Expertise
in Java, Groovy/Grails, iOS and Open Source technologies

Experienced
servicing a multitude of industries

Object Partners

Accountability
through agile techniques and practices

Delivery
of large scale, mission critical applications

Fine Grained Security in Grails Applicaitons

Andrew Miller
amiller@objectpartners.com

Have you ever seen an old photo of yourself...

...and been embarrassed at the way you looked?

Have you looked at old code you wrote...

```
PROGRAM MAIN
INTEGER N, X
EXTERNAL SUB1
COMMON /GLOBALS/ N
X = 0
PRINT *, 'Enter number of repeats'
READ (*,*) N
CALL SUB1(X,SUB1)
END
```

```
SUBROUTINE SUB1(X,DUMSUB)
INTEGER N, X
EXTERNAL DUMSUB
COMMON /GLOBALS/ N
IF(X .LT. N)THEN
  X = X + 1
  PRINT *, 'x = ', X
  CALL DUMSUB(X,DUMSUB)
END IF
END
```

...and been embarrassed at how clumsy it looked?

Have you looked at old project plans and budgets...

...and been embarrassed at how much money you spent on the things that today are so easy?

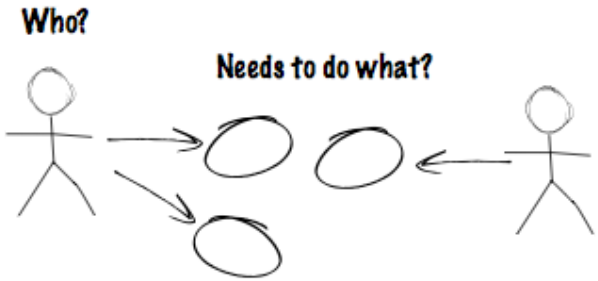
The truth is we spent entire project budgets on what, then, was the ”hard stuff”...

- figuring out how we were gonna organize our source code
- figuring out dependency management
- figuring out build automation
- figuring out how we're gonna save data in a database
- figuring out test automation
- ...

Looking back, it's amazing we had time (or money) left over to work on the important things...

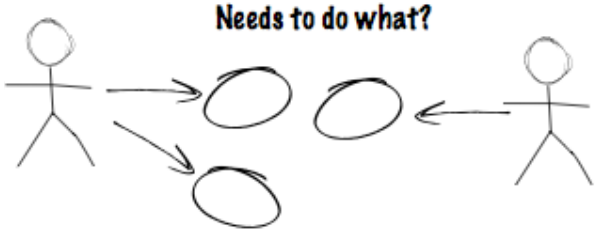
LIKE WHAT THE SOFTWARE WAS SUPPOSED TO ACTUALLY DO!

When building a system...

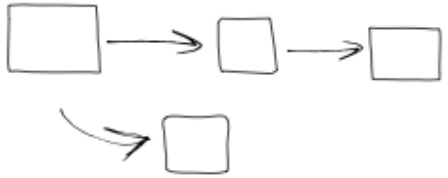


Who?

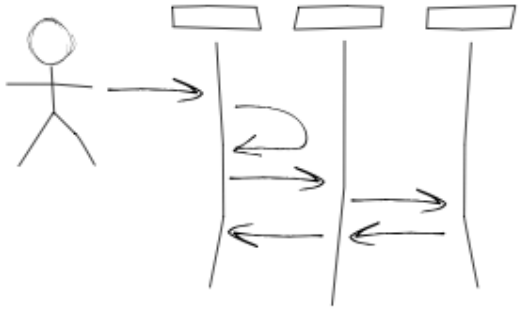
Needs to do what?



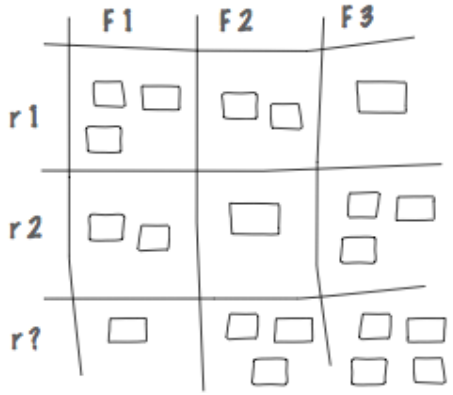
What are the elements of the system?



How's the system gonna do that?



What's the priority and schedule?



Some things are just ”hard”...

- Requirements management
- Integrating with other systems
- Asynchronous processing
- Application security
- Batch processing
- Functional testing
- Working with money
- ...

Grails solves some hard stuff right out of the box

- structuring the vertical slice
- organizing source code
- dependency management
- build automation
- persistence
- schema evolution
- test automation

- groovy
- plugins
- ...

Some things are just ”hard”...

- Requirements management
- Integrating with other systems
- Asynchronous processing
- Application security
- Batch processing
- Functional testing
- Working with money
- ...

Some things are just ”hard”...

- Requirements management
- Integrating with other systems
- Asynchronous processing
- **Application security**
- Batch processing
- Functional testing
- Working with money
- ...

Application Security

0. anonymous (no security)
1. authentication of multiple clients
2. authentication of multiple users
3. user authentication AND user authorization
4. fine grained user authorization varying across resources
5. data security (at the query level)

Users & Roles

- Anonymous : No security necessary.
- Single User : one step beyond "anonymous". Address this situation by using the @Secured annotation (provided by the spring-security-core plugin) to secure controller actions.

```
@Secured(["isFullyAuthenticated()"])
```

- Multipe Users : As soon as you've secured an application you'll discover you can *do things* with the knowledge of who's loged in. Things like audit logging are possible (who did what/when.)
- Multipe Users / Multiple Roles

Users & Roles

- Multipe Users / Multiple Roles: With multiple users you'll find that you want to allow some users to do somethings and other users to do other things. This too can be handles with the @Secured annotation

```
@Secured(["ROLE_SUPERUSER"])
def rmMinusRStar(String pathname) {
    ...
}

@Secured(["ROLE_MOM"])
def makeGoodCookies(Integer count) {
    (count * 2).times {
        ...
    }
}
```

Multiple Tenants

This is a very common problem...

- multiple users
- multiple resources
- each resource is owned by a single user
- "OWNER" can grant access to other users

Eventually we're going to need to store data for multiple customers/tenants in the same database.

How do we secure this?

Multiple Tenants

When "roles" vary across multiple tenants user & roles aren't enough. Roles are great for implementing details like "Only the superuser can run with scissors" and "Never let Mom touch this feature!" But roles don't capture variation across resources.

One approach is to use the `spring-security-acl` plugin. The `spring-security-acl` plugin provides more than just these annotations. It's reason for existence is to provide ACL support for fine grained control of access rights BUT...

The annotations are applied at the service level, not on controller actions.

Multiple Tenants

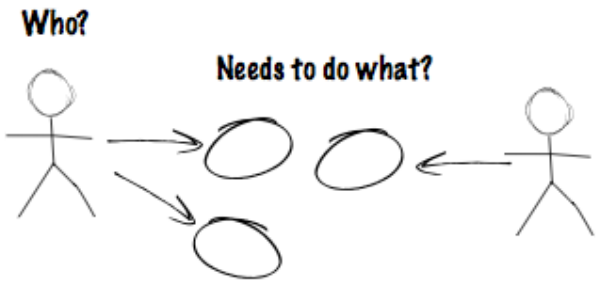
```
@Secured(["isFullyAuthenticated()"])
def show(Organization org) {
    ...
}
```

Multiple Tenants

```
@Secured(["isFullyAuthenticated()"])
@Authorized( clazz = Organization, idParam = "id", permission="VIEWER" )
def show(Organization org) {
    ...
}
```

When "Roles" Vary Across Multiple Tenants

0. anonymous (no security)
1. authentication of multiple clients (bearer tokens to secure a rest api)
2. authentication of multiple users (spring security to restrict access to ctrl/action)
3. user authentication AND user authorization (spring security "authorities" to restrict acces to ctrl/action)
4. user authorization that varies across different data elements == fine grained security
5. data security (at the query level)


Who?
Needs to do what?

Fine Grained Security Example

1. I (a user) can create new resources
2. I "own" resources that I've created
3. I am a "VIEWER" for some resources
4. I can grant "OWNER", "VIEWER", and so on to other users

don't drive drunk

usage example: class level authorization

```
@Authorized( clazz = Project, permission="OWNER" )
def create() {
    ...
}

@Authorized( clazz = Project, permission="OWNER" )
def save() {
    ...
}
```

usage example: instance level authorization

```
@Authorized( clazz = Project, idParam = "id", permission="OWNER" )
def edit(Project project) {
    render template: "edit", model: [project: project]
}

@Authorized( clazz = Project, idParam = "id", permission="OWNER" )
def update(Project project) {
    ...
}
```

usage example: list of authorized instances

```
def list() {
    def user = springSecurityService.currentUser
    def projectList = authorizationService.authorizedInstances(
                          user?.id,
                          Project,
                          params
                      )
    def projectTotal = authorizationService.authorizedInstanceCount(
                           user?.id,
                           Project
                       )
    [ projectList: projectList, projectTotal: projectTotal, ]
}
```

implementation...

grails-domain-authorization plugin (in progress)
github.com/onetribeyoyo/grails-domain-authorization

- `@Authorized` annotation
- `DomainAuthorizationFilters`
- `AuthorizationService`
- `Authorization` (domain object)
- `AuthorizationTagLib`

Questions?

grails-domain-authorization plugin (in progress)
github.com/onetribeyoyo/grails-domain-authorization

magic-task-machine (example app)
github.com/onetribeyoyo/mtm

Andrew Miller

amiller@objectpartners.com

(612) 269-7369