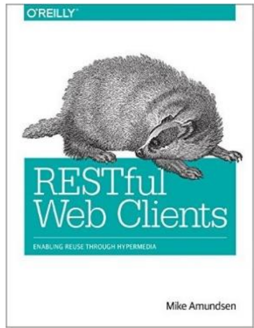
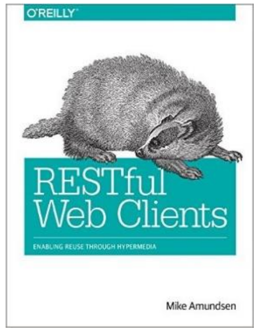
# DORR Architecture for Hypermedia Services

Mike Amundsen
API Academy
@mamund

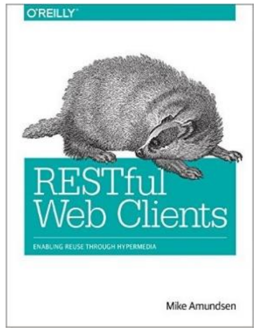


## Highlights

- Evolvability
- Loosely Coupled
- DORR Pattern
- Conway



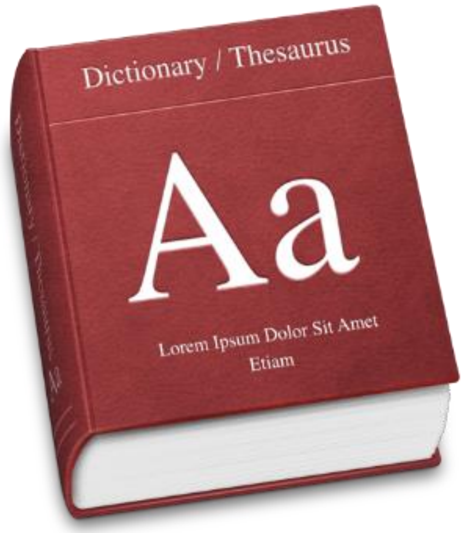
# Evolvability



## Evolvability

**evolution:**

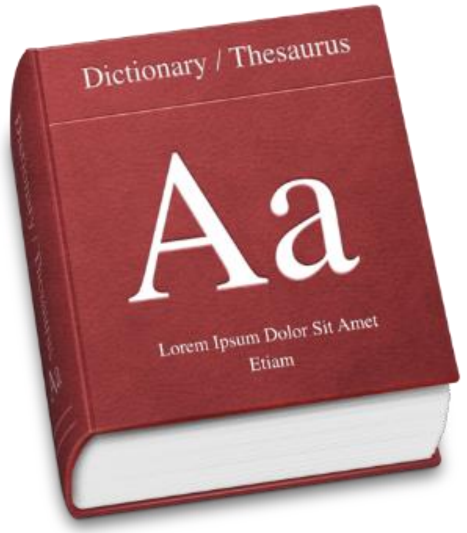
*“the gradual development of something, especially from simple to a more complex form”*



## Evolvability

**evolution:**

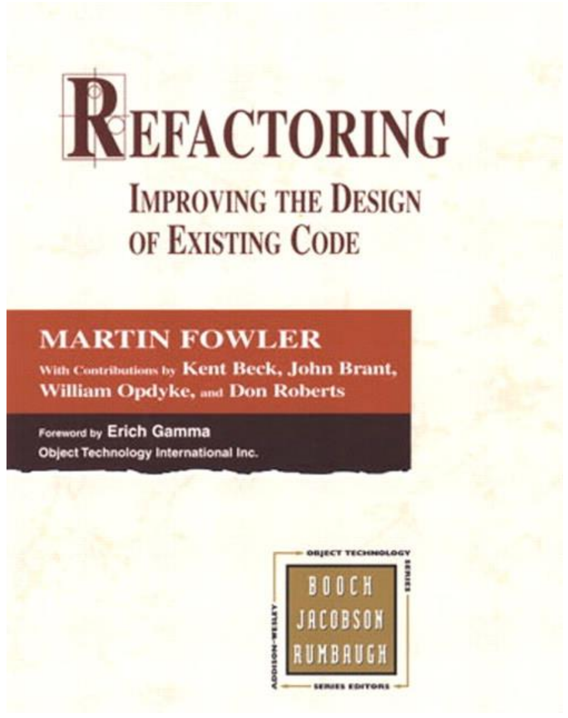
*“the **gradual** development of something, especially from simple to a more complex form”*



## Evolvability

**Refactoring:**

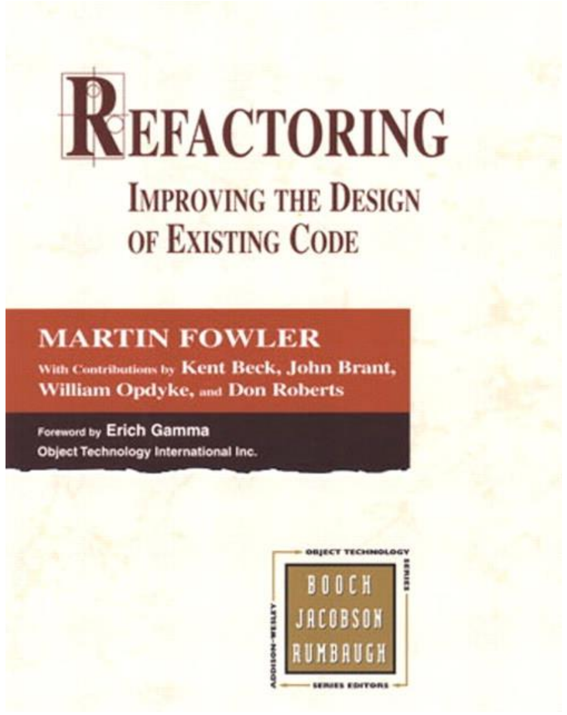
*"the process of restructuring existing computer code without changing its external behavior."*


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH JACOBSON RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

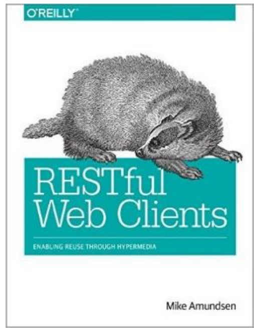
## Evolvability

**Refactoring:**

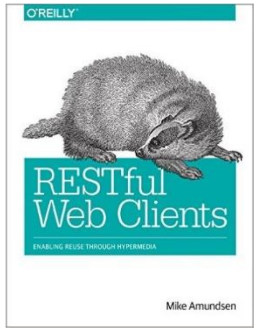
*"the process of restructuring existing computer code* ***without changing its external behavior.****"*


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
BOOCH JACOBSON RUMBAUGH

# Evolvability
# is a
# *System Property*

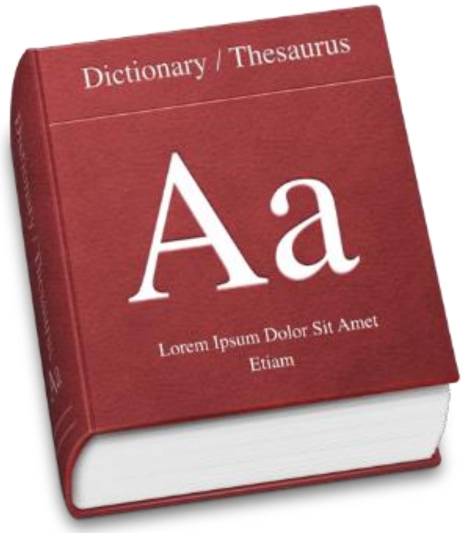


## Loosely-Coupled



## Loosely-Coupled

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

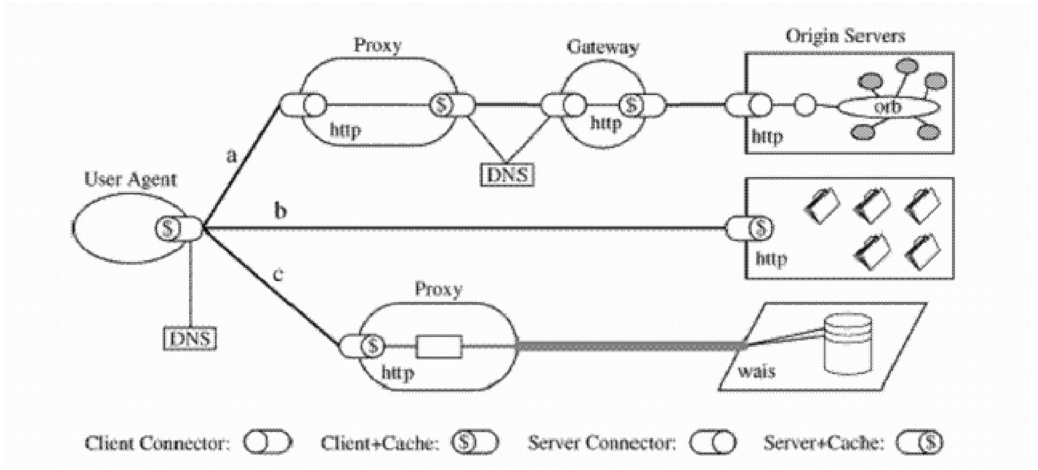


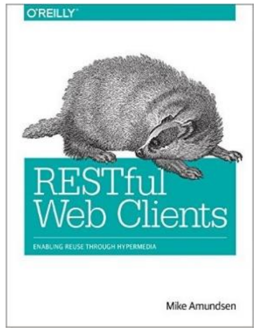
## Loosely-Coupled

*“Program to an interface, not an implementation.” -GoF*

## Loosely-Coupled

*"Your system is **not** loosely-coupled if deploying component A means you MUST also deploy component B."*

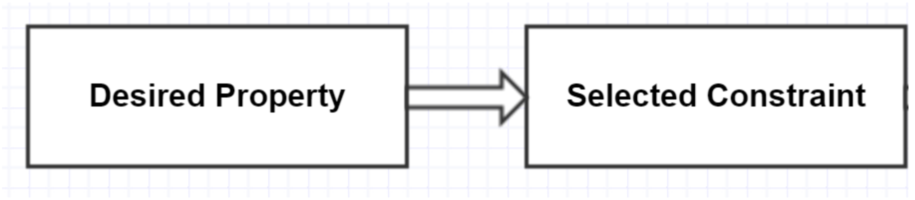


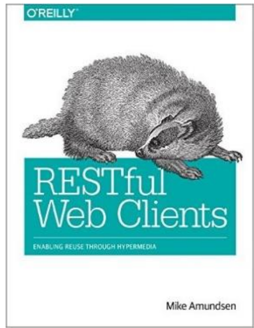


## Loosely-Coupled

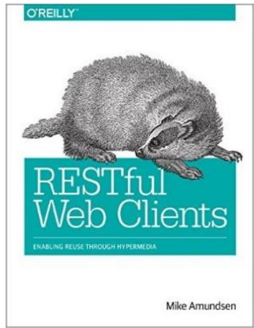
*“Embrace independent evolvability.”*
*– Darrel Miller, Runscope*

# Loosely-Coupled
# as a
# *Constraint*



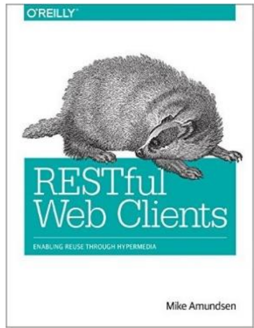


# D.O.R.R.


O'REILLY
RESTful Web Clients
ENABLING REUSE THROUGH HYPERMEDIA
Mike Amundsen

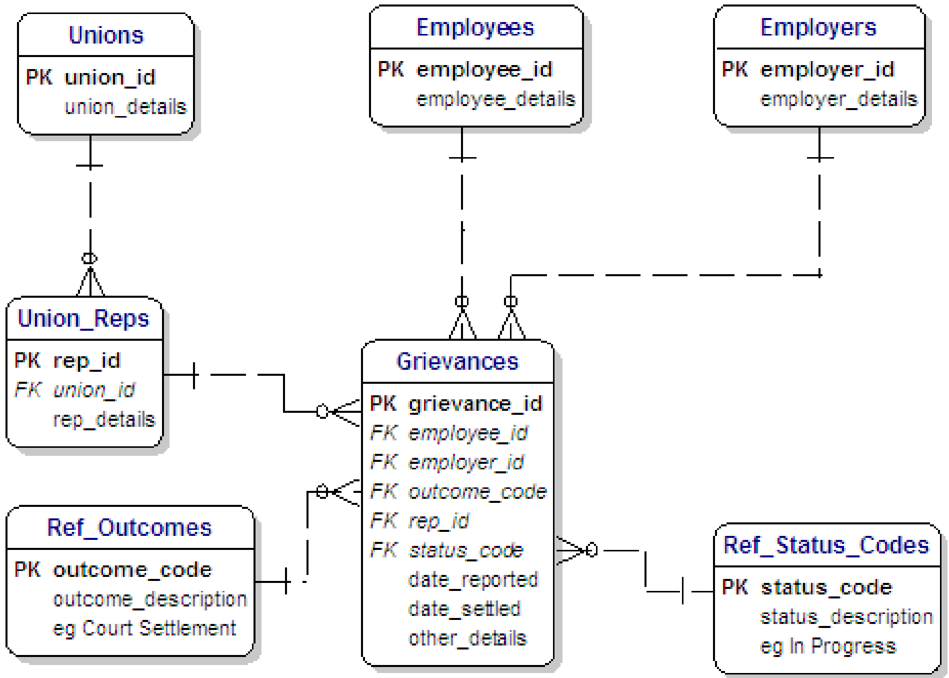
## D.O.R.R

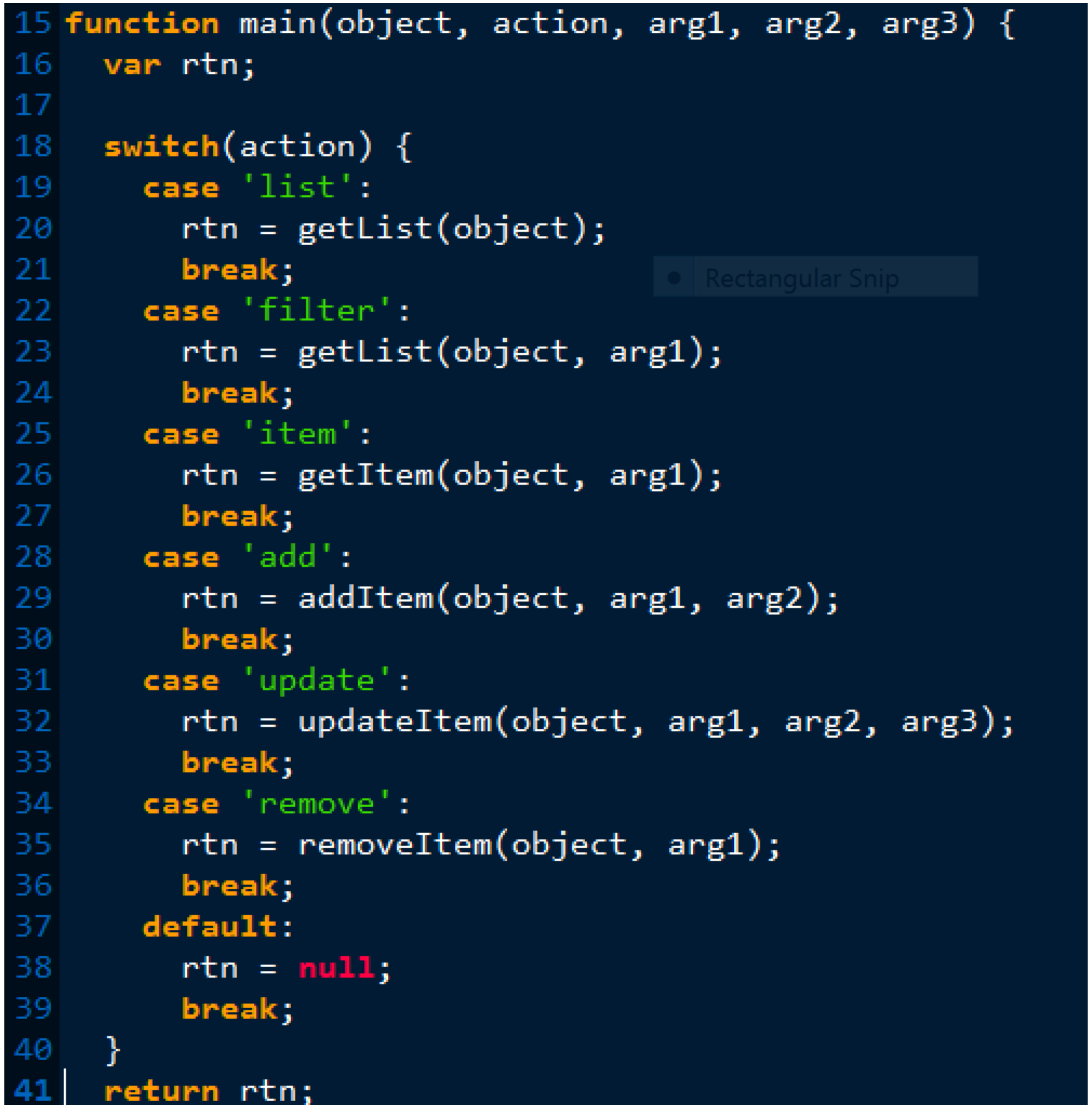
- Data Model (storage)
- Object Model (functionality)
- Resource Model (interface)
- Representation Model (message)



# D.O.R.R

## *"Data Model"*

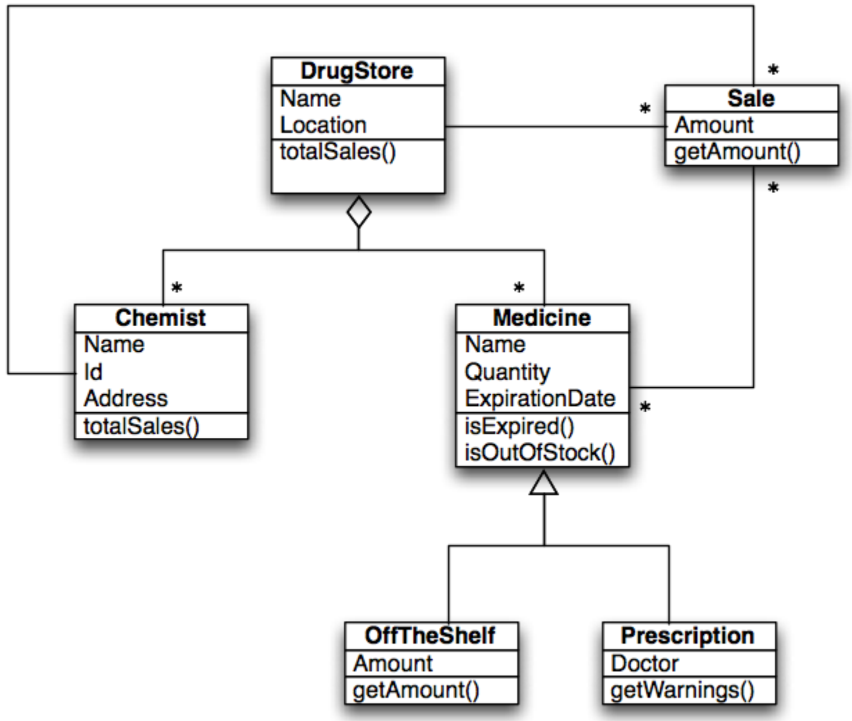


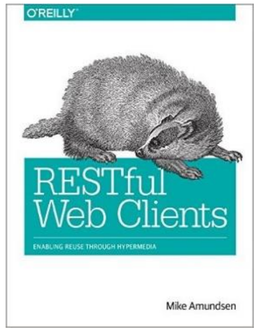


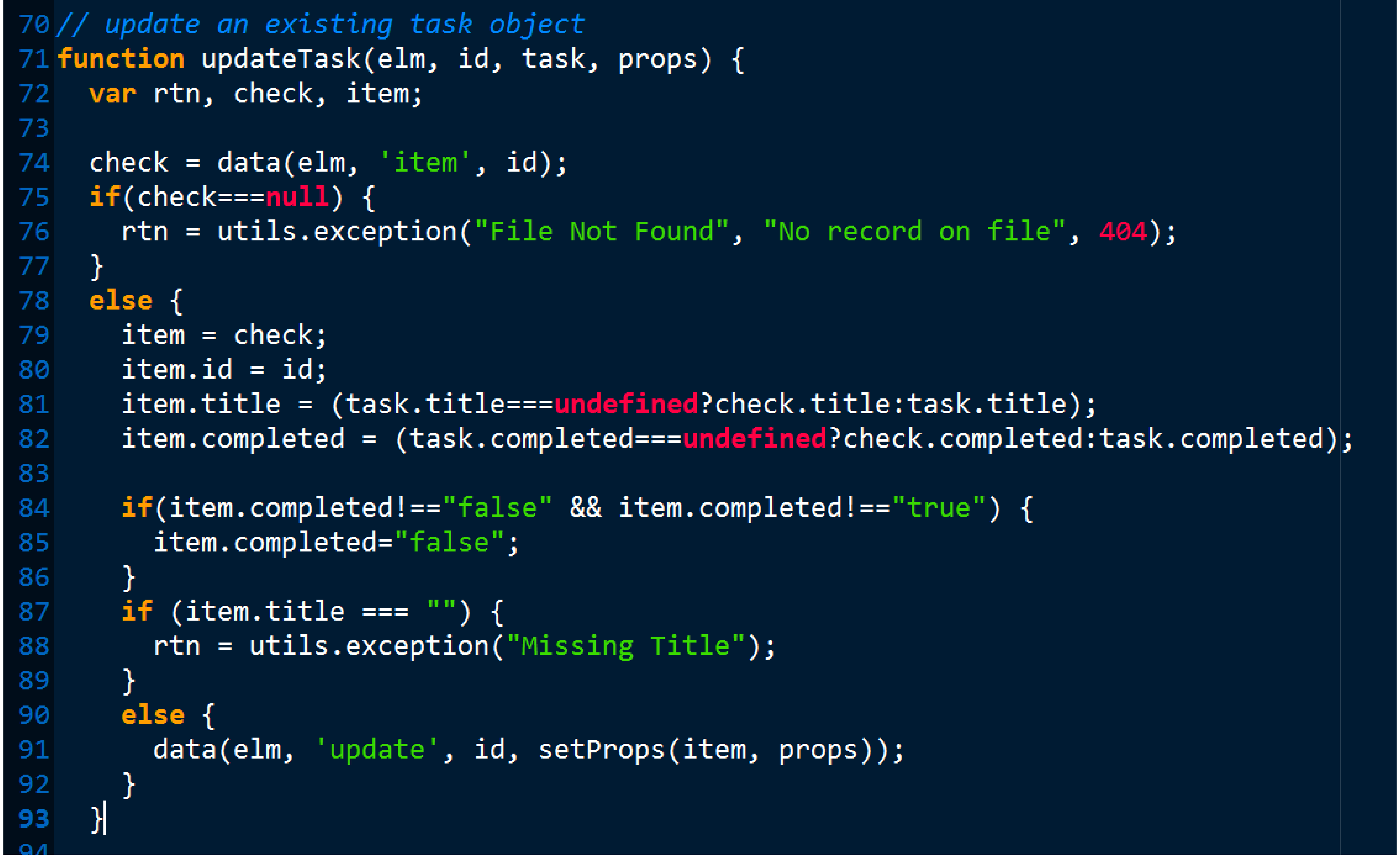
```javascript
function main(object, action, arg1, arg2, arg3) {
  var rtn;

  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"*



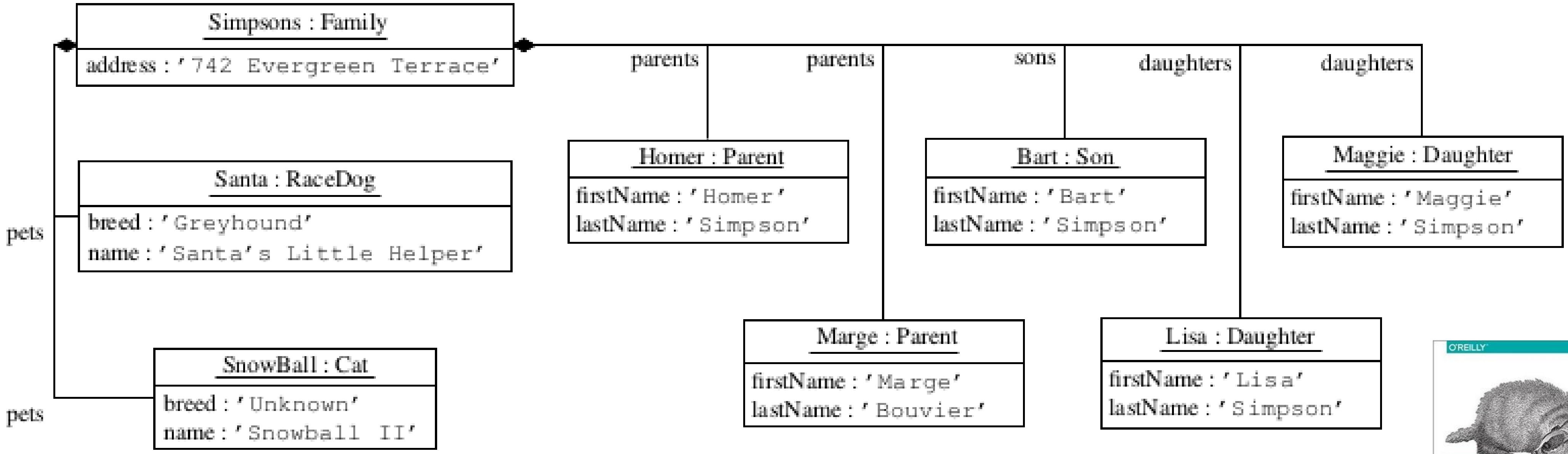


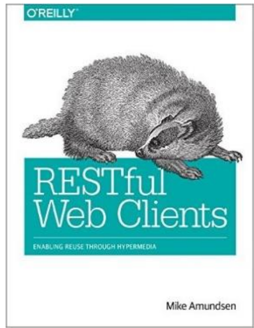


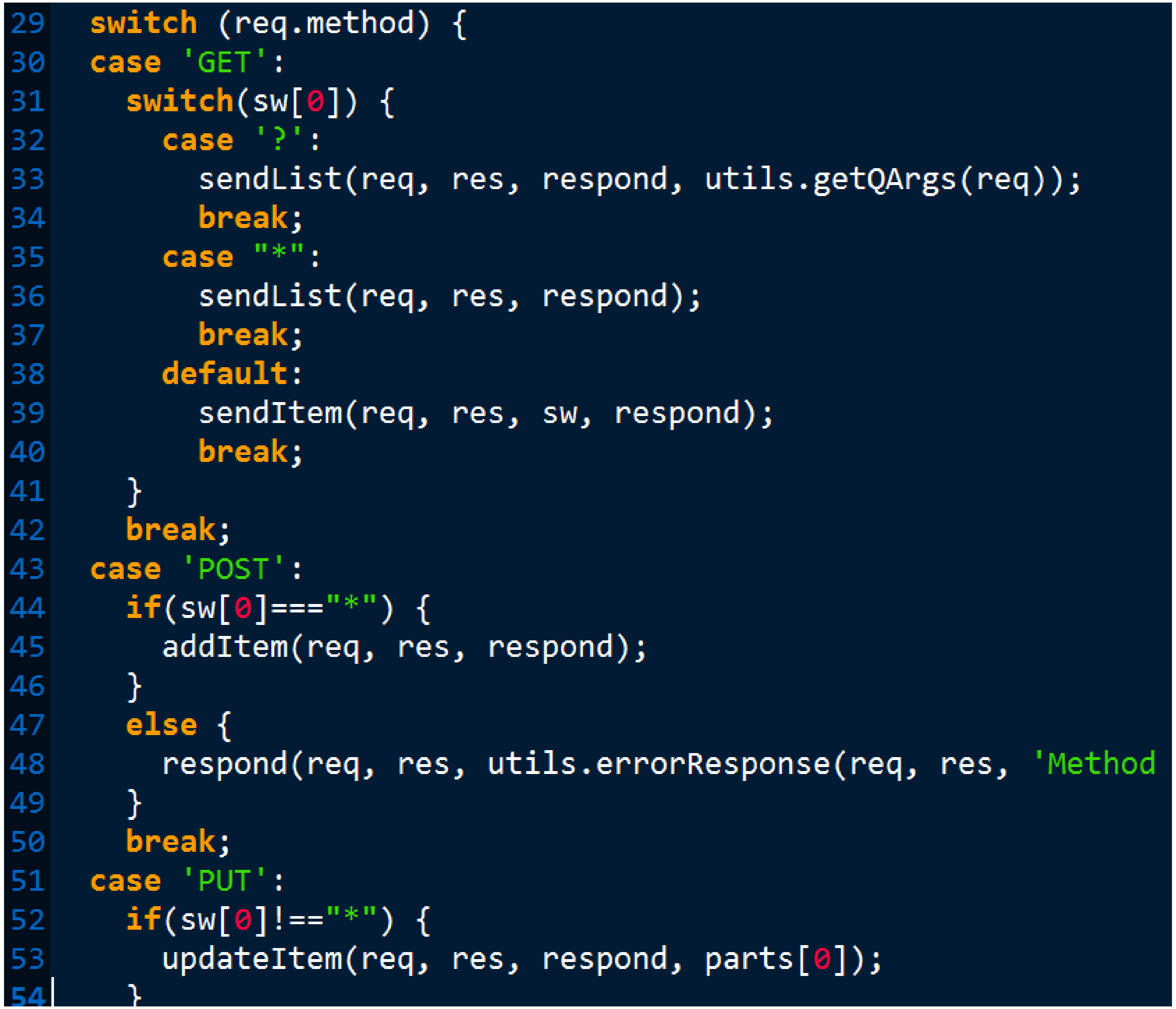
```
// update an existing task object
function updateTask(elm, id, task, props) {
  var rtn, check, item;

  check = data(elm, 'item', id);
  if(check===null) {
    rtn = utils.exception("File Not Found", "No record on file", 404);
  }
  else {
    item = check;
    item.id = id;
    item.title = (task.title===undefined?check.title:task.title);
    item.completed = (task.completed===undefined?check.completed:task.completed);

    if(item.completed!=="false" && item.completed!=="true") {
      item.completed="false";
    }
    if (item.title === "") {
      rtn = utils.exception("Missing Title");
    }
    else {
      data(elm, 'update', id, setProps(item, props));
    }
  }
```

# D.O.R.R

## *"Resource Model"*



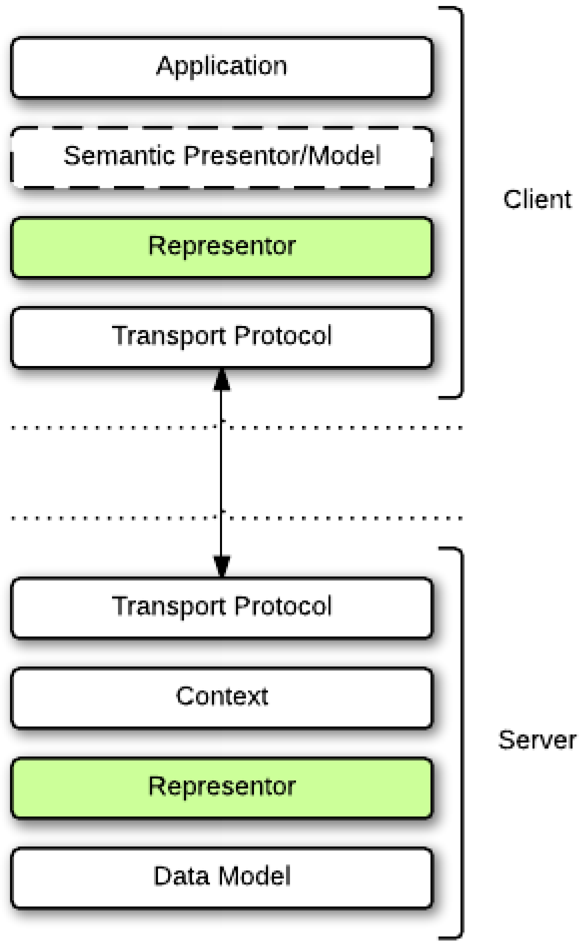


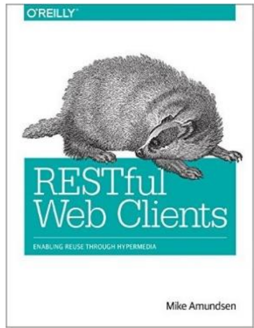


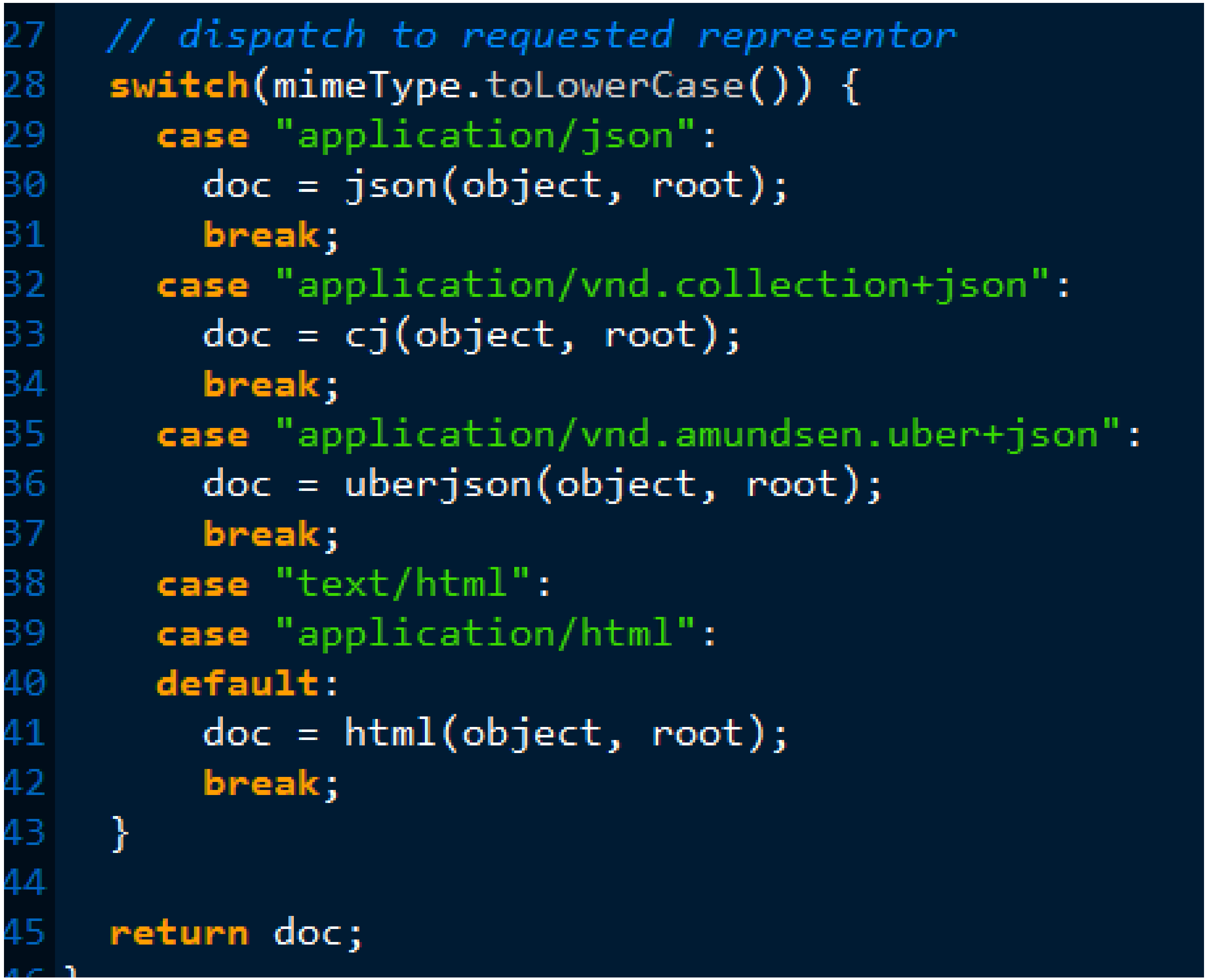
```
switch (req.method) {
case 'GET':
  switch(sw[0]) {
    case '?':
      sendList(req, res, respond, utils.getQArgs(req));
      break;
    case "*":
      sendList(req, res, respond);
      break;
    default:
      sendItem(req, res, sw, respond);
      break;
  }
  break;
case 'POST':
  if(sw[0]==="*") {
    addItem(req, res, respond);
  }
  else {
    respond(req, res, utils.errorResponse(req, res, 'Method
  }
  break;
case 'PUT':
  if(sw[0]!=="*") {
    updateItem(req, res, respond, parts[0]);
  }
```

# D.O.R.R

*“Representor Model”*



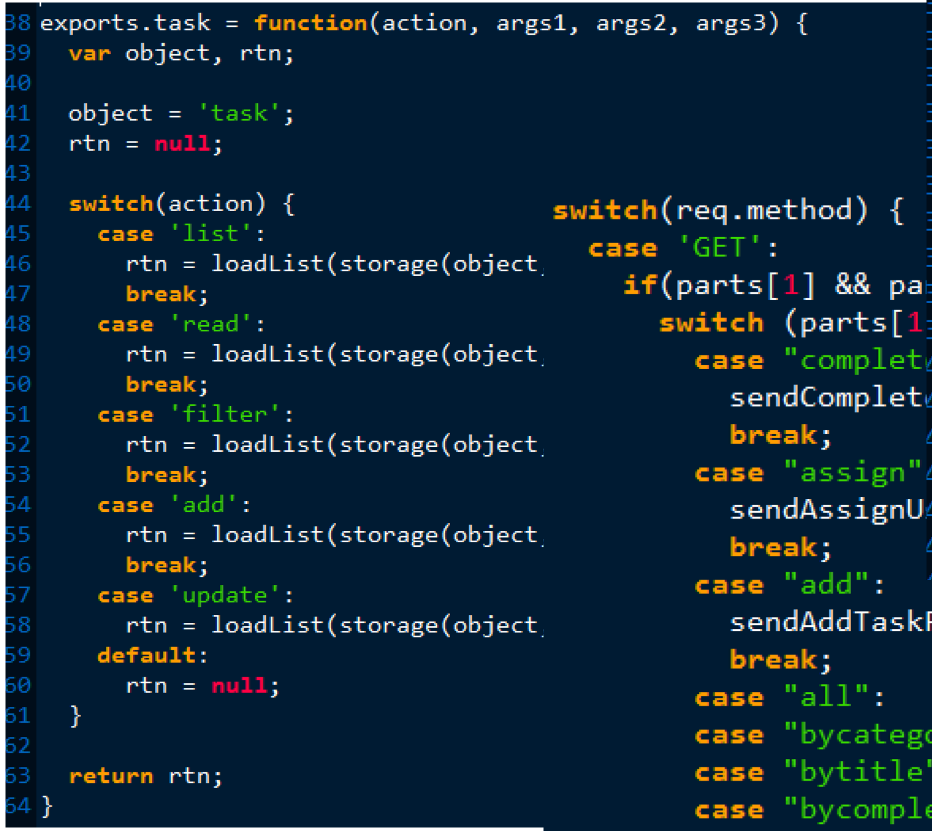




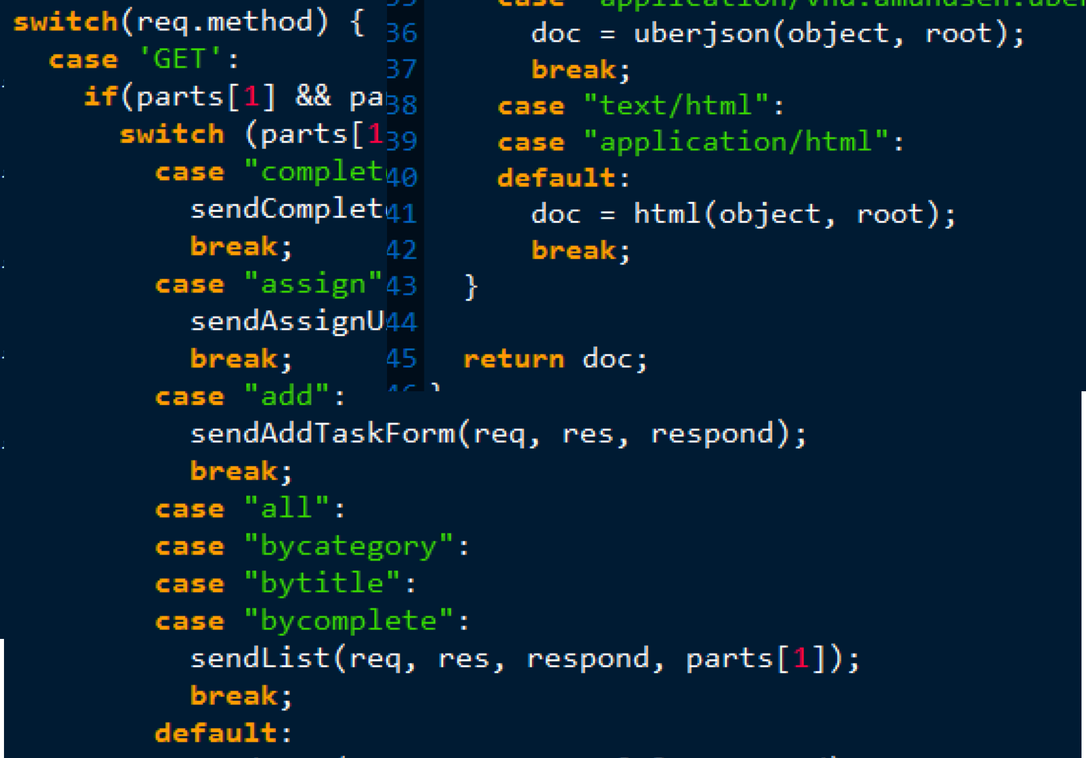
```
  // dispatch to requested representor
  switch(mimeType.toLowerCase()) {
    case "application/json":
      doc = json(object, root);
      break;
    case "application/vnd.collection+json":
      doc = cj(object, root);
      break;
    case "application/vnd.amundsen.uber+json":
      doc = uberjson(object,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

  return doc;
}
```

# D.O.R.R

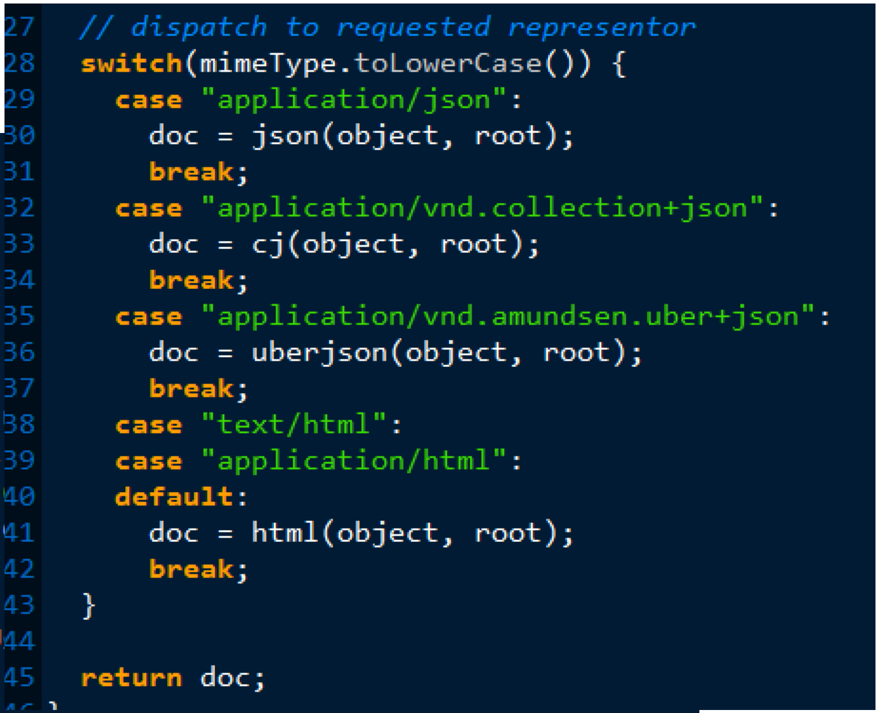
*“Embrace independent evolvability.”*



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

  object = 'task';
  rtn = null;

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

  return rtn;
}
```

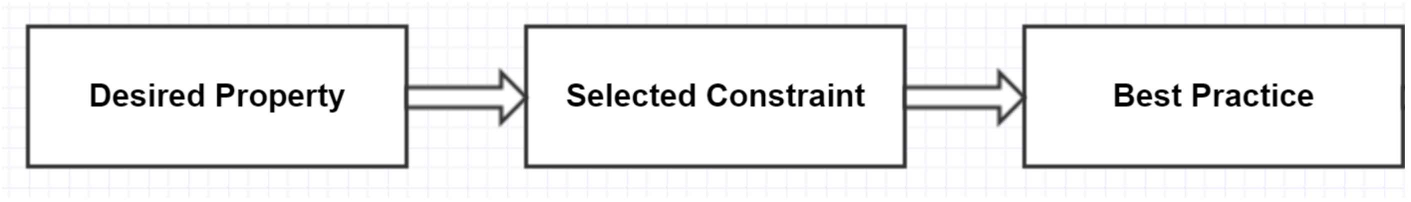


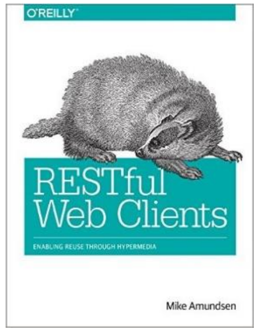
```
switch(req.method) {
  case 'GET':
    if(parts[1] && pa
      switch (parts[1
        case "complet
          sendComplet
          break;
        case "assign"
          sendAssignU
          break;
        case "add":
          sendAddTaskForm(req, res, respond);
          break;
        case "all":
        case "bycategory":
        case "bytitle":
        case "bycomplete":
          sendList(req, res, respond, parts[1]);
          break;
        default:
```



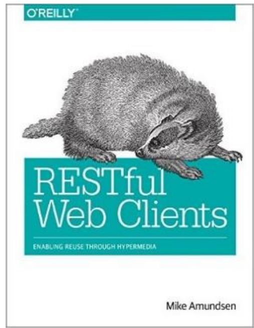
```
  // dispatch to requested representor
  switch(mimeType.toLowerCase()) {
    case "application/json":
      doc = json(object, root);
      break;
    case "application/vnd.collection+json":
      doc = cj(object, root);
      break;
    case "application/vnd.amundsen.uber+json":
      doc = uberjson(object,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

  return doc;
```

# D.O.R.R.
# as a
# *Best Practice*



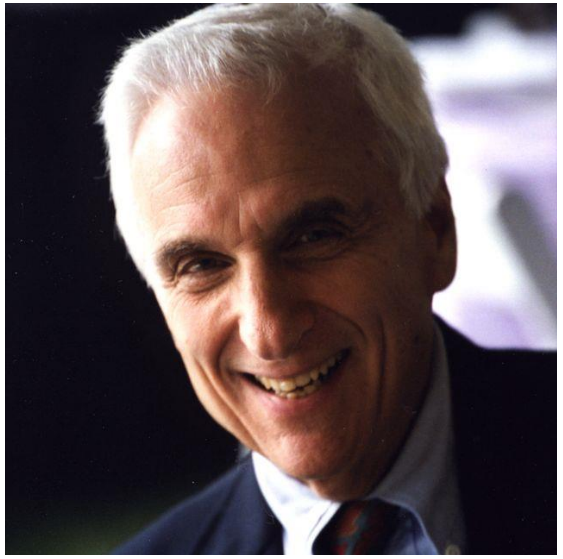


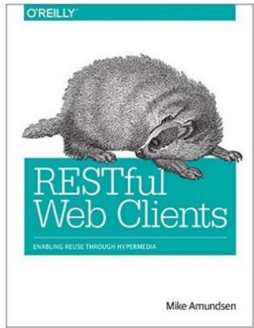
## Conway's Law


O'REILLY
RESTful Web Clients
ENABLING REUSE THROUGH HYPERMEDIA
Mike Amundsen

## Conway's Law

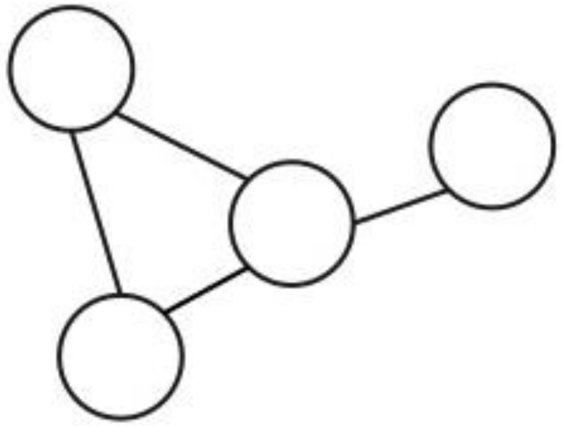
*"Organizations produce systems which are copies of their communication structures." – Mel Conway, 1968*

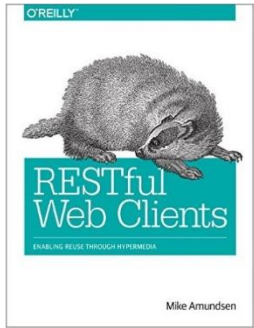




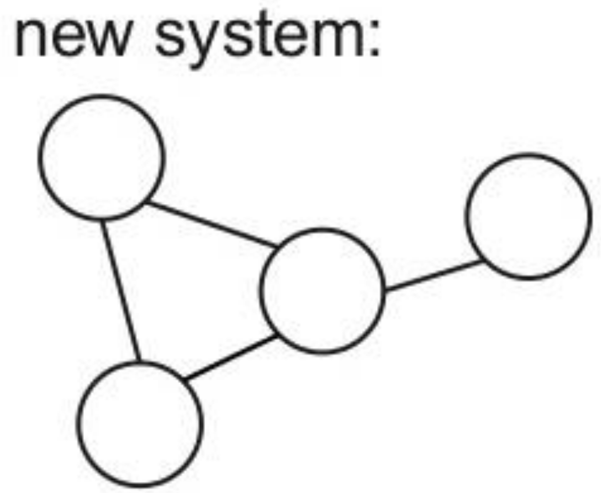
# Conway’s Law

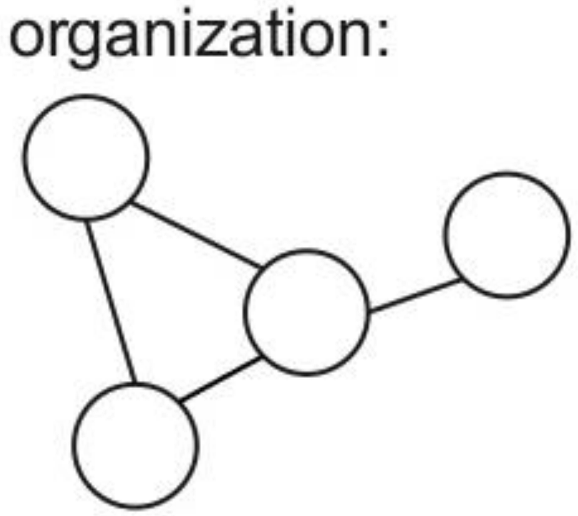
organization:

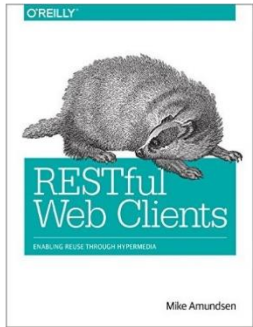




## Conway's Law

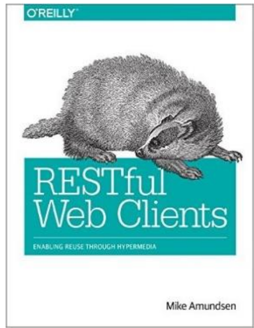

new system:


organization:


O'REILLY
RESTful Web Clients
ENABLING REUSE THROUGH HYPERMEDIA
Mike Amundsen

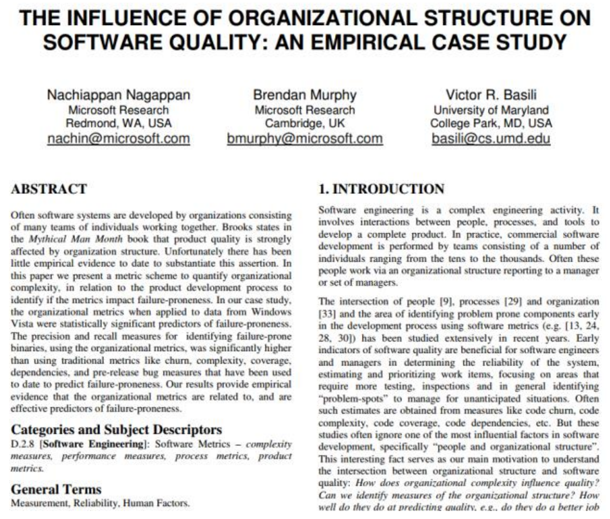
## Conway’s Law

*“If you have four teams working on a compiler project, you gat a four-pass compiler” – 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.

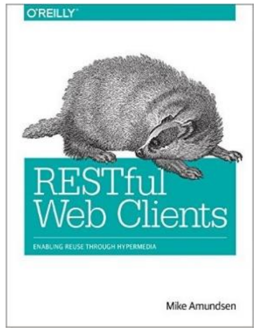
**Categories and Subject Descriptors**
D.2.8 [**Software Engineering**]: Software Metrics – *complexity measures, performance measures, process metrics, product metrics.*

**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*



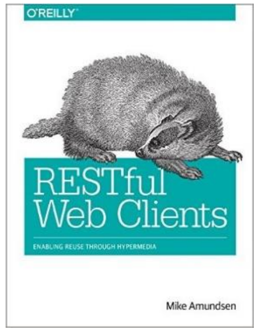
O'REILLY

RESTful Web Clients

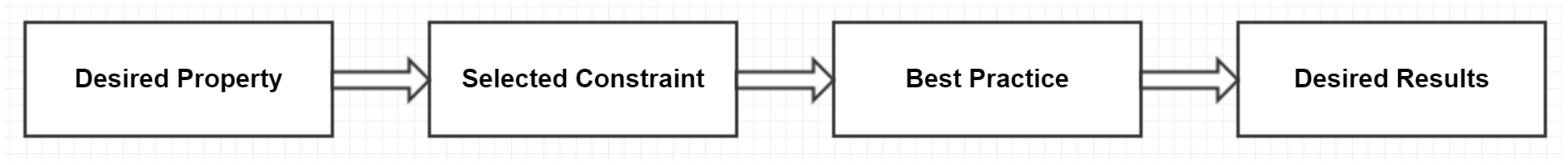
ENABLING REUSE THROUGH HYPERMEDIA

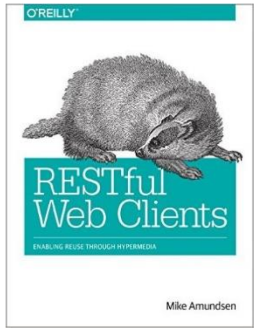
Mike Amundsen

## Conway's Law
## is
## *Inevitable*



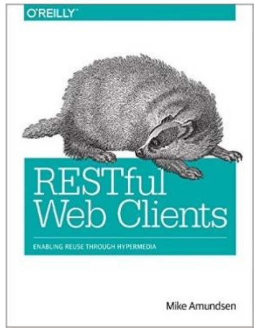
# Summary


Desired Property
Selected Constraint
Best Practice
Desired Results


O'REILLY
RESTful Web Clients
ENABLING REUSE THROUGH HYPERMEDIA
Mike Amundsen

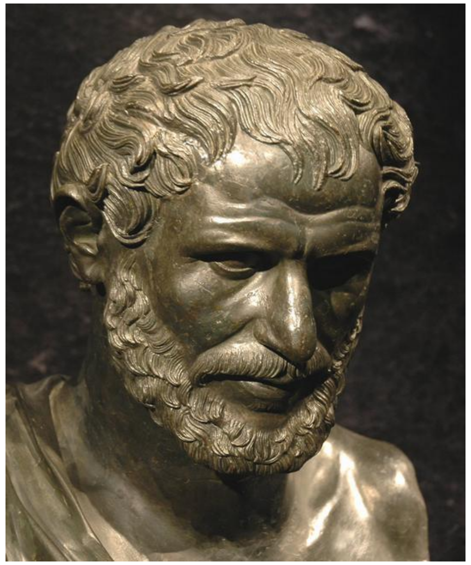
## Continuous Deployment

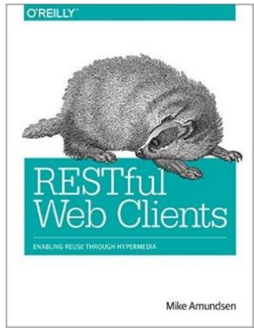
*"Every day you don't release to production is another day you risk falling behind."*



## Change

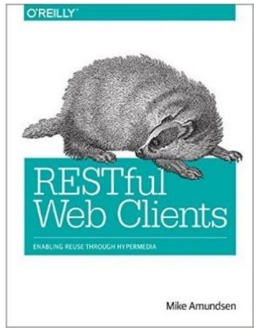
*“The only thing that is constant is change.” -- Heraclitus*





## Highlights

- Evolvability
- Loosely Coupled
- DORR Pattern
- Conway



# DORR Architecture for Hypermedia Services

Mike Amundsen
API Academy
@mamund