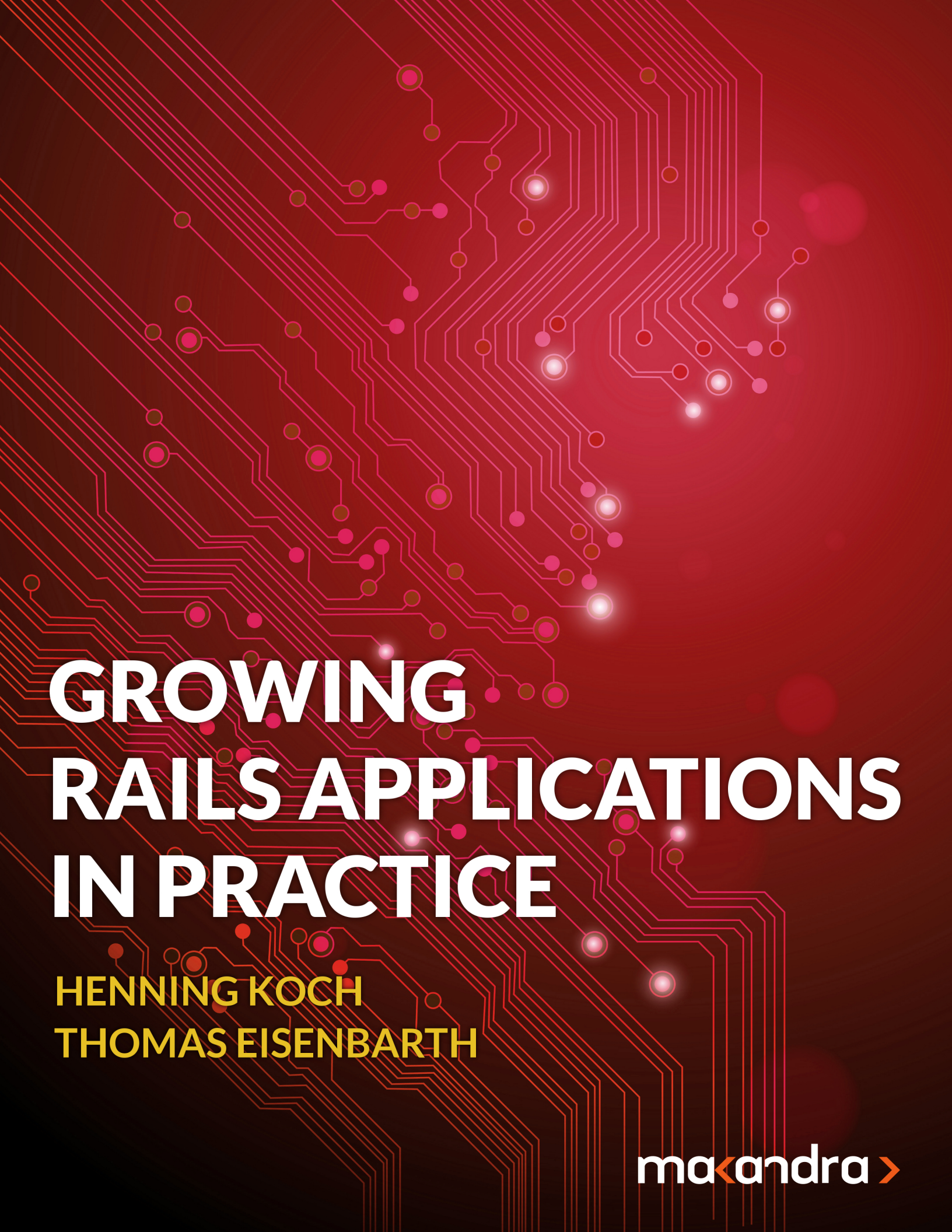
GROWING
RAILS APPLICATIONS
IN PRACTICE
HENNING KOCH
THOMAS EISENBARTH
manning

GROWING RAILS APPLICATIONS IN PRACTICE

Structure large Ruby on Rails apps using the tools you already know and love.

By Henning Koch and Thomas Eisenbarth

8. Organizing large codebases with namespaces

As a Rails application grows, so does its `app/models` folder. We've seen applications grow to hundreds of models. With an `app/models` directory that big, it becomes increasingly hard to navigate. Also it becomes near-impossible to understand what the application is about by looking at the `models` folder, where the most important models of your core domain sit next to some support class of low significance.

A good way to not drown in a sea of `.rb` files is to aggressively namespace models into sub-folders. This doesn't actually reduce the number of files of course, but makes it much easier to browse through your model and highlights the important parts.

Namespacing a model is easy. Let's say we have an `Invoice` class and each invoice can have multiple invoice items:

```
class Invoice < ActiveRecord::Base
  has_many :items
end

class Item < ActiveRecord::Base
  belongs_to :invoice
end
```

Clearly `Invoice` is a composition of `Items` and an `Item` cannot live without a containing `Invoice`. Other classes will probably interact with `Invoice` and not with `Item`. So let's get `Item` out of the way by nesting it into the `Invoice` namespace. This involves renaming the class to `Invoice::Item` and moving the source file to `app/models/invoice/item.rb`:

app/models/invoice/item.rb

```
class Invoice::Item < ActiveRecord::Base
  belongs_to :invoice
end
```

What might seem like a trivial refactoring has great effects a few weeks down the road. It is a nasty habit of Rails teams to avoid creating many classes, as if adding another file was an expensive thing to do. And in fact making a huge `models` folder even larger is something that does not feel right.

But since the `models/invoice` folder already existed, your team felt encouraged to create other invoice-related models and place them into this new namespace:

File	Class
app/models/invoice.rb	Invoice
app/models/invoice/item.rb	Invoice::Item
app/models/invoice/reminder.rb	Invoice::Reminder
app/models/invoice/export.rb	Invoice::Export

Note how the namespacing strategy encourages the use of Service Objects in lieu of fat models that contain more functionality than they should.

Real-world example

In order to visualize the effect that heavy namespacing has on a real-world-project, we refactored one of our oldest applications, which was created in a time when we didn't use namespacing.

Here is the `models` folder before refactoring:

app/models before refactoring

```
activity.rb
amortization_cost.rb
api_exchange.rb
api_schema.rb
budget_calculator.rb
budget_rate_budget.rb
budget.rb
budget_template_group.rb
budget_template.rb
business_plan_item.rb
business_plan.rb
company.rb
contact.rb
event.rb
fixed_cost.rb
friction_report.rb
internal_working_cost.rb
invoice_approval_mailer.rb
invoice_approval.rb
invoice_item.rb
invoice.rb
invoice_settings.rb
invoice_template.rb
invoice_template_period.rb
listed_activity_coworkers_summary.rb
```

```
note.rb
person.rb
planner_view.rb
profit_report_settings.rb
project_filter.rb
project_link.rb
project_profit_report.rb
project_rate.rb
project.rb
project_summary.rb
project_team_member.rb
project_type.rb
rate_group.rb
rate.rb
revenue_report.rb
review_project.rb
review.rb
staff_cost.rb
stopwatch.rb
task.rb
team_member.rb
third_party_cost.rb
third_party_cost_report.rb
topix.rb
user.rb
variable_cost.rb
various_earning.rb
workload_report.rb
```

Looking at the huge list of files, could you tell what the application is about? Probably not (it's a project management and invoicing tool).

Let's look at the refactored version:

app/models after refactoring

```
/activity
/api
/contact
/invoice
/planner
/report
/project
activity.rb
contact.rb
planner.rb
invoice.rb
project.rb
user.rb
```

Note how the `app/models` folder now gives you an overview of the core domain at one glance. Every single file is still there, but neatly organized into a clear directory structure. If we asked a new developer to change the way invoices work, she would probably find her way through the code more easily.

Use the same structure everywhere

In a typical Rails application there are many places that are (most of the time) structured like the `models` folder. For instance, you often see helper modules or unit tests named after your models.

When you start using namespaces, make sure that namespacing is also adopted in all the other places that are organized by model. This way you get the benefit of better organization and discoverability in all parts of your application.

Let's say we have a namespaced model `Project::Report`. We should now namespace helpers, controllers and views in the same fashion:

File	Class
`app/models/project/report.rb`	`Project::Report`
`app/helpers/project/report_helper.rb`	`Project::ReportHelper`
`app/controllers/projects/reports_controller.rb`	`Projects::ReportsController`
`app/views/projects/reports/show.html.erb`	View template

Note how we put the controller into a `Projects` (plural) namespace. While this might feel strange at first, it allows for natural nesting of folders in in `app/views`:

```
app/
  views/
    projects/
      index.html.erb
      show.html.erb
      reports/
        show.html.erb
```

If we put the controller into a `Project` (singular) namespace, Rails would expect view templates in a structure like this:

```
app/
  views/
    project/
      reports/
        show.html.erb
    projects/
      index.html.erb
      show.html.erb
```

Note how two folders `project` (singular) and `projects` (plural) sit right next to each other. This doesn't feel right. We feel that the file organization of our views is more important than keeping controller namespace names in singular form.

Organizing test files

When we have tests we nest the test cases and support code like we nest our models. For instance, when you use RSpec and Cucumber, your test files should be organized like this:

File	Description
`spec/models/project/report_spec.rb`	Model test
`spec/controllers/projects/reports_controller_spec.rb`	Controller test
`features/project/reports.feature`	Cucumber untegration test
`features/step_definitions/project/report_steps.rb`	Step definitions

Other ways of organizing files

Of course models/controllers/tests don't always map 1:1:1, but often they do. We think it is at the very least a good default with little ambiguity. When you look for a file in a project structured like this, you always know where to look first.

If another way to split up your files feels better, just go ahead and do it. Do not feel forced to be overly consistent, but always have a good default.