(object, 'lis
      break;
    case 'read':
      rtn = loadList(storage(object, 'ite
      break;
    case 'filter':
      rtn = loadList(storage(object, 'fil
      break;
    case 'add':
      rtn = loadList(storage(object, 'add
      break;
    case 'update':
      rtn = loadList(storage(object, 'upd
    default:
      rtn = null;
  }

  return rtn;
```

```
switch(req.method) {
  case 'GET':
    if(parts[1] && parts[1].indexOf('?')===-1) {
      switch (parts[1]) {
        case "complete":
          sendCompleteForm(req, res, parts[2], respond);
          break;
        case "assign":
          sendAssignUserForm(req, res, parts[2], respond);
          break;
        case "add":
          sendAddTaskForm(req, res, respond);
          break;
        case "all":
        case "bycategory":
        case "bytitle":
        case "bycomplete":
          sendList(req, res, respond, parts[1]);
          break;
        default:
          sendItem(req, res, parts[1], respond);
```

D. O. R. R.

“Embrace Independent Evolvability”

```
exports.task = function(action, args1, args2, args3) {
  var object, rtn;

  object = 'task';
  rtn = null;

  switch(action) {
    case 'list':
      rtn = loadList(storage(object, 'lis
      break;
    case 'read':
      rtn = loadList(storage(object, 'ite
      break;
    case 'filter':
      rtn = loadList(storage(object, 'fil
      break;
    case 'add':
      rtn = loadList(storage(object, 'add
      break;
    case 'update':
      rtn = loadList(storage(object, 'upd
    default:
      rtn = null;
  }

  return rtn;
```

```
switch(req.method) {
  case 'GET':
    if(parts[1] && parts[1].indexOf('?')===
      switch (parts[1]) {
        case "complete":
          sendCompleteForm(req, res, parts[
          break;
        case "assign":
          sendAssignUserForm(req, res, part
          break;
        case "add":
          sendAddTaskForm(req, res, respond
          break;
        case "all":
        case "bycategory":
        case "bytitle":
        case "bycomplete":
          sendList(req, res, respond, parts[1]);
          break;
        default:
          sendItem(req, res, parts[1], respond);
```

```
// dispatch to requested representor
switch(mimeType.toLowerCase()) {
  case "application/json":
    doc = json(object, root);
    break;
  case "application/vnd.collection+json":
    doc = cj(object, root);
    break;
  case "application/vnd.amundsen.uber+json":
    doc = uberjson(object, root);
    break;
  case "text/html":
  case "application/html":
  default:
    doc = html(object, root);
    break;
}
```

D. O. R. R.
is a
Best Practice

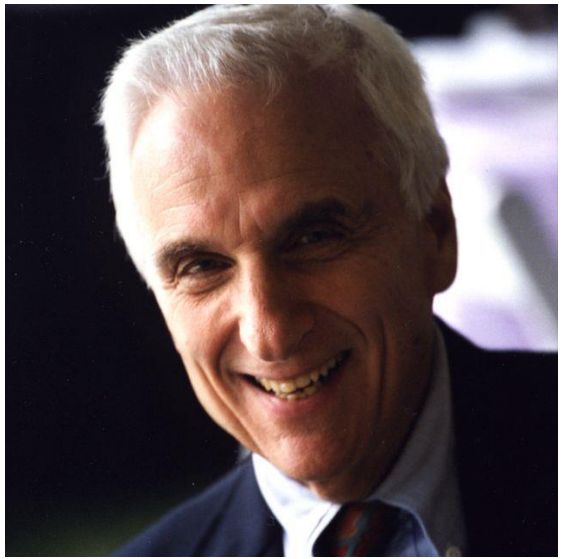
D. O. R. R.
is a
Best Practice

Conway's Law

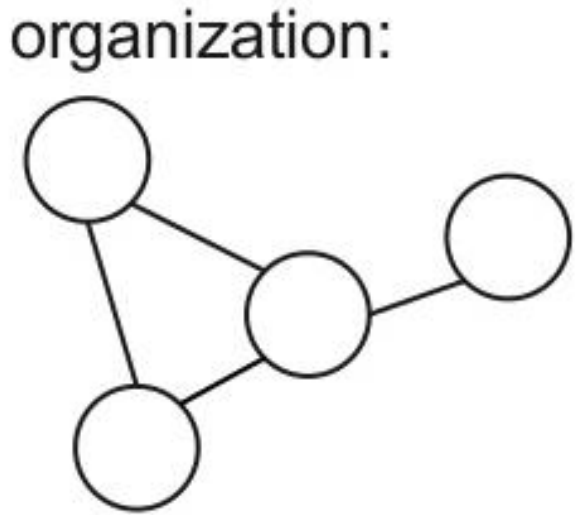
Conway's Law

"Organizations produce systems which are copies of their communication structures."

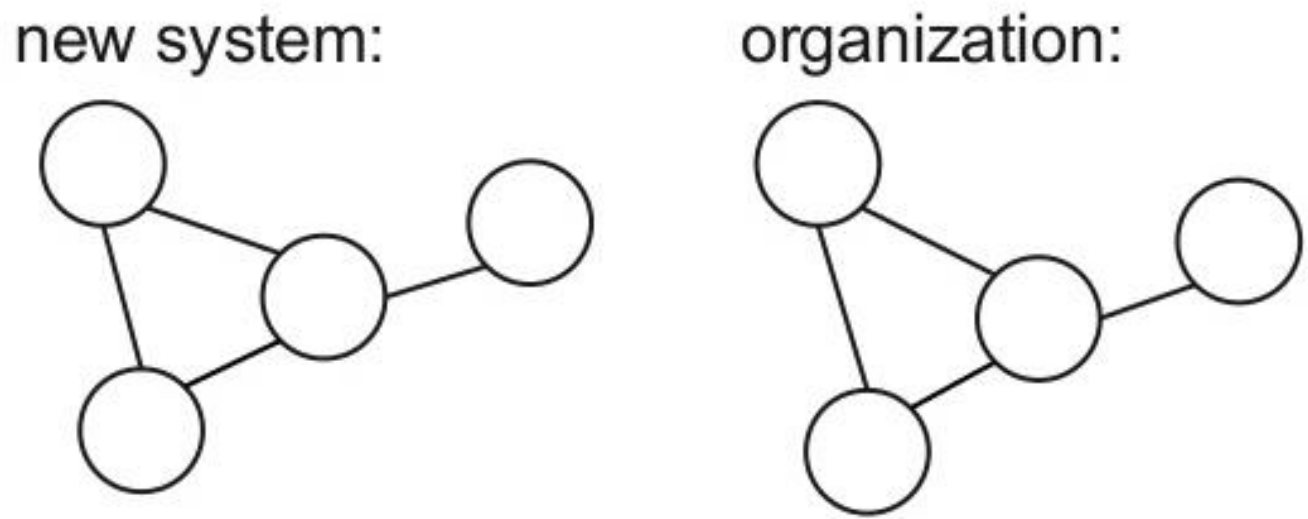
– Mel Conway, 1968



Conway’s Law



Conway’s Law



Conway's Law

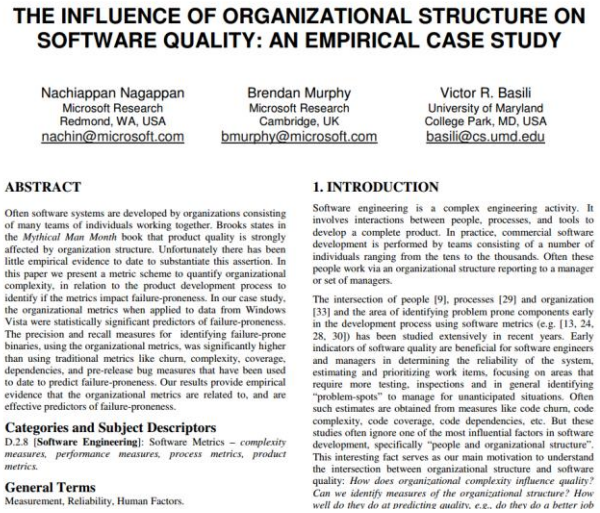
"If you have four teams working on a compiler, you get a four-pass complier."

– Eric S. Raymond

Conway's Law

"Organizational metrics can predict software failure-proneness with a precision and recall of 85 percent."

– Microsoft Research



THE INFLUENCE OF ORGANIZATIONAL STRUCTURE ON SOFTWARE QUALITY: AN EMPIRICAL CASE STUDY

Nachiappan Nagappan
Microsoft Research
Redmond, WA, USA
nachin@microsoft.com

Brendan Murphy
Microsoft Research
Cambridge, UK
bmurphy@microsoft.com

Victor R. Basili
University of Maryland
College Park, MD, USA
basili@cs.umd.edu

ABSTRACT

Often software systems are developed by organizations consisting of many teams of individuals working together. Brooks states in the *Mythical Man Month* book that product quality is strongly affected by organization structure. Unfortunately there has been little empirical evidence to date to substantiate this assertion. In this paper we present a metric scheme to quantify organizational complexity, in relation to the product development process to identify if the metrics impact failure-proneness. In our case study, the organizational metrics when applied to data from Windows Vista were statistically significant predictors of failure-proneness. The precision and recall measures for identifying failure-prone binaries, using the organizational metrics, was significantly higher than using traditional metrics like churn, complexity, coverage, dependencies, and pre-release bug measures that have been used to date to predict failure-proneness. Our results provide empirical evidence that the organizational metrics are related to, and are effective predictors of failure-proneness.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Software Metrics – *complexity measures, performance measures, process metrics, product metrics.*

General Terms

Measurement, Reliability, Human Factors.

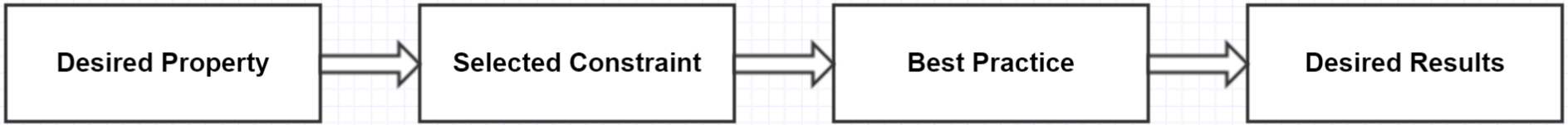
1. INTRODUCTION

Software engineering is a complex engineering activity. It involves interactions between people, processes, and tools to develop a complete product. In practice, commercial software development is performed by teams consisting of a number of individuals ranging from the tens to the thousands. Often these people work via an organizational structure reporting to a manager or set of managers.

The intersection of people [9], processes [29] and organization [33] and the area of identifying problem prone components early in the development process using software metrics (e.g. [13, 24, 28, 30]) has been studied extensively in recent years. Early indicators of software quality are beneficial for software engineers and managers in determining the reliability of the system, estimating and prioritizing work items, focusing on areas that require more testing, inspections and in general identifying "problem-spots" to manage for unanticipated situations. Often such estimates are obtained from measures like code churn, code complexity, code coverage, code dependencies, etc. But these studies often ignore one of the most influential factors in software development, specifically "people and organizational structure". This interesting fact serves as our main motivation to understand the intersection between organizational structure and software quality: *How does organizational complexity influence quality? Can we identify measures of the organizational structure? How well do they do at predicting quality, e.g., do they do a better job*

Conway's Law
is
Inevitable

Summary



Continuous Deployment

Every day you don't release to production is another day you risk falling behind.

Continuous Change

“The only thing that is constant is change.”

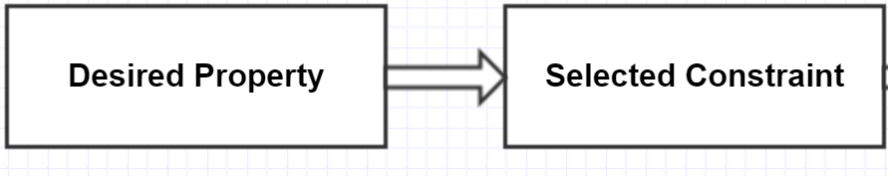
- Heraclitus

Summary

1 **Adaptability as a System Property**

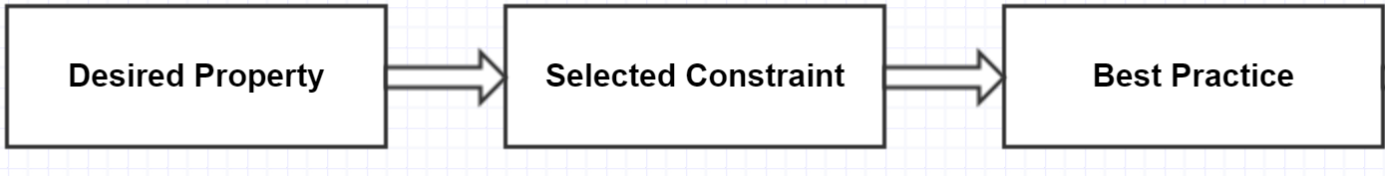
Summary

1. Adaptability as a System Property
2. Loosely-Coupled as a Constraint



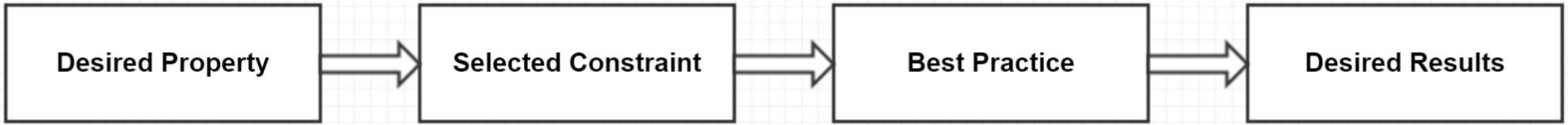
Summary

1. Adaptability as a System Property
2. Loosely-Coupled as a Constraint
3. D.O.R.R. as a Best Practice



Summary

1. Adaptability as a System Property
2. Loosely-Coupled as a Constraint
3. D.O.R.R. as a Best Practice
4. Conway's Law is Inevitable



Future of Change

"Those who ignore the mistakes of the future are bound to make them."

- Dr. Joseph Miller

Q & A

DevOps: API Management and Application Development

ca World '15

Implementing Adaptable Microservices; A Methodology for Loosely-Coupled Components

Mike Amundsen
CA Technologies
Dir. API Architecture, API Academy

@mamund
#CAWorld

Session: DO3X96S