

How we learned to stop worrying and love the

Grand Unified Task System

SCCS
Swarthmore College Computer Society

sccs.swarthmore.edu/projects/guts

Questions we'll answer today:

1. Who are we, anyway?
2. Why did we make GUTS?
3. How does GUTS work?
4. What did we learn?
5. Can I use GUTS? (Yes!)

SCCS - Swarthmore College Computer Society



- Student-run computer enthusiast/sysadmin group
- Founded in 1991 to support a network of phone cable strung between windows of two adjacent student dorms
- Introduced networking to the college two years before Swarthmore's IT department existed
- Today we have our own facilities and offer computing services (email, mailing lists, web space, shell access etc.) to students, faculty and college staff
- Run by a band of volunteer student sysadmins

Why did we make GUTS?

- Manually handling user requests (account creation, password changes) becomes tedious
- Various checks of input can be done automatically, for some we want administrator review
- As full-time students, the time we have to attend to user requests is limited, so automation and ease of review for staff is essential
- Predecessor software to serve this purpose suffered from several limitations

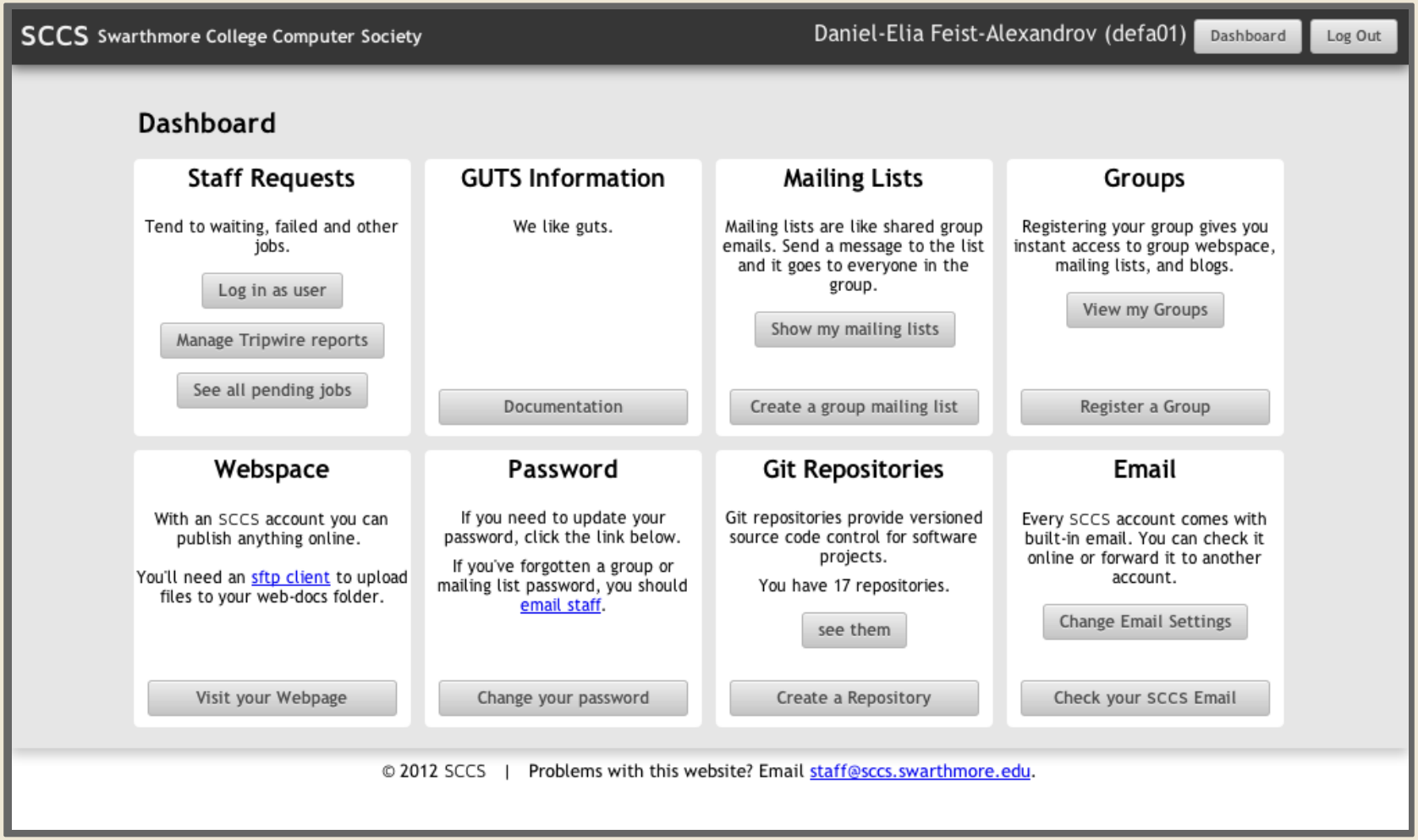
What came before it?

- The LOS task request system - an earlier effort to build such a system (presented at LISA '02)
- Good for its time, but by 2011, showing its age
- Required approving tasks via a shell or GPG-signed email - no web interface
- Task definitions were an idiosyncratic combination of perl and XML, hard to understand and extend
- Web forms for submitting tasks were automatically generated from task descriptions - neat idea, but in practice led to poor user experience

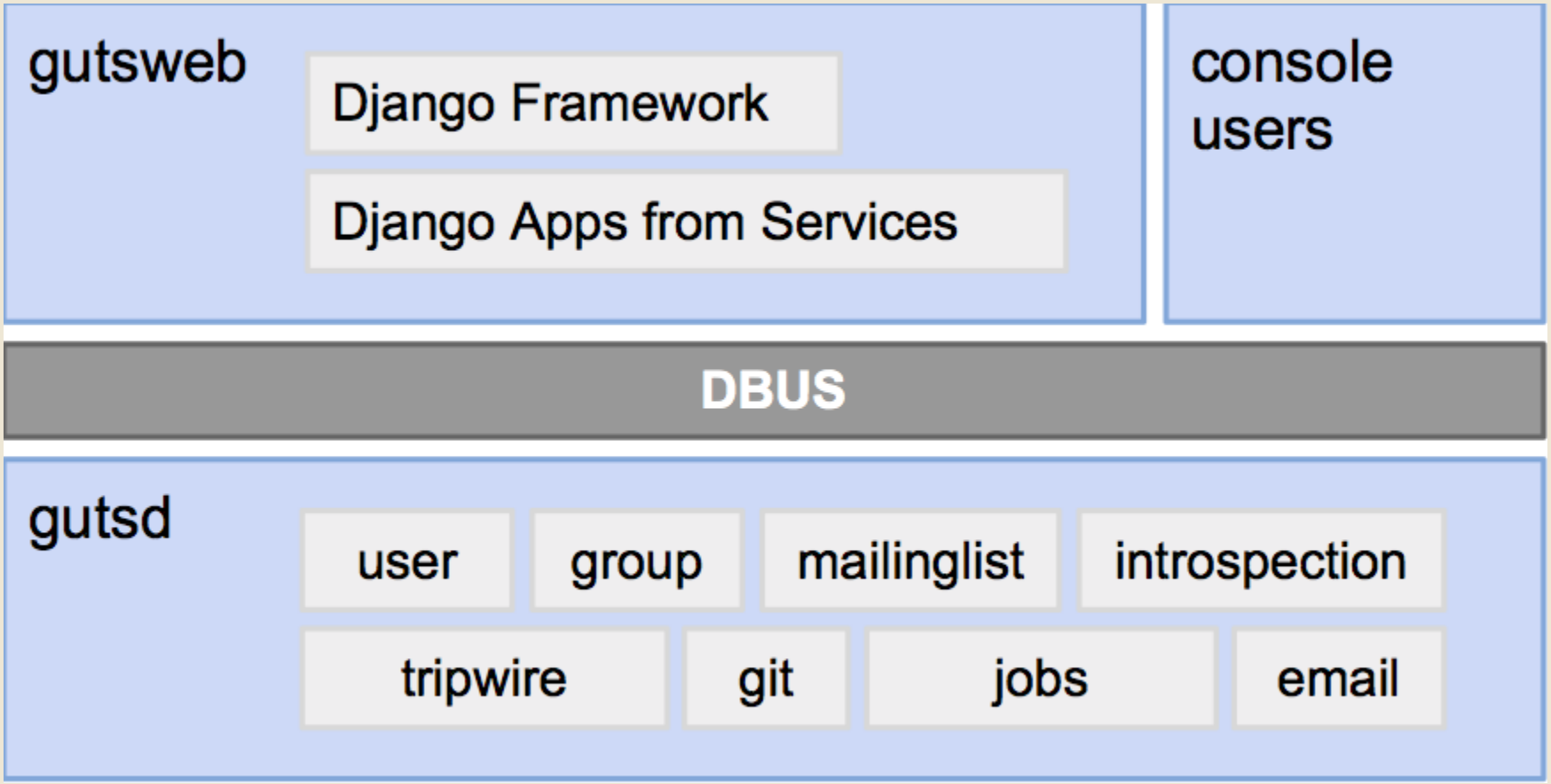
Design goals for GUTS

- Modern web interface for task management, review, and approval - allow mobile access
- Same interface for admins and users (i.e. admins get the benefit from UI efforts)
- Easy to understand, maintain, and extend - use common frameworks, modular design
- Better user experience, more flexible UI than automatically generated forms
- Keep security and privilege separation of Los

The GUTS dashboard



Overview of GUTS architecture



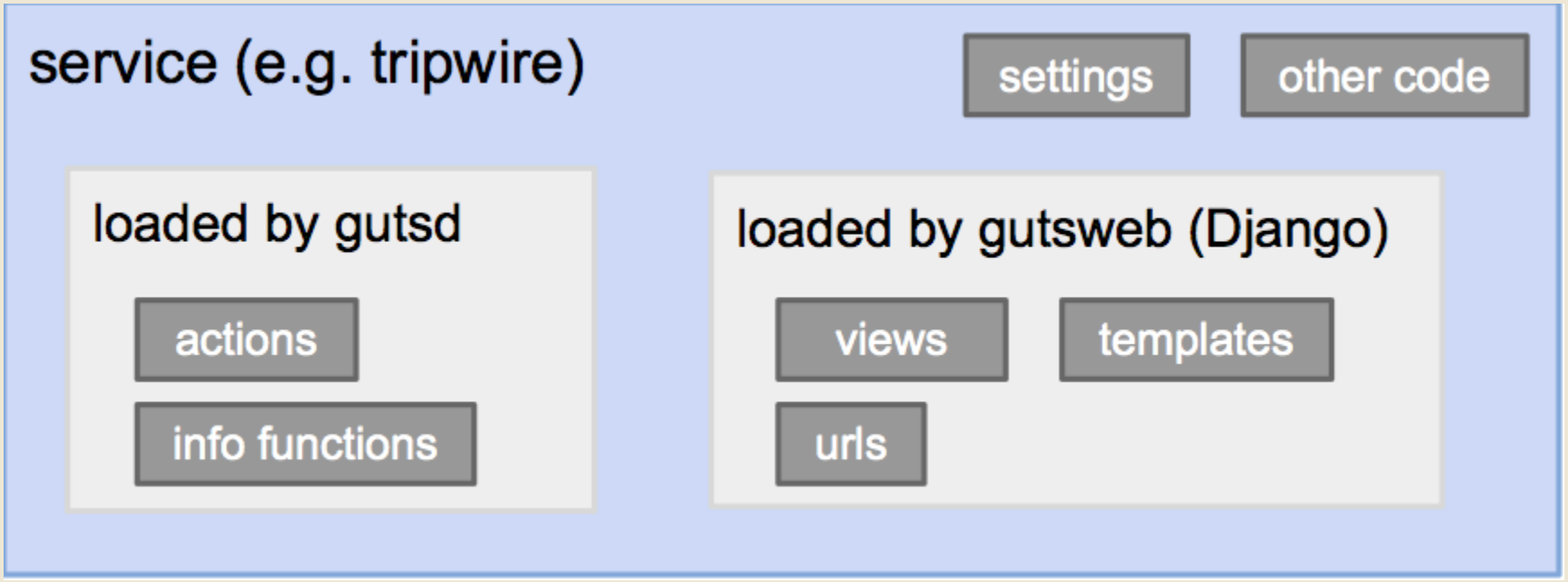
`gutsweb`, `gutsd` and `libguts`

- `gutsweb` is the front-facing part of GUTS
  - Uses Django framework to provide a unified, rich UI to both users and admins
  - runs as unprivileged user
- `gutsd` does all the heavy lifting
  - runs with elevated privileges and calls the Python code that actually executes the desired task
- the two communicate via D-Bus
  - ensures separation of web UI from privileged execution
- `libguts`, the GUTS library for writing services
  - allows developer to easily write GUTS services in Python

Services in GUTS

- GUTS services are modular extensions to the core system that are modeled on Django apps
- Easy to develop service without knowing GUTS core
- Components of a service
  - Frontend Django app files (loaded by `gutsweb`)
    - `urls.py` - defines service sub-URLs for this service
    - `views.py` - implements application views
    - `templates/` - holds HTML template files for pages
  - Backend code for task execution (loaded by `gutsd`)
    - `functions.py` - contains registered GUTS functions
    - external code (e.g. for Mailman, git, etc.)
  - Common `settings.py` file for service-specific settings

Anatomy of a GUTS service



Parameter checking

- Registered GUTS functions can make use of parameter checkers provided by `libguts`
  - These allow for verification of commonly used parameters
  - e.g. checking if a parameter is a valid UNIX user/group

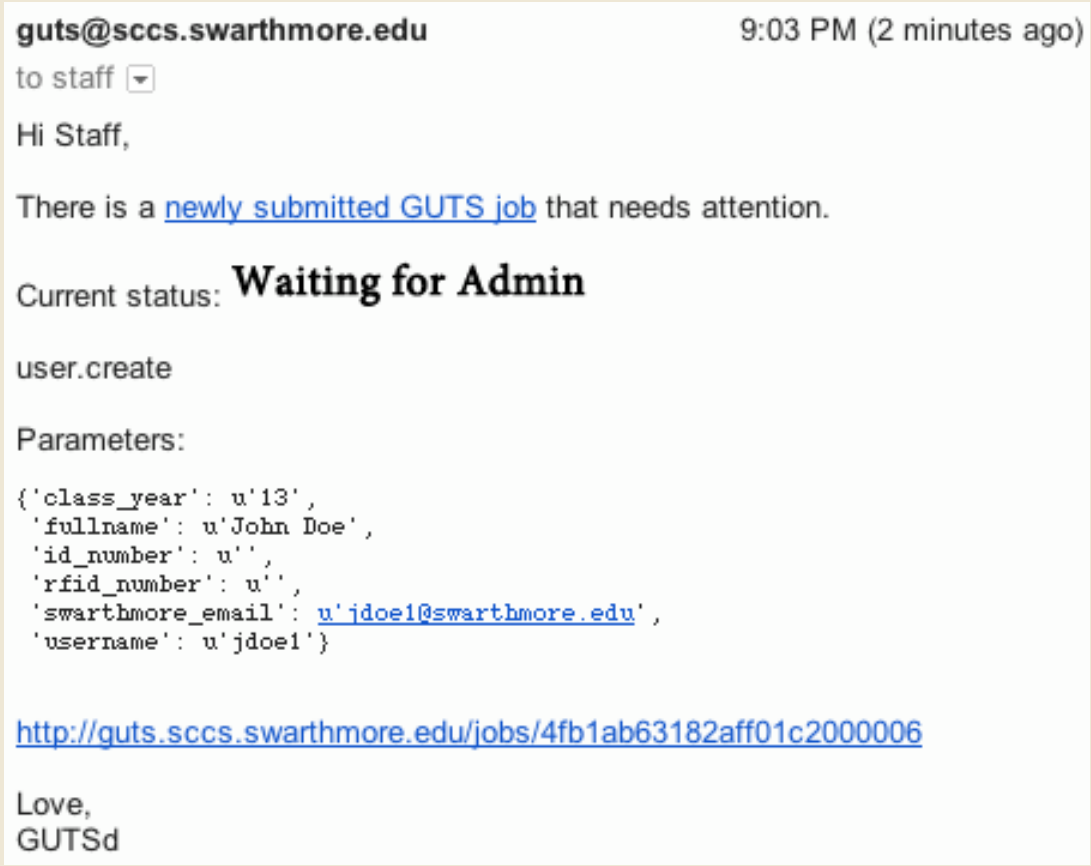
```
_permissions = (None, "none", "read", "write", "admin")
_visibilities = (None, "public", "private")
@guts.action(service="git", checker=check_user_is_admin)
def update_settings(context,
                    name = Param(checker=repo_name_check, required=True),
                    description = Param( default=None),
                    visibility  = Param(choices=_visibilities, required=True),
                    entity_name = Param("user-or-group", default=None),
                    permissions = Param(choices=_permissions, default=None)):
    pass # GUTS action logic goes here
```

Nifty things we like about GUTS

- Parameter Checkers
  - Take advantage of Python's default argument syntax to bundle information about GUTS function parameter types and checkers
- Introspection
  - GUTS dashboard interface that lets an admin inspect individual services and their functions
  - Ability to execute tasks with fully customizable parameters without having to go to the shell

Nifty things we like about GUTS

- Live status images in emails for tasks needing approval



guts@sccs.swarthmore.edu 9:03 PM (2 minutes ago)

to staff

Hi Staff,

There is a newly submitted GUTS job that needs attention.

Current status: **Waiting for Admin**

user.create

Parameters:

```
{'class_year': u'13',
 'fullname': u'John Doe',
 'id_number': u'',
 'rfid_number': u'',
 'swarthmore_email': u'jdoe1@swarthmore.edu',
 'username': u'jdoe1'}
```

http://guts.sccs.swarthmore.edu/jobs/4fb1ab63182aff01c2000006

Love,
GUTSd

So, the outcome: success!

We launched GUTS to the College community in October 2011, roughly 6 months after initial development began.

Successes:

- Continued iterating and improving GUTS over the year
  - Grew from ~1.5k lines to over 5k lines of Python
- Staff response time to user requests dropped by 20x
  - From median time of over 4 hours to only 14.3 minutes
- Reached a greater number of potential users
  - Tripled number of account creations per semester
- Tied together the majority of our services via GUTS (Mailman, git, email, etc.)

What we learned

The good news:

- Integrated interfaces help both users and admins
- Solid core + modular extensions = easy development
- Building your own system from scratch is rewarding!

The other stuff:

- Users are users - hard to expose all functionality
  - e.g. we still get password reset request emails
- Some services led to rethinking parts of GUTS
  - e.g. Tripwire requires asynchronous task execution

Current and future work on GUTS

- Remodel front end of GUTS to make it more intuitive, take advantage of Javascript/JQuery, mobile version
- Implement new GUTS services
  - Changing information/photo in campus directory
  - Reserving SCCS spaces and equipment
  - Modifying settings on user web space
- Add features to the GUTS core
  - Asynchronous task execution
  - Testing framework, mocks for key services
- Generalize GUTS and eliminate SCCS-specific code, which brings us to...

Can I have it?

Yes!

GUTS is released under the GNU GPLv3.

Find more information about GUTS, source code, documentation, and our paper at:

`sccs.swarthmore.edu/projects/guts`

We'd like to thank...

- Our LISA shepherd Marc Chiarini and the anonymous reviewers for their input
- The USENIX association for providing us with student grants
- The Swarthmore College Chapter of *Sigma Xi* for providing us with travel grants

Questions?

Thanks for attending!

Daniel Feist-Alexandrov & Nick Felt