Why not duplicate?

Designing and Maintaining Software (DAMS)

Louis Rose

Habitable Software

Leaner

Less **Complex**

Loosely **Coupled**

More **Cohesive**

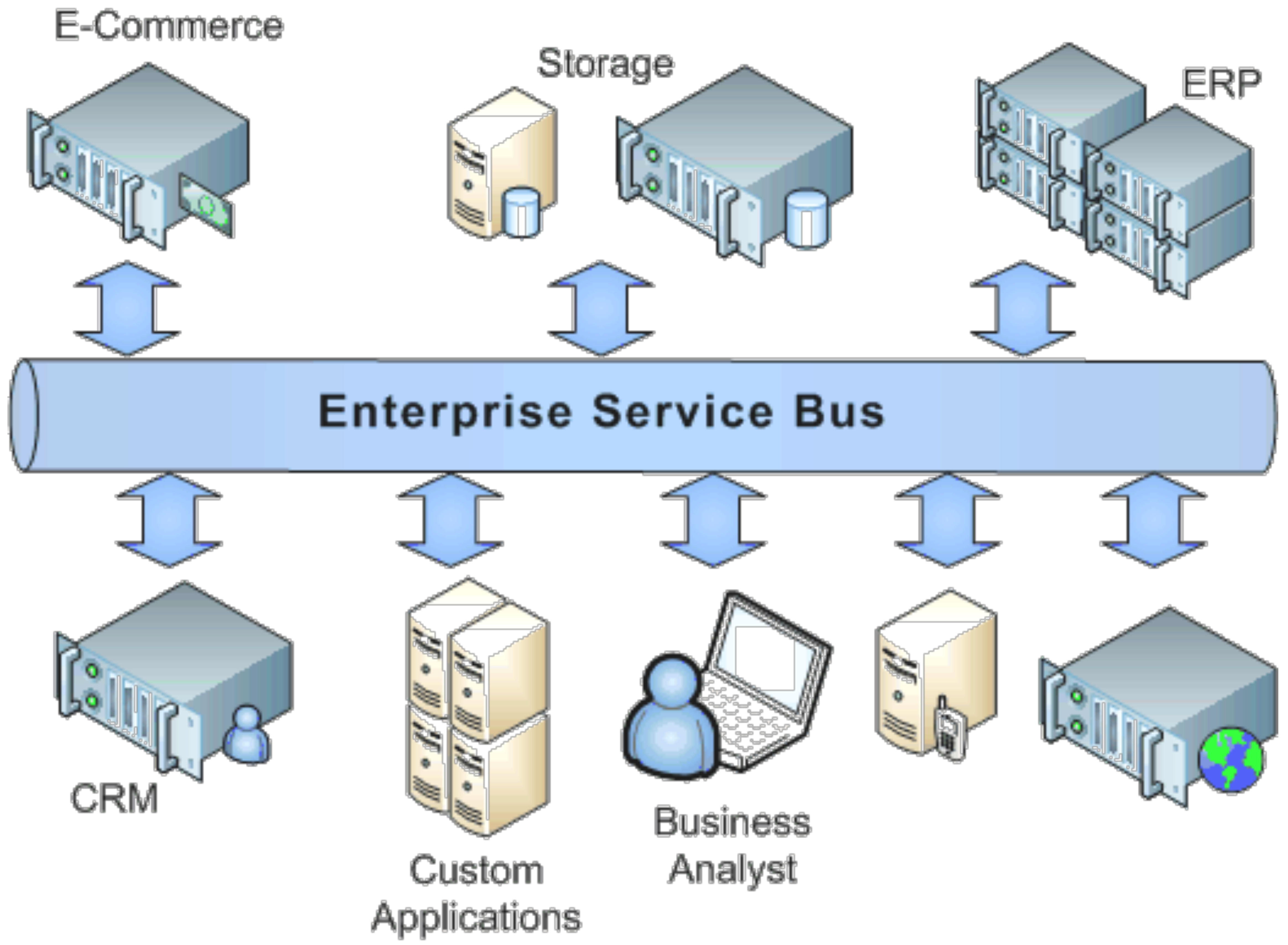
Avoids **Duplication**

Clearer

More **Extensible**

???

Bad Practice



Don’t Repeat Yourself (DRY)

“Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.”

- Andrew Hunt & David Thomas
The Pragmatic Programmer
Addison-Wesley, 1999

DRY software is…

Consistent

Easier to change

More likely to contain
better abstractions

Why does duplication arise?

The environment (seems to) require duplication

Duplication is unapparent

Laziness

Essential or accidental?

DRY and a cautionary tale

Idea: Clone Detection

Automatically identify fragments
of similar text within a project

Challenges

Clones can be syntactically different
but semantically equivalent

Clones can be over programs written in multiple
languages or in unstructured files (e.g. README)

Fragments can be identical now, but have
different reasons to change in the future

Challenges

Clones can be syntactically different
but semantically equivalent

Clones can be over programs written in multiple
languages or in unstructured files (e.g. README)

Fragments can be identical now, but have
different reasons to change in the future

Essential vs Accidental?

```
# "computeBalance" becomes "compute_balance"
def java_to_ruby_method(method_name)
  value = method_name[0..0].downcase + method_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end

# "MyModuleName" becomes "my_module_name"
def to_file_name(module_name)
  value = module_name[0..0].downcase + module_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end
```

Accidental: two different translations that have different reasons to change

http://www.informit.com/articles/article.aspx?p=1313447

Essential vs Accidental?

```
# "computeBalance" becomes "compute_balance"
def java_to_ruby_method(method_name)
  value = method_name[0..0].downcase + method_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end

# "MyModuleName" becomes "my_module_name"
def to_file_name(module_name)
  value = module_name[0..0].downcase + module_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end
```

```
# Client code
module_name = module_registry.find(modules.first).name
to_file_name(module_name)
```

http://www.informit.com/articles/article.aspx?p=1313447

Essential vs Accidental?

```ruby
# "computeBalance" becomes "compute_balance"
def java_to_ruby_method(method_name)
  value = method_name[0..0].downcase + method_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end

# "MyModuleName" becomes "my_module_name"
def to_file_name(module_name)
  value = module_name[0..0].downcase + module_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end
```

```
~~# Client code~~
~~module_name = modeul_registry.find(modules.first).name~~
~~camel_to_snake_case(module_name)~~
```

http://www.informit.com/articles/article.aspx?p=1313447

Essential vs Accidental?

```
# "computeBalance" becomes "compute_balance"
def java_to_ruby_method(method_name)
  value = method_name[0..0].downcase + method_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end

# "MyModuleName" becomes "my_module_name"
def to_file_name(module_name)
  value = module_name[0..0].downcase + module_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end
```

Essential: translating from CamelCase to snake_case
is a lower-level abstraction

http://www.informit.com/articles/article.aspx?p=1313447

A Possible Resolution

```ruby
# "computeBalance" becomes "compute_balance"
def java_to_ruby_method(method_name)
  value = method_name[0..0].downcase + method_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end

# "MyModuleName" becomes "my_module_name"
def to_file_name(module_name)
  value = module_name[0..0].downcase + module_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end

# "MyModuleName" becomes "my_module_name"
def camel_to_snake_case(camel_name)
  value = camel_name[0..0].downcase + camel_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end
```

http://www.informit.com/articles/article.aspx?p=1313447

A Possible Resolution

```ruby
# "computeBalance" becomes "compute_balance"
def java_to_ruby_method(method_name)
  camel_to_snake_case(method_name)
end


# "MyModuleName" becomes "my_module_name"
def to_file_name(module_name)
  camel_to_snake_case(module_name)
end


# "MyModuleName" becomes "my_module_name"
def camel_case_to_snake_case(camel_name)
  value = camel_name[0..0].downcase + camel_name[1..-1]
  value.gsub(/[A-Z]/) { |cap| "_#{cap.downcase}" }
end
```

http://www.informit.com/articles/article.aspx?p=1313447

Idea: Eliminate Duplication

Once identified essential duplication
should be removed immediately

Challenges

Reducing duplication often increases coupling

Discovering additional data points might
change the approach to eliminating duplication

Example

```
class StuffedCrust
  def bake
    # baking logic
  end
end
```

```
class DeepPan
  def bake
    # identical baking logic
  end
end
```

Example

```ruby
class StuffedCrust < Pizza
  def bake
     # baking logic
  end
end
```

```ruby
class DeepPan < Pizza
  def bake
     # identical baking logic
  end
end
```

Example

```
class StuffedCrust < Pizza
end
```

```
class DeepPan < Pizza
end
```

```
class Pizza
  def bake
    # baking logic
  end
end
```

Example

```
class StuffedCrust < Pizza
end


class Pizza
  def bake
     # baking logic
  end
end
```

```
class DeepPan < Pizza
end


class Calzone
  def bake
     # folding logic
     # baking logic
  end
end
```

Example

```ruby
class StuffedCrust < Pizza
end


class Pizza
  def bake
     # baking logic
  end
end
```

```ruby
class DeepPan < Pizza
end


class Calzone < Pizza
  def bake
     # folding logic
    super # baking logic
  end
end
```

Summary

Avoid duplication by representing every piece of knowledge once and only once

Consider whether duplication is accidental or essential before taking action

When reducing duplication: wait for the right abstraction & prefer to depend on stable canons