effective strategies for writing proposal work plans for research software

Chase Million

10⁶C Million Concepts

some bonafides of the author

- 20 years of experience in research software support roles
- 12 years as CEO of a research software and consulting company
- PI of multiple software-heavy research grants
- Written and co-written many proposals. Reviewed hundreds of proposals across multiple agencies
- Some major software projects include:
    - research software / support roles on Mars Exploration Rovers (Spirit and Opportunity), Mars Science Laboratory (Curiosity), Mars 2020 (Perseverance)
    - lead software engineer of the Galaxy Evolution Explorer imaging pipeline.
    - the planetary data reader (pdr), a Python library to support read operations on ~2PB of legacy observational data of the solar system from NASA missions

RSE Stories interview: `https://us-rse.org/rse-stories/2022/chase-million/`

thesis: work plans make or break proposals

- In my experience, review panels rarely get hung up on the question of "is this worth doing?"
- Panels instead get hung up on the questions of "is this feasible?" and "will the expense or effort plausibly lead to success?"
- Unfortunately, scientists typically receive no training in how to make (or judge) useful, plausible work plans or estimates.
    - But they get *a lot* of training on how to describe the context and importance of scientific questions or controversies…
    - so the "background" and "justification" sections end up dominating proposals.

the whole secret to a compelling work plan

1. have a plan
2. describe it

steps to making a plan

1. scoping
2. requirements
3. estimation
4. scheduling

straightforward (but not easy!) scoping and estimation

- Play somewhat fast and loose with the formal project management standards.
- Use task decomposition as the basis for estimation.
- This approach is most appropriate for small-ish, novel projects (i.e. <5 people, original research). There are better (and more difficult) estimation strategies for other situations.

project scoping part one - vision & scope

- Figure out what you want to do.
- Identify the people who have the problem and / or who will be involved in creating a solution.
- State the problem—as well as hard boundaries of possible solutions—in words that all stakeholders agree to.
- Ideally ~2 sentences; no longer than 2 paragraphs.
- This is nearly equivalent to the "elevator pitch" for your proposal. If you cannot convince the review panel that you are addressing a real problem / need within <2 paragraphs, it is probably a poorly conceived project.

project scoping part two - concept of operations

- Interview intended "users" of the software.
- From the users' perspectives:
  - describe the current situation and critical features of the new system / solution
  - describe the proposed system
  - make sure that all users agree that their needs are represented accurately

project scoping part three - requirements specification

- List: what set of conditions would achieve the project vision?
- Rank them by priority: critical, stretch, nice-to-have
- Cross out any requirement that is not critical — these are not requirements
- Requirements are what a system must *do*, not what it must *be*.
- The usefulness of your project estimate is absolutely limited by the completeness of your requirements.
- The completeness of your requirements rests entirely on the accuracy of your vision & scope / conops. An excellent requirements specification for the wrong solution solving the wrong problem is worse than useless.

common categories of software requirements

- functionality
- accuracy and correctness
- reproducibility
- performance
- infrastructure
- deployment modes
- reliability
- maintainability
- usability
- interfaces
- ecosystem constraints
- quality attributes
- security

project estimation part one - work breakdown structure

- Think of a system or set of systems that addresses all requirements, aka "a solution."
- Break the implementation into tasks and sub-tasks.
- Each unit action should be perceived as achievable.
- Not too many units: 10-20 for a mid-sized project
- Keep solutions open-ended when possible; put off making decisions.
- Verify completeness: if all tasks are completed, it will meet all requirements
- Check that commonly missed categories of tasks are represented (or justify their absence).

commonly forgotten task categories

- exploring low-level implementation approaches and prototyping
- optimizing code
- documenting code; writing documentation, user guides, tutorials
- data organization or cleanup
- data and software archiving
- creating tests and test data
- other validation activities (e.g. comparing to published results)
- IT management
- software packaging and delivery
- disseminating results (e.g. talks, conferences, papers, etc.)
- project management, including periodic re-scoping and re-estimating
- communicating within the team (incl. informal and formal meetings / discussions)
- communicating outside the team (with end users, upper managers, sponsors)
- managing bug reports and feature requests

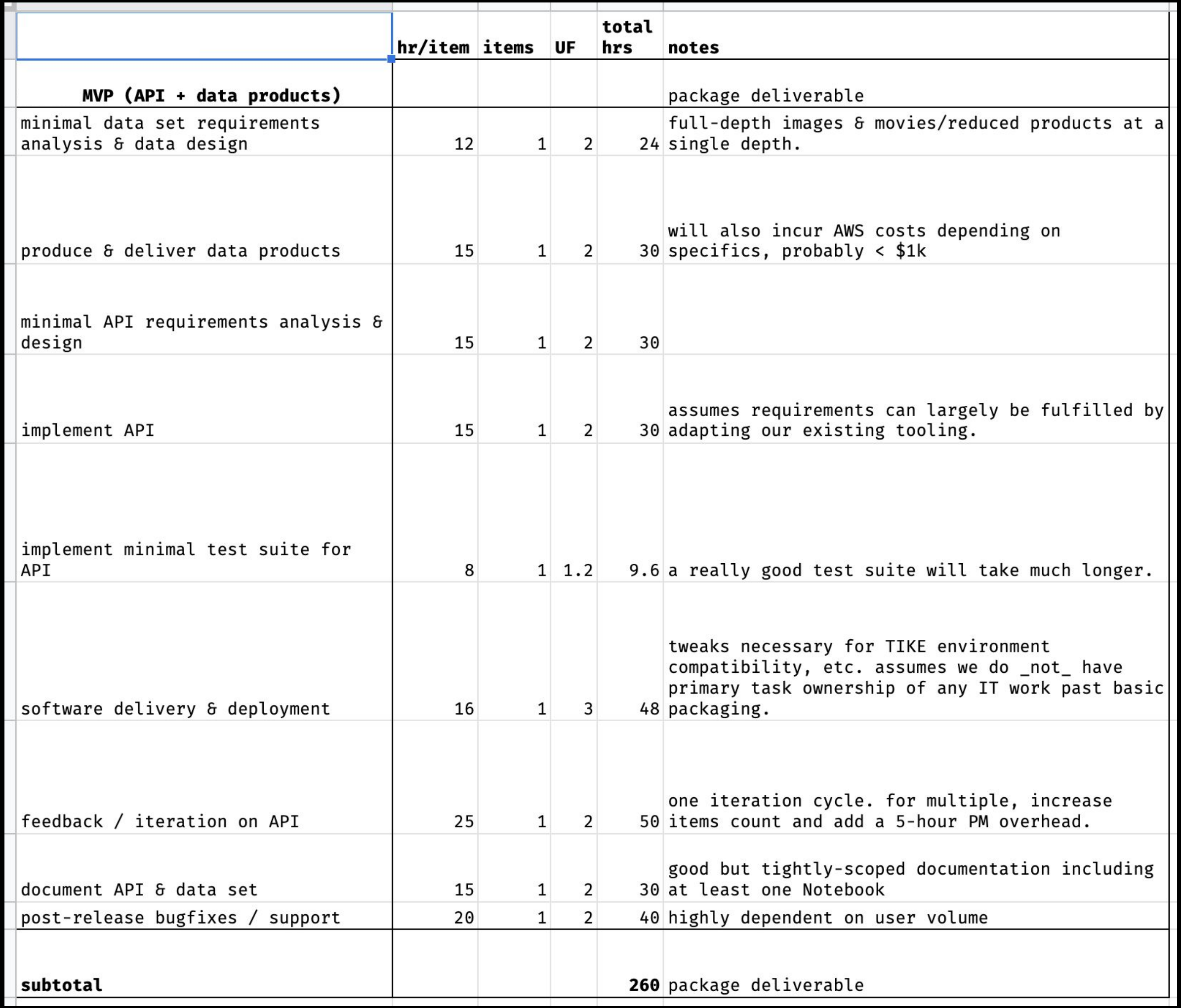
project estimation part two - estimation

- Judge the amount of time, in hours of billable effort, that it will take to complete each task. If the people who will do the work are available, they should do this for themselves.
- Assign a confidence factor of between 1.2 and 4 to each time estimate, based on the prior experience of the people who do the work. A factor of 1.2 should be used for tasks that are extremely similar to work that they have done before. A 4 is for work that is entirely new.
- The sum of outputs from (3) is the *best case scenario* for your project, the amount of resources that would be required if everything went exactly to plan. The sum of the pairwise products of (3) and (4) is the *project estimate*, or the resources that are sufficient and reasonable to complete the work.

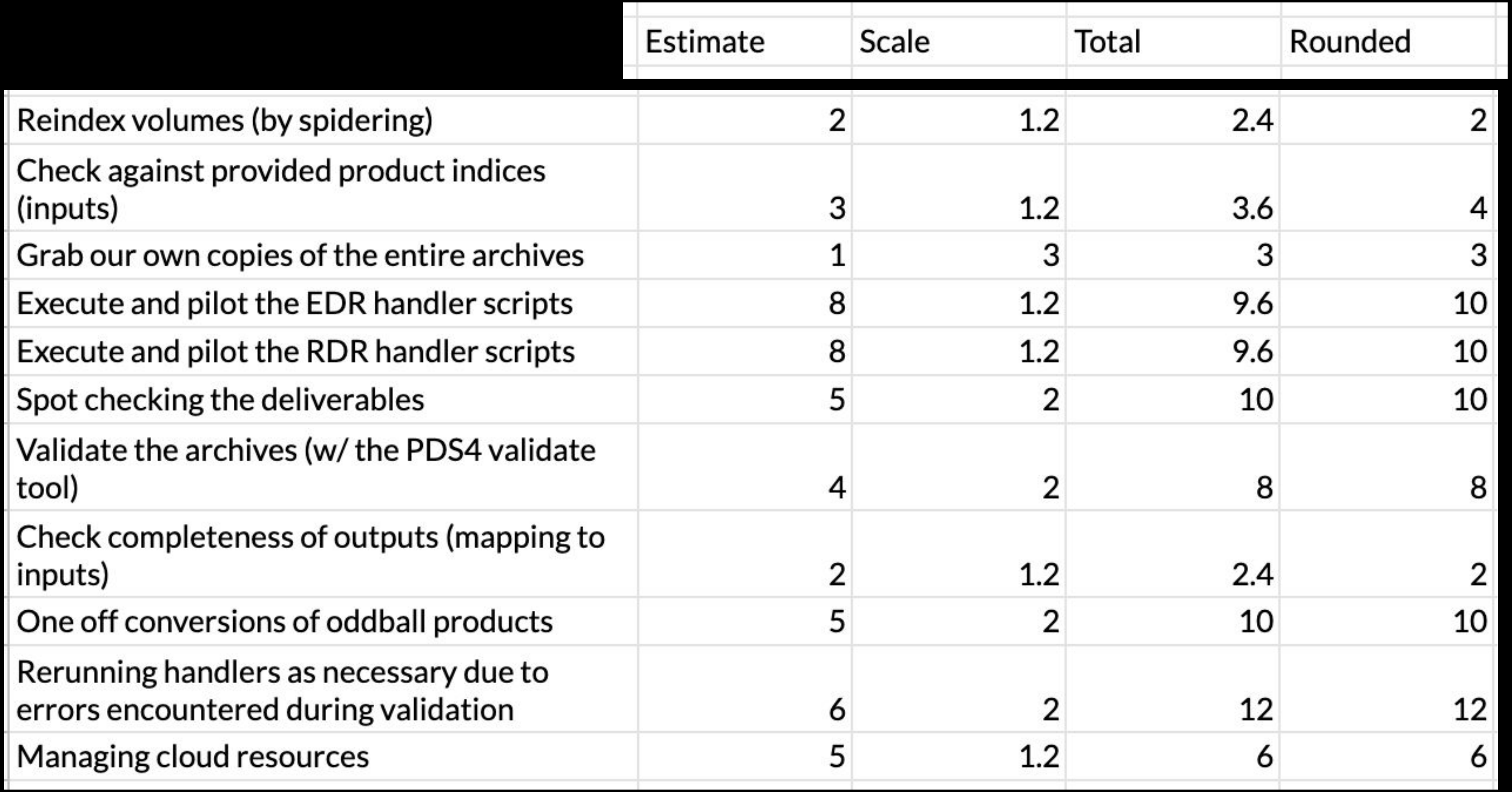
scale factors

1.2 : Deep domain expertise and prior experience doing very similar tasks. It is not exactly 1 because *everything will take longer than you expect*.
2 : Deep domain expertise but do not have prior experience doing very similar tasks. This is probably the most common scale factor in research projects.
3 : Some domain expertise, but have open questions about how to best approach and successfully complete the task. Not as easy as two; but not as uncertain as four.
4 : Confident that the task is achievable, but almost no idea how.

Note: These apply to the person(s) who will be doing the work. How long it would take someone else to do your work is irrelevant, unless you can get them to do it for you.



	hr/item	items	UF	total hrs	notes
MVP (API + data products)					package deliverable
minimal data set requirements analysis & data design	12	1	2	24	full-depth images & movies/reduced products at a single depth.
produce & deliver data products	15	1	2	30	will also incur AWS costs depending on specifics, probably < $1k
minimal API requirements analysis & design	15	1	2	30	
implement API	15	1	2	30	assumes requirements can largely be fulfilled by adapting our existing tooling.
implement minimal test suite for API	8	1	1.2	9.6	a really good test suite will take much longer.
software delivery & deployment	16	1	3	48	tweaks necessary for TIKE environment compatibility, etc. assumes we do _not_ have primary task ownership of any IT work past basic packaging.
feedback / iteration on API	25	1	2	50	one iteration cycle. for multiple, increase items count and add a 5-hour PM overhead.
document API & data set	15	1	2	30	good but tightly-scoped documentation including at least one Notebook
post-release bugfixes / support	20	1	2	40	highly dependent on user volume
subtotal				**260**	package deliverable



	Estimate	Scale	Total	Rounded
Reindex volumes (by spidering)	2	1.2	2.4	2
Check against provided product indices (inputs)	3	1.2	3.6	4
Grab our own copies of the entire archives	1	3	3	3
Execute and pilot the EDR handler scripts	8	1.2	9.6	10
Execute and pilot the RDR handler scripts	8	1.2	9.6	10
Spot checking the deliverables	5	2	10	10
Validate the archives (w/ the PDS4 validate tool)	4	2	8	8
Check completeness of outputs (mapping to inputs)	2	1.2	2.4	2
One off conversions of oddball products	5	2	10	10
Rerunning handlers as necessary due to errors encountered during validation	6	2	12	12
Managing cloud resources	5	1.2	6	6

the estimation algorithm

0. Is this a relatively small software project as part of a grant-funded research effort? If yes, proceed.
1. Define the project objectives clearly, in language that every stakeholder understands and agrees on. At minimum, this should include a vision and scope document. It can optionally include concept of operations and requirements specification documents.
2. Conceive of an approach that meets the requirements. Break it into sub-projects or tasks, each small enough that you can conceive of completing it successfully.
3. Judge the amount of time, in hours of billable effort, that it will take for the people who do the work to complete each task. If those people are available, they should do this for themselves.
4. Assign a confidence factor of between 1.2 and 4 to each time estimate, based on the prior experience of the people who do the work. A factor of 1.2 should be used for tasks that are extremely similar to work that they have done before. A 4 is for work that is entirely new to them.
5. The sum of outputs from (3) is the *best case scenario* for your project, the amount of resources that would be required if everything went exactly to plan. The sum of the pairwise products of (3) and (4) is the *project estimate*, or the resources that are sufficient and reasonable to complete the work.

caveats re: estimation

- All estimates are wrong; the quality metric of an estimate is whether it is useful.
- An estimate is not the basis of a negotiation. Never alter the estimate to hit a benchmark. Change the scope of work or high-level implementation and re-estimate, or change the benchmark (e.g. get more resources).
- Re-estimate frequently once the project is underway. This will help you catch many problems earlier.

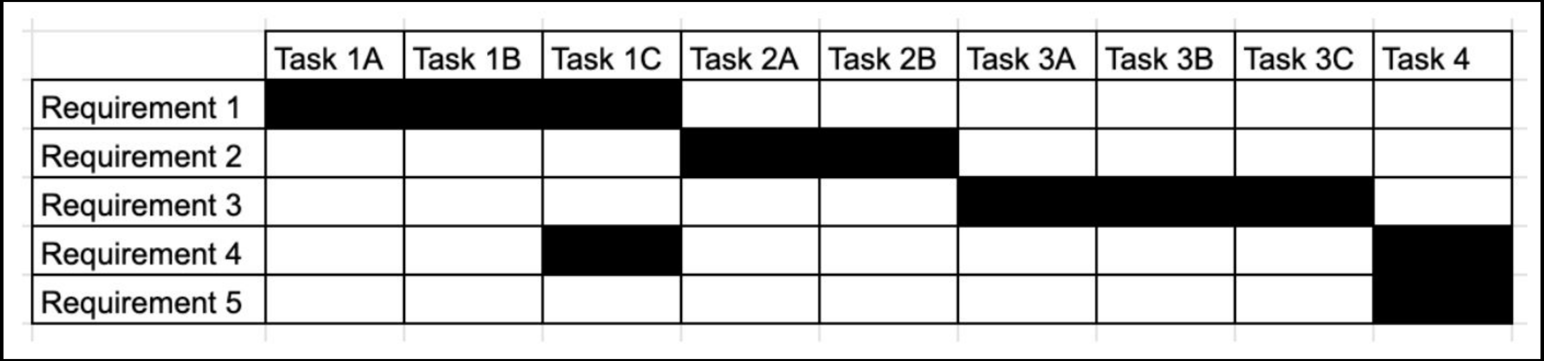
putting it all into the proposal

- The tasks and subtasks of the WBS create natural section and subsection headings; they can be consolidated for clarity.
- A few graphical artifacts—a task matrix and Gantt chart—will hammer home that you have thought carefully about the work and what it entails.
- There are specific software tools for creating these charts, but you can also just use any spreadsheet program.

task matrices

Tie your project objectives / requirements directly to specific tasks.

requirements traceability matrix (RTM)



	Task 1A	Task 1B	Task 1C	Task 2A	Task 2B	Task 3A	Task 3B	Task 3C	Task 4
Requirement 1	■	■	■						
Requirement 2				■	■				
Requirement 3						■	■	■	
Requirement 4			■						■
Requirement 5									■

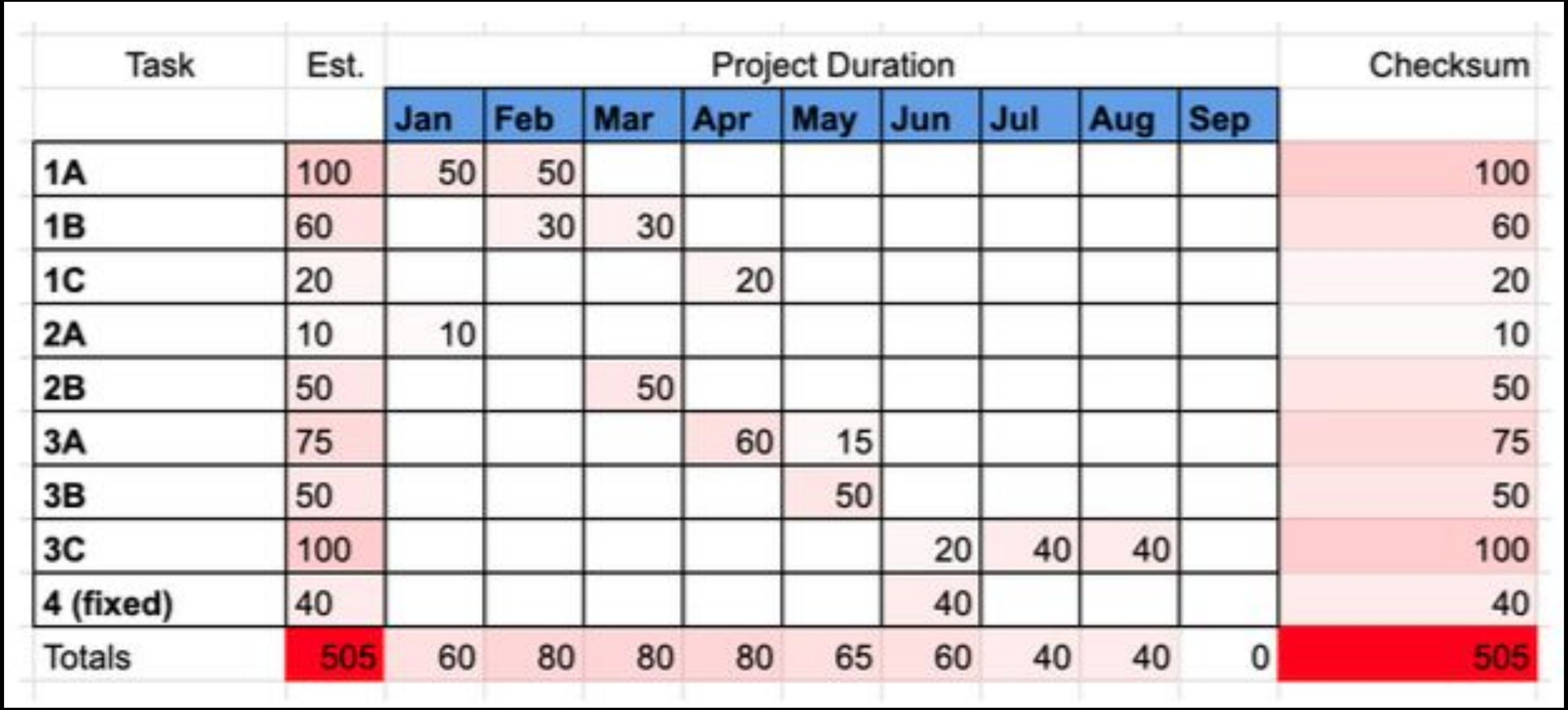
It might make more sense, within the narrative of a proposal, to do a "science traceability matrix" (STM), tying scientific objects to tasks or requirements

See description of STMs by Sabrina Feldman at the NASA PI Launchpad Workshop (Nov. 9, 2019):
https://science.nasa.gov/science-pink/s3fs-public/atoms/files/Launchpad_Session3_STM_18Nov2019_smf_final.pdf
and Jared Leisner at the NASA PI Launchpad Workshop (June 15, 2021):
https://science.nasa.gov/science-pink/s3fs-public/atoms/files/PI%20Launchpad--STM--JLeisner%20210615.pdf

gantt charts

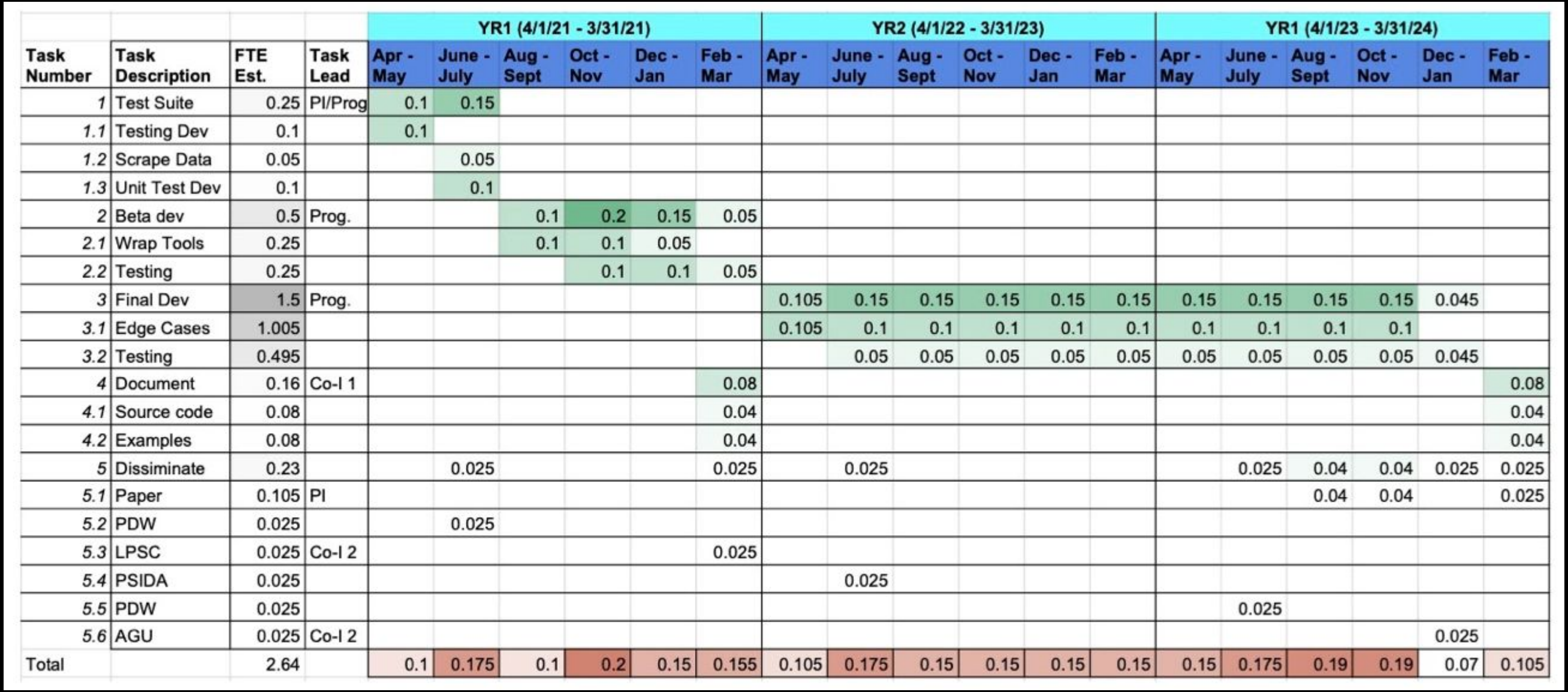
- Show distribution of work effort across task and time
- Gantt charts primarily work off calendar time, not billable time; the conversion between the two requires careful consideration.
- Gantt charts can also include information about personnel or task ownership.

example gantt chart — hypothetical project



Task	Est.	Project Duration									Checksum
		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	
1A	100	50	50								100
1B	60		30	30							60
1C	20				20						20
2A	10	10									10
2B	50			50							50
3A	75				60	15					75
3B	50					50					50
3C	100						20	40	40		100
4 (fixed)	40						40				40
Totals	505	60	80	80	80	65	60	40	40	0	505

example gantt chart — real project

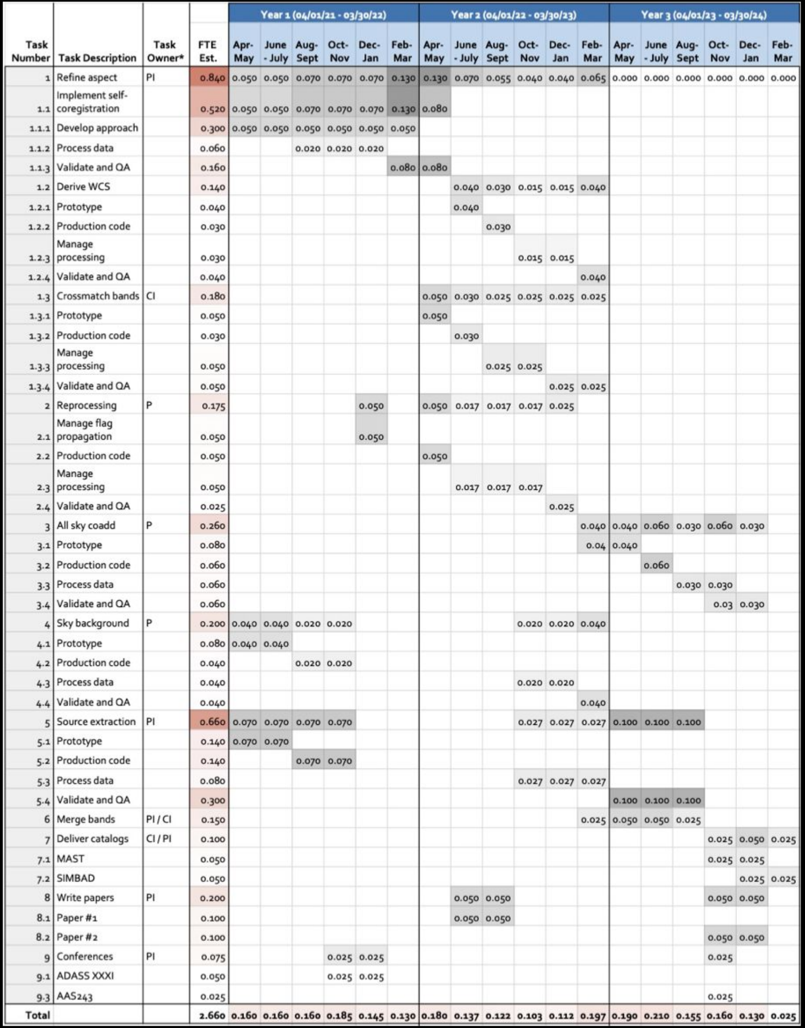


				YR1 (4/1/21 - 3/31/21)						YR2 (4/1/22 - 3/31/23)						YR1 (4/1/23 - 3/31/24)					
Task Number	**Task Description**	**FTE Est.**	**Task Lead**	**Apr - May**	**June - July**	**Aug - Sept**	**Oct - Nov**	**Dec - Jan**	**Feb - Mar**	**Apr - May**	**June - July**	**Aug - Sept**	**Oct - Nov**	**Dec - Jan**	**Feb - Mar**	**Apr - May**	**June - July**	**Aug - Sept**	**Oct - Nov**	**Dec - Jan**	**Feb - Mar**
1	Test Suite	0.25	PI/Prog	0.1	0.15																
1.1	Testing Dev	0.1		0.1																	
1.2	Scrape Data	0.05			0.05																
1.3	Unit Test Dev	0.1			0.1																
2	Beta dev	0.5	Prog.			0.1	0.2	0.15	0.05												
2.1	Wrap Tools	0.25				0.1	0.1	0.05													
2.2	Testing	0.25					0.1	0.1	0.05												
3	Final Dev	1.5	Prog.							0.105	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.045	
3.1	Edge Cases	1.005								0.105	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1		
3.2	Testing	0.495									0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.045	
4	Document	0.16	Co-I 1						0.08												0.08
4.1	Source code	0.08							0.04												0.04
4.2	Examples	0.08							0.04												0.04
5	Dissiminate	0.23			0.025				0.025		0.025						0.025	0.04	0.04	0.025	0.025
5.1	Paper	0.105	PI															0.04	0.04		0.025
5.2	PDW	0.025			0.025																
5.3	LPSC	0.025	Co-I 2						0.025												
5.4	PSIDA	0.025									0.025										
5.5	PDW	0.025															0.025				
5.6	AGU	0.025	Co-I 2																	0.025	
Total		2.64		0.1	0.175	0.1	0.2	0.15	0.155	0.105	0.175	0.15	0.15	0.15	0.15	0.15	0.175	0.19	0.19	0.07	0.105

example gantt chart

Note that:

- Total work effort matches the estimate.
- Work effort is distributed somewhat evenly across time. No big spikes.
- Some tasks are fixed in time (like conferences).
- Some tasks are fixed in sequence, e.g. they are in the paths of other tasks.



				Year 1 (04/01/21 - 03/30/22)						Year 2 (04/01/22 - 03/30/23)						Year 3 (04/01/23 - 03/30/24)					
Task Number	Task Description	Task Owner*	FTE Est.	Apr-May	June - July	Aug-Sept	Oct-Nov	Dec-Jan	Feb-Mar	Apr-May	June - July	Aug-Sept	Oct-Nov	Dec-Jan	Feb-Mar	Apr-May	June - July	Aug-Sept	Oct-Nov	Dec-Jan	Feb-Mar
1	Refine aspect	PI	0.840	0.050	0.050	0.070	0.070	0.070	0.130	0.130	0.070	0.055	0.040	0.040	0.065	0.000	0.000	0.000	0.000	0.000	0.000
1.1	Implement self-coregistration		0.520	0.050	0.050	0.070	0.070	0.070	0.130	0.080											
1.1.1	Develop approach		0.300	0.050	0.050	0.050	0.050	0.050	0.050												
1.1.2	Process data		0.060			0.020	0.020	0.020													
1.1.3	Validate and QA		0.160						0.080	0.080											
1.2	Derive WCS		0.140								0.040	0.030	0.015	0.015	0.040						
1.2.1	Prototype		0.040								0.040										
1.2.2	Production code		0.030									0.030									
1.2.3	Manage processing		0.030										0.015	0.015							
1.2.4	Validate and QA		0.040												0.040						
1.3	Crossmatch bands	CI	0.180							0.050	0.030	0.025	0.025	0.025	0.025						
1.3.1	Prototype		0.050							0.050											
1.3.2	Production code		0.030								0.030										
1.3.3	Manage processing		0.050									0.025	0.025								
1.3.4	Validate and QA		0.050											0.025	0.025						
2	Reprocessing	P	0.175					0.050		0.050	0.017	0.017	0.017	0.025							
2.1	Manage flag propagation		0.050					0.050													
2.2	Production code		0.050							0.050											
2.3	Manage processing		0.050								0.017	0.017	0.017								
2.4	Validate and QA		0.025											0.025							
3	All sky coadd	P	0.260												0.040	0.040	0.060	0.030	0.060	0.030	
3.1	Prototype		0.080												0.04	0.040					
3.2	Production code		0.060														0.060				
3.3	Process data		0.060															0.030	0.030		
3.4	Validate and QA		0.060																0.03	0.030	
4	Sky background	P	0.200	0.040	0.040	0.020	0.020						0.020	0.020	0.040						
4.1	Prototype		0.080	0.040	0.040																
4.2	Production code		0.040			0.020	0.020														
4.3	Process data		0.040										0.020	0.020							
4.4	Validate and QA		0.040												0.040						
5	Source extraction	PI	0.660	0.070	0.070	0.070	0.070						0.027	0.027	0.027	0.100	0.100	0.100			
5.1	Prototype		0.140	0.070	0.070																
5.2	Production code		0.140			0.070	0.070														
5.3	Process data		0.080										0.027	0.027	0.027						
5.4	Validate and QA		0.300													0.100	0.100	0.100			
6	Merge bands	PI / CI	0.150												0.025	0.050	0.050	0.025			
7	Deliver catalogs	CI / PI	0.100																0.025	0.050	0.025
7.1	MAST		0.050																0.025	0.025	
7.2	SIMBAD		0.050																	0.025	0.025
8	Write papers	PI	0.200								0.050	0.050							0.050	0.050	
8.1	Paper #1		0.100								0.050	0.050									
8.2	Paper #2		0.100																0.050	0.050	
9	Conferences	PI	0.075				0.025	0.025											0.025		
9.1	ADASS XXXI		0.050				0.025	0.025													
9.3	AAS243		0.025																0.025		
Total			2.660	0.160	0.160	0.160	0.185	0.145	0.130	0.180	0.137	0.122	0.103	0.112	0.197	0.190	0.210	0.155	0.160	0.130	0.025

general proposal tips and strategies

- The work plan should compose at least 30% of the page allotment.
- Reviewers will be exhausted and looking for cognitive shortcuts. They will probably not parse prose carefully or accurately. Give them clear tables and charts to hang onto.
- For any tables and charts or graphics of any kind, make sure that the font is approximately the same size as the body font of the proposal if not larger.
- Verify that there is consistent terminology between the graphics and body text.
- Use consistent organizational structure for the work plan and body text; major project tasks should have names that are echoed (and maybe bolded / underlined or used as section headings) in the body text.

general proposal tips and strategies

- Every proposal should have a gantt chart. It's worth taking a whole page for it.
- Make sure that all tasks are assigned to at least one team member and that all budgeted team members are assigned to tasks.
- You <u>must assume</u> that reviewers do not have even a general understanding of project management ideas, let alone this specific approach. Therefore, include 1-2 paragraphs explaining the method used to generate the work plan and estimates.

recommended reading – project planning / estimation

- "Software Estimation: Demystifying the Black Art" by S. McConnell
- "Software Estimation Without Guessing: Effective Planning in an Imperfect World" by G. Dinwiddle
- IEEE Std. 830-1998 "IEEE Recommended Practice for Software Requirements Specifications"
- "a practical guide to research software project estimation" by C. Million

```
https://github.com/MillionConcepts/software_project_management
```

- "Strategies for research software project estimation" by C. Million

```
https://www.youtube.com/watch?v=Ks0zOa4Z5bM
```

recommended reading – proposal preparation

- "Science in Action: How to Follow Scientist and Engineers Through Society" by B. Latour
- "Visual Explanations" (and everything else) by E. Tufte
- "Thinking Fast and Slow" by D. Kahneman
- "Influence: The Psychology of Persuasion" by R. Cialdini
- "The Missing README: a guide for the new software engineer." by C. Riccomini and D. Ryaboy

Chase Million
Million Concepts
chase@millionconcepts.com
github.com/millionconcepts

This research was supported by the Better Scientific Software Fellowship, part of the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.