Science Product Development through Community Collaboration and the Open Source Framework

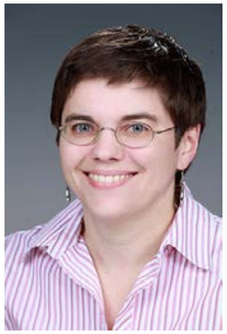
JENNIFER COMSTOCK, KRISTA GAUSTAD, JOE HARDIN, EUGENE CLOTHIAUX

Laura Riihimaki, Scott Collis, Pavlos Kollias

Science Product Development Led by Team of Scientists

ARM
CLIMATE RESEARCH FACILITY

Translators


Laura Riihimaki
Clouds - Radiometric

Connor Flynn
Aerosols

Scott Collis
Precipitation Radar

Scott Giangrande
Cloud Radar

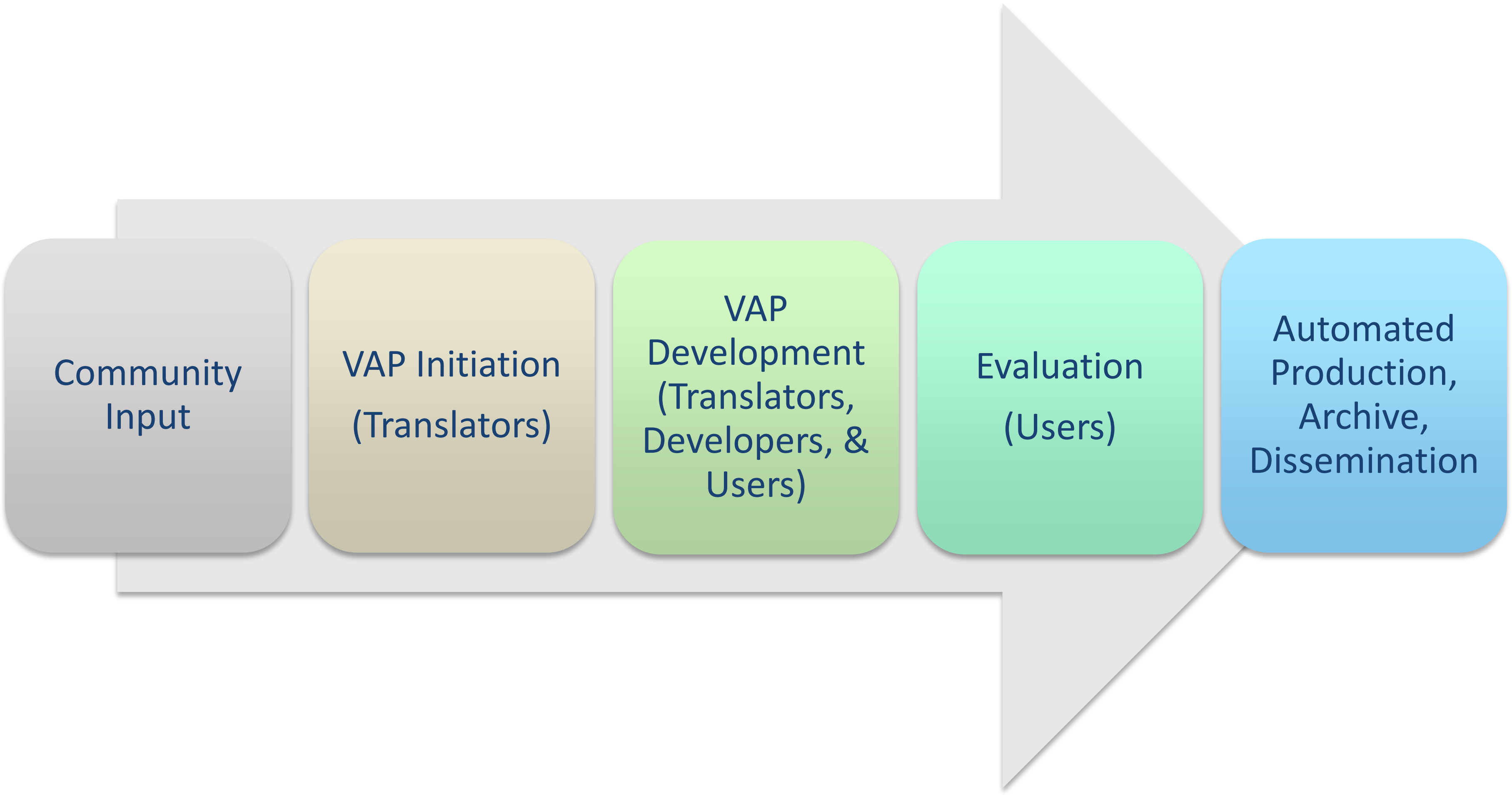
Shaocheng Xie
Modeling

Laurie Gregory
External Data
Products

Chitra Sivaraman
Software
Development

Justin Monroe
Data Quality

VAP Development Process

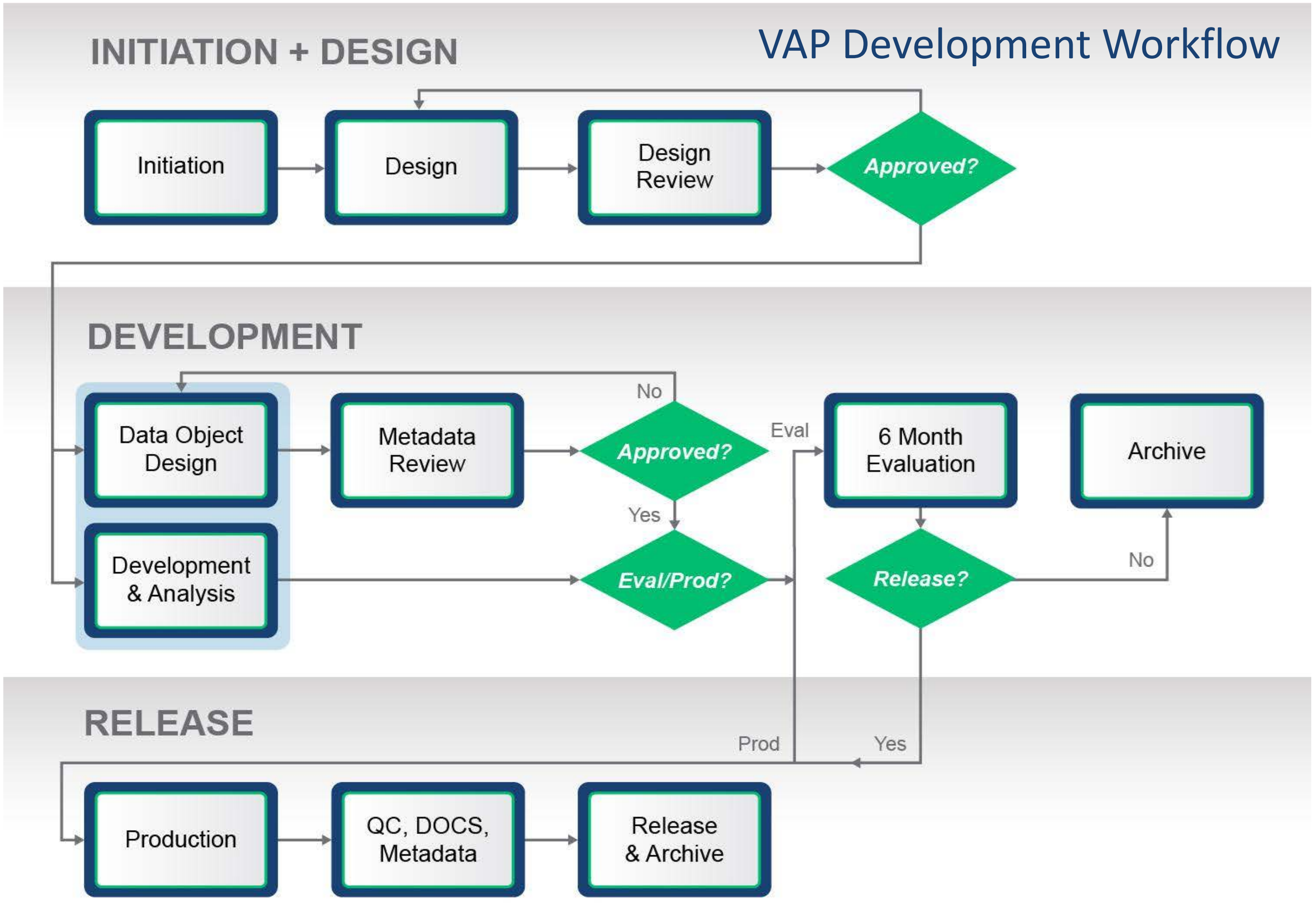

Community Input
VAP Initiation
(Translators)
VAP Development (Translators, Developers, & Users)
Evaluation
(Users)
Automated Production, Archive, Dissemination

VAP Development Process

Initiation
- Community Need
- PI Sponsor – provides code
- Contract

Development
- Traditional path
- Code Sprint
- Evaluation or Production?

Release
- Production ready
- Documentation



Code Sprint Greatly Reduces VAP Development Time

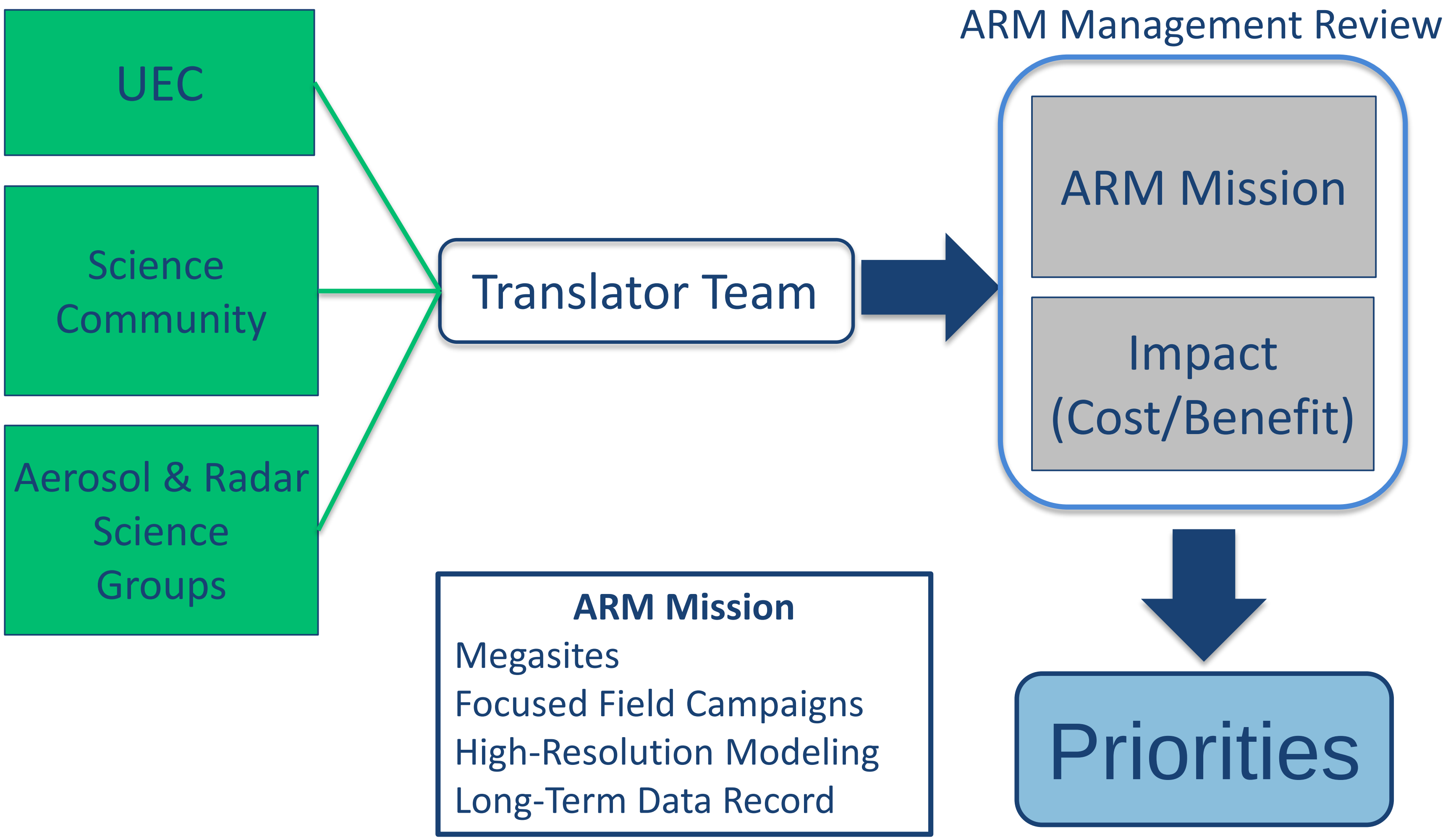
SACR Advanced Velocity-Azimuth Display (ADV-VAD) VAP

Last week of June 2016	July 2016	August 2016	Sept. 2016
Code Sprint: draft implementation plan, recode in Python	Complete code, port to ARM ADI environment	Additional analysis & testing, prepare for release	DOD approved, historical processing done. VAP released for Evaluation

VAP development support (~70 hours)

Two more complex SACRADV code sprint VAPs reached Evaluation in Jan/2017

Prioritizing ARM Activities Value Added Products



ARM Mission
Megasites
Focused Field Campaigns
High-Resolution Modeling
Long-Term Data Record

Session Topics

ARM
CLIMATE RESEARCH FACILITY

- Introduction to ARM Science Product Development (Jennifer Comstock)
- ARM Data Integrator (Krista Gaustad)
- Open source development and code sharing (Joe Hardin)
- Scientists Perspective on Code Sprints – Experiences from SACR ADV (Eugene Clothiaux)

ARM
CLIMATE RESEARCH FACILITY

Helpful hints to make your code ADI compatible

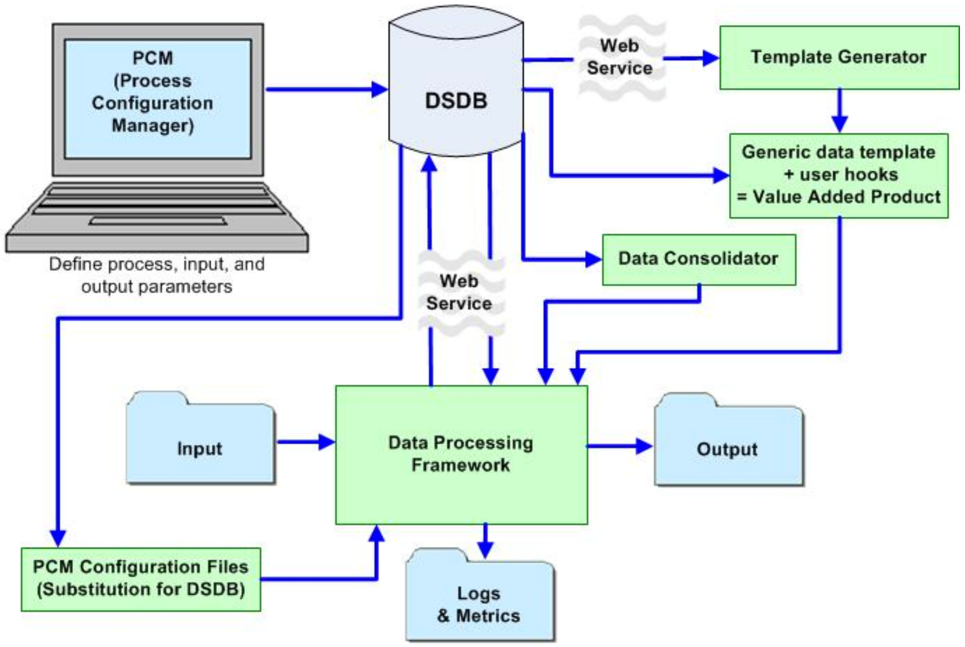
KRISTA GAUSTAD

Pacific Northwest National Laboratory—ARM Developer, ADI lead

U.S. DEPARTMENT OF ENERGY

ARM Data Integrator (ADI) Simplifies Code Development and Application of Standards

Suite of tools, code libraries, structures and interfaces to standardize ARM code and data, and make operational processing of data products more robust and efficient.


PCM (Process Configuration Manager)
Define process, input, and output parameters
DSDB
Web Service
Template Generator
Generic data template + user hooks = Value Added Product
Data Consolidator
Web Service
Input
Data Processing Framework
Output
PCM Configuration Files (Substitution for DSDB)
Logs & Metrics

- Promotes robust processes that use well tested libraries and functions
- Enforces ARM Standards
- Documents dependencies, metrics, status, and logs
- Automates reprocessing
- Captures provenance
- Streamlines algorithm implementation

What can users do to make their code ADI compatible?

1. Write your code in an ADI compatible language
   - C
   - Python
   - IDL
   - MatLab
2. Separate the code that reads and writes data from all other code

Separating input/output functionalities

- Read data
  - Get file list
  - For each file
    - For each line in file
      - Read var data into array
      - Apply limit tests
- Perform analysis
- Write data

- Get data from ADI
  - For each day
    - For each sample
      - Read var data into array
- Preprocess
  - For each sample
    - Apply limit tests
- Perform analysis
- Put data into ADI

Additional best practices for creating successful operational code

ARM CLIMATE RESEARCH FACILITY

https://www.arm.gov/policies/coding-guidelines

- Comment your code well
- Modularize your code
- Fun with flags
  - Test your algorithm for long periods of time to understand when it does and doesn’t work
  - Document times when algorithm works poorly (because of data quality, algorithm assumptions not met, etc) with quality flags

What can users do to make their code ADI compatible? Top two things:

1. Write your code in an ADI compatible language
   - C
   - Python
   - IDL
   - MatLab
2. Separate the code that reads and writes data from all other code

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

Collaborative Code Sharing and Development

JOSEPH C HARDIN, SCOTT COLLIS

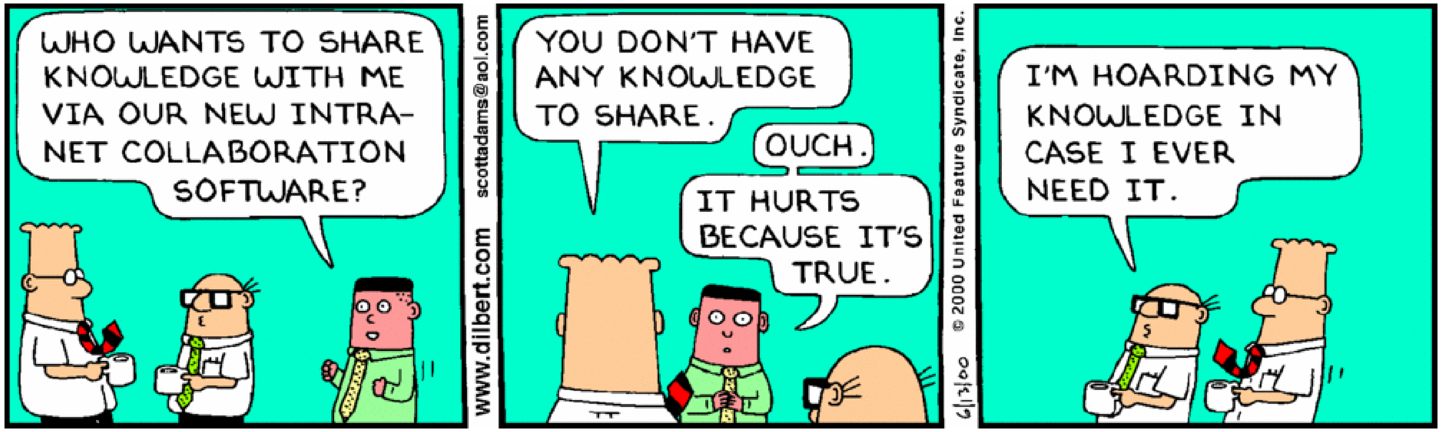
PNNL Radar Engineering and Operations

DOE ARM/ASR PI Meeting 2017

U.S. DEPARTMENT OF
ENERGY

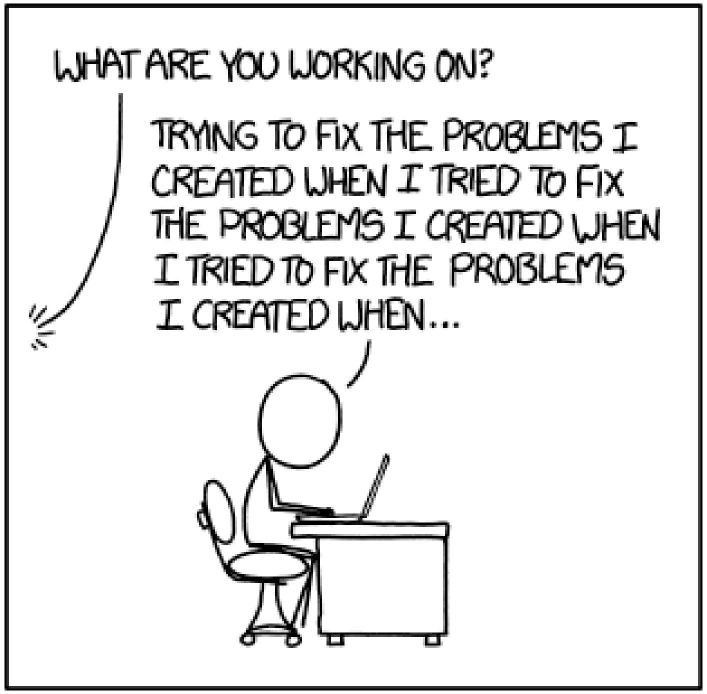
Collaboration

- ARM loves contributed code and algorithms.
- It will spend developer time adapting community algorithms into VAPS.
  - This works better if the original author is involved.
- Development of VAPS requires a back and forth between a
  - Developer who does not understand the underlying algorithm
  - Science point of contact who does not understand ARMs data infrastructure.
- Let's not re-invent the wheel. There are accepted methods for development between distributed teams.


WHO WANTS TO SHARE KNOWLEDGE WITH ME VIA OUR NEW INTRA-NET COLLABORATION SOFTWARE?
scottadams@aol.com
www.dilbert.com
YOU DON'T HAVE ANY KNOWLEDGE TO SHARE.
OUCH.
IT HURTS BECAUSE IT'S TRUE.
© 2000 United Feature Syndicate, Inc.
6/13/00
I'M HOARDING MY KNOWLEDGE IN CASE I EVER NEED IT.

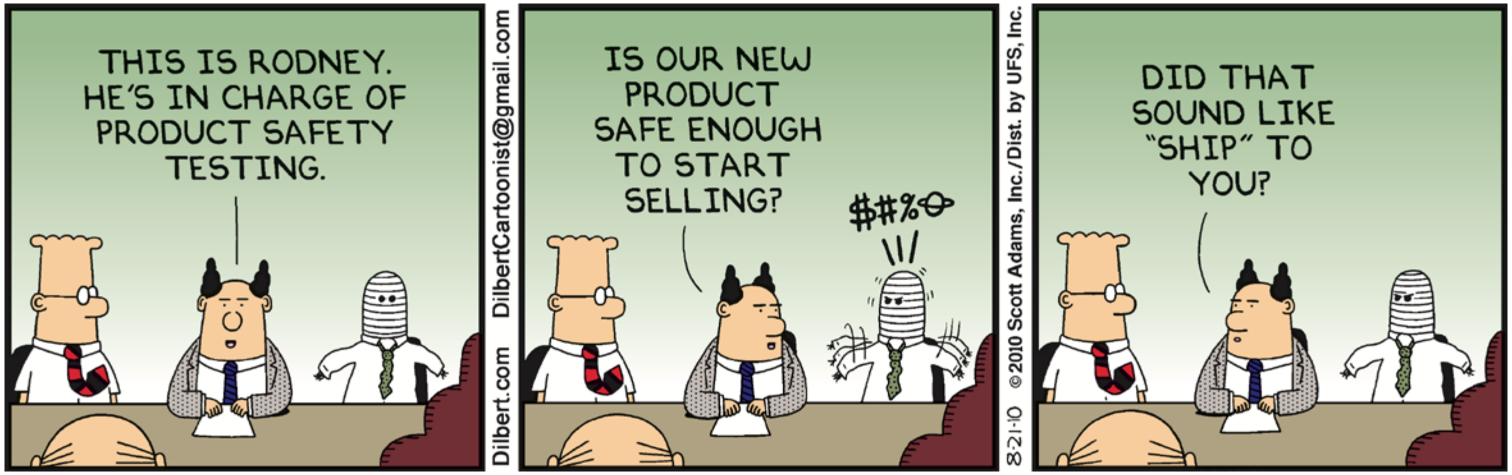
Where?

- ARM uses SVN internally (for now).
- git is a better choice for external collaboration.
- There are many online git repositories to choose from.
  - ARM maintains a presence on both Github, and a enterprise hosted Gitlab server (code.arm.gov).
- Use git. There is a learning curve. I promise it is worth it.
  - The standard git workflow (Commit often, branch, pull request) simplifies software development and collaboration.
  - Use it even if it's just you.
  - E-mail me. I'll provide resources to learn it.


WHAT ARE YOU WORKING ON?
TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...

Advice on Structure of Code

- Isolate code into at least three components.
  - Reading, Writing, Processing
- Writing will be handled by ADI. Use common structures such as hash tables to store data.
- Top level interface.
- Use an appropriate style standard (Google, pep8, etc)
- Handle edge cases. Real data is messy.
- Make paths configurable. Generally good deployment practices.


THIS IS RODNEY. HE'S IN CHARGE OF PRODUCT SAFETY TESTING.
IS OUR NEW PRODUCT SAFE ENOUGH TO START SELLING?
$#%⊕
DID THAT SOUND LIKE "SHIP" TO YOU?
Dilbert.com DilbertCartoonist@gmail.com
8-21-10 ©2010 Scott Adams, Inc./Dist. by UFS, Inc.

Example Collaborative Development: Before Handoff

- You want to develop a fancy new precipitation VAP based on gauge corrected radar data.
- First steps:
  - Develop algorithm and test it.
  - Publish it.
  - Run a batch of data through, store output for reference dataset.

- In preparation for handoff to developers:
  - Develop list of inputs
    - Will it work for any data level? Any radar? All gauges? Does sampling time matter?
    - Remember ARM operational configurations can change.
  - Handle edge cases, missing data, missing data streams.
    - If gauge is missing, still do QPE from radar only?
  - List output variables, and their associated metadata

Example Collaborative Development: Interaction with Developer

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

- Back and forth with developer
  - They are not domain experts, but usually have some knowledge in the area.
  - Primary job is to translate your algorithm into ARM infrastructure.
  - Developer will put algorithm into ADI, work on data flow, generally operationalize.
  - This will be a back and forth process.
- Reference files
  - They will generate a set of files to ensure modifications to output data can be tracked and known.
- Metadata
  - ARM has somewhat strict metadata standards. It can be a painful process but is useful to end users.
- Evaluation Area (Limited beta period)
- Release

A few tips

- Pareto Principle (80/20 rule).
- Why submit VAP?
  - Fame, Fortune, and more realistically citations for a paper.
- Take the messiest dataset you can find, realize the real data will likely be worse (At least at times)
- Don't fail gracefully, fail early.
  - Use assertions.
- Use exceptions and failure codes to signify abnormal conditions, don't just blindly crash.

- A good references:
- "Effective Computation in Physics" – Anthony Scopatz, Kathryn D. Huff