* Model development workflow
  + Workflow vs products (idea taken from openscience slides)
  + Workflow includes:
    - Editor you use to write code
    - Code you ran earlier
    - Name of home directory
  + Product includes:
    - Raw data
    - Packages used
    - Code someone needs to run to get your results
    - Figures and code used to make those figures
    - Functions
  + While aspects of everyone’s workflow can be different (ie what ide you use to run code), the products shouldn’t be and anyone should be able to produce the same products despite differences in their workflows.
  + Main point: *Don’t want to hard code anything about workflow into your products*
* Openscience – many federal agencies, including NOAA have announced this year is the year of open science
  + “Open Science is the principle and practice of making research products and processes available to all, while respecting diverse cultures, maintaining security and privacy, and fostering collaborations, reproducibility, and equity”
  + Think of “open-ness” as a spectrum
    - Open with yourself
    - Open with co-workers (people on that project) – makes everyone feel included in the project and good for peer-review
      * We should get in the practice of peer-reviewing each other’s code
      * Allows people to jump from one aspect to another easier without having to spend a lot of time getting caught up or waiting for someone to send them all the files they need
    - Open with other SAP members (not directly working on project), which might look like sharing code to do certain analyses (ie CPUE standardization).
    - Open publically – eventually products will be this
  + Ok to share messy or incomplete code
    - Don’t wait to share until its perfect (push to github)
* Tools to use for openness
  + Google drive
  + Github repositories
    - Private
    - Public
  + Groups and trainings – openscapes and NMFS R User Group
* Why we should use an automated workflow
  + saves time
  + saves energy – not having to re-write code every time, reuse when you can
    - need to make code flexible and generic
  + makes onboarding easier – for the next person doing the assessment
  + for future you – helps you remember exactly what you did when you go back to the code a few months or even a couple of years later
  + makes things uniform
  + can put your energy towards developing the model and not towards the little details
  + Main point: *automate as much as possible!!!!*
    - Don’t waste your time doing the tedious tasks bc you have more important stuff to do than resizing figures
* What mistakes it can help avoid
  + Reduces the chance of making small mistakes that can have big impacts
  + If there are systemic mistakes, this can be an issue but once you find it and fix it, it’s easy to fix and re-run everything
* why SAP should go in this direction
  + anyone can download our repo and run the models themselves
  + made it a lot easier during the review to run alternate scenarios
  + can share the responsibility (this also means there isn’t one failpoint)
  + share knowledge within the group (and with others outside of the group as well)
* How we implemented it in American Samoa:
* Challenges of AmSam Assessment
  + This assessment had some unique challenges to it. The main one being we had 2 people working on 9 separate models at the same time. Also, this was the first time SS was being used for these species so we didn’t have an existing framework to build on.
  + Things we realized early on were:
    - The models were fairly simple and would be structured the same for each species. 🡪 because of this we were able to write custom functions to input the data and parameter values into the input files
    - Because we were both working on the models at the same time and still testing parameter values, we needed a way we could adjust the values and re-run models in real time. 🡪 set up 2 google sheets that we could change in real time and pull info directly into R
    - Want to be able to share quick summary of model versions with people without them having to download the repo and run the models on their computer. 🡪 We included pdf reports of summary plots and diagnostic results for each model version.
* Our workflow
  + We used google drive and github to contain pretty much everything we needed. Google drive is good for keeping big files like the raw data and things we both needed to access and be able to change frequently.
  + The only thing that was kept exclusively on our personal computer was the processed data. We didn’t push this to the repository because of size issues and privacy. We would download the latest raw data folder from the google drive, run all of the data processing scripts and get the outputs in a separate folder that were ready to be put directly into the model.
  + On the github repository, we kept all of the code needed to run every step, the 4 input files to run each model version as we worked on developing the base case and alternate models. And we also kept model summary reports. We also tracked issues and tasks that needed to be done and could assign them to whoever was responsible for that task.
  + I’ll go into more detail on each component but this is a broad overview of how everything worked together.
* Repository structure
  + Included some instructions in the readme on how to use the repo so when anyone goes back to it, they know where things are and what they need to run
  + 3 main folders – scripts, ss3 models, and ss3 final models
  + Scripts:
    - Subfolders for data processing scripts, model building scripts, and scripts to create figures and tables for the report.
    - Additionally, we had our 2 main scripts that ran everything, 01\_Run data scripts.R and 02\_Run SS models.R. Within these 2 scripts we were able to call all of the other scripts and run the functions.
  + SS3 models:
    - Where we kept the different versions of models for each species as we developed them. Each new thing we wanted to implement or test, would get its own subdirectory.
  + SS3 final models:
    - Where we kept the final base case model and the alternate scenarios we ran for each one. We also had some special figures and tables just for the alternate runs. These files will reproduce what is in the final assessment report.
* 01 Raw Data – kept raw data in folder on google drive
  + Used code to determine if we had latest version on our computer and if not download and unzip newest version to replace old
  + Run data processing scripts to get model input (cleaned and processed data)
* 02 Process data
  + Run all of the data processing scripts one at a time
* 03 Set inputs
  + We used a separate script to run the models
  + At the top we created a list of argument inputs that we rarely changed. Included the species name, the life-history parameters we wanted to use, the range of values to profile over, if we wanted to use superyears and what years to include, and the range of projected catch to forecast over.
  + Further down, inside the larger lapply function, we set other key arguments that were changed more frequently. They include, the name of the directory we wanted to run everything in, turn on and off running the model, printing the summary reports, and running diagnostics. Also, if we wanted to run bootstraps or forecasting. Another thing we played around with was changing the start year so we included that as an argument that we could easily change.
* 04 Build models and run
  + Our core function was build\_all\_ss which was a wrapper for several smaller functions that would write new control, data, starter, and forecast files, run diagnostics, print plots and summary reports. We could also run forecasts and bootstraps.
  + I mentioned earlier that we kept google sheets with parameter inputs so the code here shows how we actually pulled those values straight from google drive into R. We also accounted for occasions when we wouldn’t be able to use google drive but had the spreadsheets downloaded on our computers. This was another argument we could change.
  + Really leverage the googlesheets4 r package to connect directly to google drive through r
* Modular Code – separate functions to do each component
  + One of the strengths of this workflow was how we modularized all of our code. This helped keep everything succinct, easy to follow, easy to debug, and quicker to run.
  + Write SS files
  + Run SS
  + Run diagnostic tests
  + Run bootstrap
  + Run forecasting
  + Creating figures and tables
  + Use parallel processing to run multiple models
* 05 Produce model diagnostics reports
  + Used the flexibility of Quarto
  + We created a customized qmd report with model diagnostic plots for model fits, runs test, likelihood profiling, retrospectives, etc.
  + Made it easier to track model development, share progress, and discuss models with others.
* 06 Generate formatted figs and tables
  + When we were putting together the results into the report, we found ourselves having to go into different folders and find a bunch of different figures and tables, and resize or format them to look like how we wanted it. This might not have been so bad if we only had 1 species to do this for but we had to do each thing 9 times.
  + Likely if you are copy and pasting the same thing more than 3 times, you can (and should) write code to automate that. So using quarto, we created a report that brought in all the figures we needed and formatted the tables exactly how we wanted them to look and then exported it all in a word doc that we could then put right into the report.
  + Flextable and officer are really good packages for doing this. Here’s some code for how we formatted an excel table.
  + First set some defaults for the table, font and size, border color. Then convert the dataframe into a flextable object.
  + The table that I’m showing here was one with a bunch of reference points, which meant a lot of the text needed special styling like subscript or superscript or italicizing. Stuff that doesn’t normally get transferred when going from R to excel to word.
  + With flextable you can specify all of this. This chunk is adding some borders to the header row. And the next chunk is showing examples of how to format the text with special styles.
  + That’s how we got a table that looks like this in excel to one that looks like this in word with no work on our part.
* lessons we learned in our development
  + make sure you use packages that work with all platforms of R
  + want to make functions flexible enough to give you options but not worry about coding every single possibility
  + develop as you go, our structure changed from our initial ideas but as we worked we streamlined it
  + make code modular
* small things to implement in your code now to reduce friction later
  + more comments
  + read other people’s code (you can learn from their code, new functions, style, etc.)
  + make sure you are using relative paths and including all of the packages you used in that script
  + when naming scripts, give them informative names and try sequencing them
    - or make a note at the top if a different script needs to be run first
  + get in the habit of syncing code with repository regularly
* Discussion
  + Is this system useful for single-species assessments?
  + How can we help each other with setting these systems up?
  + How can this be useful for international assessments?
  + What about reference manager software?
    - I didn’t touch on this at all, and we honestly didn’t include it in our process, but that could be another avenue that we start looking at.
    - Sharing papers more easily for projects
  + Github for project management