SLM

Sample Lifecycle Manager
• BCF - Bio Computing Facility

• UAGC - University of Arizona Genetics Core

• ARL - Arizona Research Laboratories
UAGC provides molecular biology lab services to:
- On campus UA researchers
- University and non-profit scientists around the world
- Corporate programs including:
  - National Geographic Genographic project
  - Family Tree DNA

UAGC processes > 100K samples per year
UAGC services:

- Sample prep:
  - extraction, quantification, normalization, pcr, storage, plating
- 454 sequencing
- Ion Torrent sequencing
- Sequenom genotyping and methylation analysis
- Sanger sequencing
- Fragment/str/microsatellite analysis
- Taqman genotyping
- Transgenic genotyping and cell line authentication
- Real-Time PCR
- Bioinformatics support and data analysis
- UAGC environment:
  - High throughput automated laboratory
● SLM Goal:
  o Integration of all UAGC services to provide complete and customized tissue-to-data workflow services for our customers.

● SLM Scope:
  o Customer facing sample submission, data delivery, and collaboration.
  o Lab facing sample and workflow management.
  o Model every existing laboratory process, and allow 'easy' expansion for new processes.
- SLM Features:
  - User, staff, and lab management
  - Sample submission
  - Sample and reagent volume and concentration tracking
  - Automated dilutions
  - 'Cherrypicking' reaction setup and plating
  - Robotic transfer builder
  - GLP status logging
  - Samplesheet creation
  - Result data delivery and collaboration
  - Billing reports
- SLM First Generation
  - ActionsScript/Flex front end, Python backend.
  - Traditional architecture with many user options.
● Why it didn't work
  o Too many options to cover all possible workflows
  o Difficult for users to understand all options
  o Difficult for developers to manage/test all possible option configurations.
SLM Second Generation
  
  - Provide a modular architecture that makes it easier for developers to custom code exactly what's needed for a specific workflow.
- **Eager**
  - A framework for developing workflow based web applications

- Based on Django - a Python library for MVC web apps
- Provides extensible workflow definitions
- Provides common LIMS (laboratory information management system) models and methods:
  - Tube/plate management
  - Well label translations
  - Grid file parsing/writing
  - Volume/concentration tracking
  - Results data management
● HTML

● Pros:
  o Modular using server-side templates
  o No special software required
  o Easily styled with CSS
  o Accessibility

● Cons:
  o Browser compatibility issues
  o RIA requires lots of Javascript, which can have performance problems on older machines
Dojo Javascript Library

Pros:
- Everything you could ever want is built-in
- Modules and classes allow code organization
- Automated build process for optimizing code

Cons:
- Not as elegant or concise as JQuery
- Sometimes the documentation is good (API), sometimes not so much (tutorials)....
- Apache/mod_wsgi
  
  **Pros:**
  - Available on every Linux box
  - OK documentation
  - Everyone knows how to configure it (or they should at least...)

  **Cons:**
  - Doesn't have the performance and scalability of more modern web servers such as nginx
• Python Pros

• Elegant and concise (unlike this slide)
• Great documentation
• Tons of HIGH QUALITY libraries, with large std lib
• Useful collection of built-in data types
• Cool features:
  o lambdas, list comprehensions, generators, properties, decorators
• Avoids pitfalls of other dynamic languages:
  o Namespaced
  o Everything is an object
  o Strong typing with no implicit or explicit casting
  o Runtime error on undefined variables
  o Error handling is consistent: runtime errors are always exceptions
● Python Cons

● Cons:
  ○ Some people think it's too slow... (BUT, writing C extensions is pretty easy)
  ○ Some people don't like blocks defined by indentation... (BUT, those people are crazy)
● **Django** Pros

● Large community
● Great documentation
● Reusable 'applications'
● Simple request handlers
● Form builder and validation, including CSRF protection
● ORM (for simple models and tasks)
● Command line scripts and shell interface
● Designers can create and modify templates
● Easy URL routing
● Built-in management site (we don't use this feature)
Django Cons

- ORM (for complex models and tasks)
- They call their 'controller' the 'view'
- Some nit-picky technical issues with templates
- SQLite

- Pros:
  - Single file database
  - No configuration required

- Cons:
  - Can't handle concurrent writes
● PostgreSQL

● Pros:
  o Tons of features
  o Not owned by Oracle
  o Easy configuration
  o Works how you expect it to

● Cons:
  o Connection overhead (can be mitigated by using connection pooling)
• Application demo
Contact info:

- Blog: [www.limscoder.com](http://www.limscoder.com)
- Twitter: @limscoder