Agile stock assessment

Akira Hayashi, Momoko Ichinokawa, Junji Kinoshita, Akihiro Manabe

Fisheries Resources Institute, Japan Fisheries Research and Education Agency, 2-12-4 Fuku-ura, Kanazawa ward Yokohama, Kanagawa, Japan 236-8648

Abstract

Stock assessment determines the status of fishery stock to support management decision making. Considering the iterations between exploratory calculations and the need to compare outputs to elicit better model settings, we should focus on not only the accuracy of abundance estimation and the tolerance of uncertainty but also the efficiency of the project workflow. Although in Japan, a stock assessment model was introduced written in the R language in 2012, the workflow did not sufficiently adjust, creating problems because the current workflow is contrary to the principles of effective value creation. To make our project sustainable, we propose adopting the agile methodology, an iterative development method used by software developers, for stock assessment. Therefore, we wrote an example report as a package document in the R language. Developed under a continuous integration environment, the report remains up to date, with every modification on component files. This method enabled our work to be efficient and transparent by allowing and documenting scenario branching, error corrections, and annual updates. We show that the iterative development cycle benefit us by allowing us to focus on the essential business problem of the assessment project.

Keywords: agile, waterfall, build, continuous integration, reproducible

1. Introduction

Stock assessment determines the status of fishery stock to support management decision making. Since calculation results affect management scenarios, the accuracy of abundance estimation and tolerance for uncertainty in future estimations are important. Although assessment models have been improved (reviewed in Methot (2009)), there continuously are new requirements for further improvements to model features. However, considering the characteristics of stock assessment—which requires iterative work between exploratory calculations and the comparison of outputs to elicit better model settings—technical improvements to model features are insufficient. To better contribute to management, an increase in the efficiency of our project is critically needed.

Japanese stock assessment introduced an assessment model written in the R language R Core Team (2020) from 2012 Ichinokawa et al. (2019); Ichinokawa and Okamura (2014), However, an incomplete shift in the workflow has been causing problems. Although the production speed of the upstream process has increased thanks to the new assessment model, the processing speed of the downstream process remains low: we still manually paste graphs, tables and type numbers into a report written in word processing software with a graphical user interface (GUI) Hayashi et al. (2019). This mismatch Sharma (2017) causes information congestion and lowers project productivity Huang et al. (2014, 2015). Manual operation of the GUI word processor impairs not only the speed of iteration but also the quality of the product because manual operations are not reproducible Gandrud (2020); effort is wasted in recovering the required quality Kano et al. (1984) of the product at every iteration. The poor productivity of our project is inevitable because our workflow is completely contrary to lean manufacturing Womack and Jones (2010); Earley (2016), an effective method of value creation. To make our project sustainable, we must adopt a better method.

Software development professionals are good at sustainable project management, as they recognized the importance of iterative work for project success early Royce (1987) and evolved their developmental method from a slow sequential approach (waterfall methodology, Sommerville (1996)) to quick iterative approach (agile methodology, Williams and Cockburn (2003)). To keep documentation consistent with software behavior, software developers have started generating documents from the source code rather than writing it by hand Forward and Lethbridge (2002) because they know that documentation that is inconsistent with actual behavior lowers project productivity Kajko-Mattsson (2005); Fluri et al. (2007); Ibrahim et al. (2012); Shi et al. (2011); Nuseibeh et al. (2000). Considering that stock assessment projects require both calculations and documentation, there is much to learn from software developers. Even in parallel development, developers keep the directory clean because they work in an isolated workspace utilizing the :“branch” function of the version control system instead of duplicating files Zolkifli et al. (2018); This approach is also applicable to exploratory scenario branching in stock assessment. Additionally, their way of using the branch function for debugging was also useful to us. They also focus on converting their tacit knowledge to written programs because they see computer

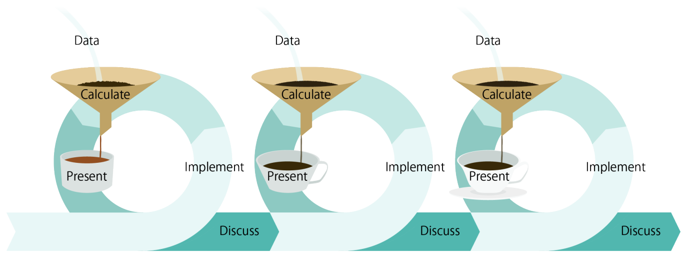


Figure 1: Conceptual scheme of the agile methodology for stock assessment

programs as a type of communication de Souza et al. (2005).

Having considered the example of software developers, it is apparent that we should manage stock assessment projects in a sustainable manner. The key to adopting agile methodology (Fig. 1) is abandoning the GUI word processor because it forces each process to be run in sequential order—figures, tables, and calculation results cannot be pasted into the document until the calculation is completed; inevitably, the subsequent processes, such as writing documents, cleaning formats, and discussions on the results, will be blocked (Fig. 2). The adoption of agile methodology enables assessment projects to adapt to frequent changes in requirements Fitzgerald (2000); Roy et al. (2018) during scientific discussions. Making our project buildable is also significant from the viewpoint of reproducible science LeVeque et al. (2012).

In this article, we explain how to develop stock assessment as a buildable software product under an agile methodology. We discuss how we developed an assessment report as part of the software and performed a complete

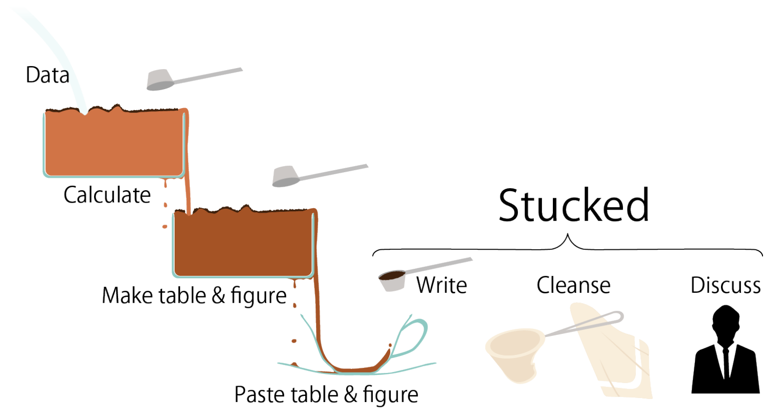


Figure 2: Conceptual scheme of the waterfall-like workflow for stock assessment due to GUI word processor

process to run at every modification. Our method enabled our work to be efficient and transparent by supporting scenario branching, error corrections, and annual updates. We discuss the effect of the proposed method on stock assessment and demonstrates its significance by relating it with methods in previous studies.

2. Materials and methods

As done in agile methodology, we started our work with the development of a minimum viable product (MVP): a software product that is rough in appearance but works properly Lenarduzzi and Taibi (2016). In our case, the MVP is a buildable assessment project from data inclusion to document generation with tentative model settings. An important practice in developing our MVP is continuous integration (CI), which is a system that builds software at every modification Duvall et al. (2007).

The essential components of the CI system are i) source codes, ii) the build system, iii) the version control system, and iv) the monitoring system. As an example of stock assessment, sources include both calculations and documentation because the report should contain both the calculation results and narratives. The sources for calculation include files that are needed to reproduce calculation, such as data and source codes written in specific programming language. Speaking of sources for documentation, we have to choose one system that supports the inclusion of external calculation results. $\TeX$ has been the de facto standard in the field of technical reporting for a long time. The second element is a server that builds all source files into a product; Thus, the server requires the programming language and documentation system to be installed. All build steps should be written in a build script such as `Makefile`. The version control system, the third element, is a tool to manage the modification history of every file Swierstra and Löh (2014). The final element is a monitoring system that triggers building in linkage with the version control system. Jenkins [1] is a widely used software for this use. The other option available is hosting services such as Travis CI [2], Circle CI [3], GitHub Actions [4], and so on.

Although we can set up a CI document system independent of programming language, We think R language and its general document system Rmarkdown Allaire et al. (2020); Xie et al. (2018) are good candidates. Using R language, we can integrate the sources for calculation and documentation into a single `.Rmd` file with the deployment process automated by a GitHub workflow using the R package `pkgdown`. The GitHub workflow renders our `.Rmd` into `.html` and deploys the product to the website. Here, tables are supported by a package `knitr`, so writing additional code snippets to generate tables is not necessary. These features will provide scientists engaging stock assessment with a good writing experience regardless of the programming language they use for the main calculation.

2.1. Buildable assessment project under CI system

To present a document consistent with the calculation, we bundled our sample report as a software package written in R language. In general, there

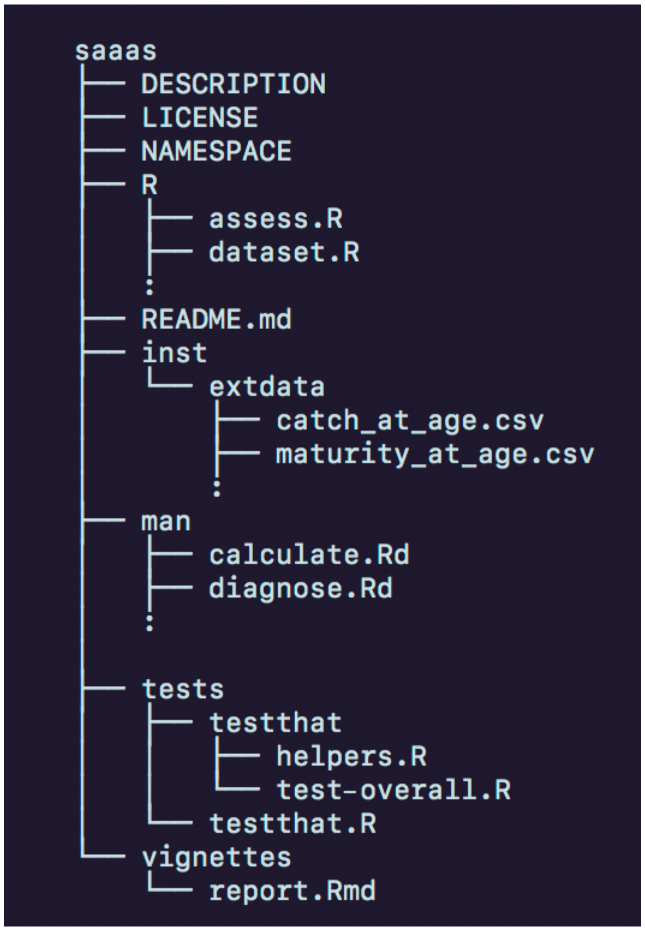
[1]`https://www.jenkins.io`
[2]`https://travis-ci.com`
[3]`https://circleci.com`
[4]`https://github.com/features/actions`

are two types of motivation to develop the R package: i) to provide end-users specific calculation routines and ii) to robustize the work of the developer's own; The motivation of this work corresponds to the latter. This article does not instruct the basic method to develop the R package because a detailed instruction is available Hadley (2015). Instead, we explained how to deal with situations frequently occurring in reporting stock assessment—handling scenario branching, debugging after discussion, and managing annual updates.

We implemented our project following the basic directory structure of the R package (Fig. 3). An Rmarkdown file `report. Rmd` was created under directory `vignettes/`. The report can be built by the function `build()` of package `devtools`. In the following section, we express the call of `some_function()` of the R package `some_package` as `some_package::some_function()`.

Following test-driven development Beck (2003), we started the implementation by writing test code; we wrote tests for `load_data()`, `calculate()`, `diagnose()`, and `visualize()` under directory `test/testthat/`, which were initially failed because the functions were non-existent. After specifying the behavior of the functions within the test, we implemented these four functions under the directory `R/` to pass the tests. After confirming that the new functions passed all the tests, we called them in `report. Rmd` following functional programming: R objects were not created because all functions are relayed by pipe operators. Our actions in this project are expressed by the program itself, written in the report—`load_data()`, `calculate()`, `diagnose()`, and `visualize()`.

Every time we call `devtools::build()`, all tests are executed, and the report is updated, with the most recent calculation results embedded.



```
saaas
├── DESCRIPTION
├── LICENSE
├── NAMESPACE
├── R
│   ├── assess.R
│   ├── dataset.R
│   ⋮
├── README.md
├── inst
│   └── extdata
│       ├── catch_at_age.csv
│       ├── maturity_at_age.csv
│       ⋮
├── man
│   ├── calculate.Rd
│   ├── diagnose.Rd
│   ⋮
│
├── tests
│   ├── testthat
│   │   ├── helpers.R
│   │   └── test-overall.R
│   └── testthat.R
└── vignettes
    └── report.Rmd
```

Table 1: Definition of terms in the version control system

Term	Definition
commit	to register a modification to files in the version control system
push	to send commits to the repository
merge	to apply a change to one specific branch(es) to another
revert	to cancel modification of the specific commit(s)
rebase	to transplant one specific branch onto another branch

Continuous integration systems were established to automate the build process. We chose Git [5] for the version control engine. To store all the code, we created a repository named `saaas` (Stock Assessment As A Software) Hayashi (2020) on the Git hosting service GitHub [6]. After "committing" our work, we "pushed" it to the repository (Table 1). The builds ran on a virtual Microsoft Azure machine [7] because we used GitHub Actions to manage the CI system. We chose GitHub Pages [8] to host the build result.

At this point, we have succeeded in creating our MVP. Although our MVP is too simple for actual use, we can now improve it in an iterative manner because every modification to the source files will be reflected in the report hosted on GitHub Pages.

[5] `https://git-scm.com`
[6] `https://github.com`
[7] `https://docs.github.com/en/actions/reference/virtual-environments-for-github-hosted-runners`
[8] `https://pages.github.com`

2.2. Issue tracking system

All work should be based on issues Kaminski et al. (2008), which are a list of ideas, problems, enhancements, tasks, and bug reports. Every issue should be managed by focusing on the following points: i) when the issue is recognized and ii) how is it resolved. Thus, we need an issue tracking system to manage tasks for the improvement of our product. In this paper, we used GitHub as an issue tracking system.

In the following sections, we explain our work within each iteration. Here, we use the term “iteration” to express the following set of work steps done in an issue-driven manner Hall et al. (2008): i) define issue and create a new branch to resolve it, ii) update code, iii) commit and push, and iv) apply modification to the repository after comparison with the original.

3. Iterations

3.1. First iteration: scenario branching

In the first iteration, we will show how scenario branching can be handled in a continuous documentation workflow. The first example issue of our project is “What will happen to stock abundance if the assumption on recent selectivity is modified?”

3.1.1. Materials and methods

The aim of this section is to handle branched scenarios with different settings. To modify the source code independently of the original version, we use the branch function of Git. The branch, like a branch of a tree from the main stem, is a parallel world of the workspace originating from a certain point in the original workspace. From `master`, the original branch, we created three branches—`branch_a`, `branch_b`, and `branch_c`—and modified the model settings on each branch. After committing the modifications, we push them to GitHub.

To enable comparison between branches, we have to upload the build results elsewhere; We used the Simple Storage Service (S3) of Amazon Web Service (AWS) for the output storage. We modified `visualize()` to export the calculation output as `result.csv` and created a GitHub Actions workflow to upload `results.csv` to S3 at every build.

To compare these results with the original scenario, we updated `visualize()` to draw calculation results for each scenario. The modified version of `visualize()` showed the difference between the current calculation and the result generated from specific branches. We also updated the GitHub Actions workflow to upload the rendered `report.html` to S3.

Next, we updated the argument of `visualize()` to compare the outputs with those from `master` and then opened a pull request from each branch to `master`. Due to the time order of the work, we compared all four branches on the pull request from `branch_c` to `master`. On the pull request referencing the current issue, we discussed the comparison and determined which scenario to choose, accepting `branch_a` and rejecting the other scenarios.

3.1.2. Results and brief discussion

The branch system enables us to handle multiple scenarios without cluttering the working directory. In the pull request, we can review not only the difference in code between scenarios but also conversations regarding the request to merge (Table 1) scenarios. We can trace how the project has evolved to the current state; the branch network shows how many scenarios have been considered and why each scenario was accepted or rejected. Thus, our method made the decision-making process traceable in future retrospective projects.

From the viewpoint of artifact management van der Hoek et al. (1997), the generated files should maintain a one-to-one correspondence with their source code. To avoid human error originating from manual operations, we ignored build artifacts such as `report.html` and `results.csv`; we committed only source files such as `report. Rmd` and R code under `R/`. Without manual operation, these artifacts were generated on the build server and sent to cloud storage for scenario comparison. By utilizing CI, we can focus our effort on the essential intellectual work.

3.2. Second Iteration: bugfix

What should we do if we find some bugs that affect all of the scenarios compared in the last iteration? Do we have to fix the bugs on all branches by hand?— The branch system also provides an effective solution in this situation. To simulate the situation, assume that there is a bug in the catch data: the demo issue for this iteration is “Fix a bug in catch data”.

3.2.1. Materials and methods

To recompare the scenarios after bugfix, we reverted (Table 1) the merge commit on the `master` branch, which brings the status back to the earlier status before merging (Fig 4). Following the conventional approach with bugfix, we created a branch named `hotfix` from `master`. On `hotfix`, we added a test to detect the bug, confirmed that the new test breaks the build,

doi:10.20944/preprints202010.0474.v1

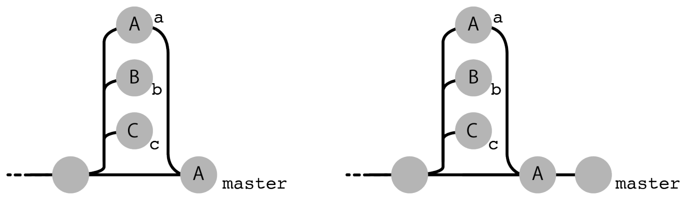


Figure 4: Reverting the merge commit on `master` to cancel the modification added by `branch_a`

and then corrected the catch data to fix the build (Fig. 5, left). After fixing the bug on `master` by merging the pull request from `hotfix` to `master` (Fig. 5, middle), we rebased (Table 1) the three scenario branches on `master` to apply the same bugfix (Fig. 5, right). We force-pushed the rebased branches because the default command cannot push a branch with a revised modification history. Referring to the current issue number, we reopened the pull requests to `master` from each scenario branch and recompared the build outputs. Following the same procedures followed in the previous iteration, `branch_a` was accepted and merged into `master`.

3.2.2. Results and brief discussion

Using the rebase function of Git, we enabled reassessment of the scenario branches after bugfix was applied. Although merging could also be used to apply bugfix to the scenario branches, rebasing is appropriate in this

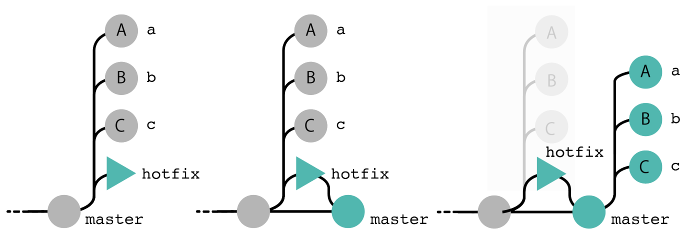


Figure 5: Bugfix of scenario branches by rebasing

situation because it can overwrite commit history: if we had used the merge function to apply bugfix, the final commit of each branch would be bugfix and would mask the important commit comparison on the pull request (Fig. 6). Rebasing kept the branch network clean and rational and enabled us to focus on the essential differences between branches.

3.3. Third Iteration: clarify tacit knowledge

We continued to update the stock assessment. However, we had hard-coded the value of the argument `year` inside the software but still had not clarified our tacit knowledge on the consistency of years between the data and the calculations: our calculation uses data from one year prior to the assessment, but this “knowledge” cannot be executed because it is not expressed as working code. Thus, the system is not fully self-documented in terms of its maintainability between years. The example issue for this iteration is “clarify the intended one-year lag between data and calculations.”

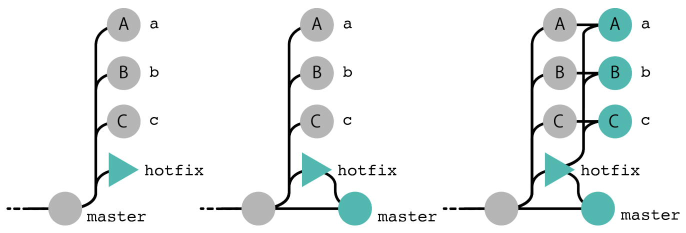


Figure 6: Complicated commit history after merging `hotfix` into scenario branches

3.3.1. Materials and methods

We created a function `metadata()` that contains the metadata of the project such as species name, stock name, and year. Then, we added a test that confirms the consistency between the years of the data and those of the metadata. Since our calculation uses data from one year prior to the assessment, we had to confirm that the lag in years between the data and the metadata equals one.

We simplified our functions using `metadata()`: we replaced the argument `year` of `fcurrent_years()` and `terminal_f_years()` with `metadata()` for the year reference; then, the function `calculate()` became free from the argument `year`.

Next, we implemented tests for species name and year of the `metadata`, which work as system tests.

3.3.2. Results and brief discussion

By adding system tests, we managed to confirm the overall consistency of the software. Whenever we run `devtools::test()`, all tests are run, and the result confirms that everything has gone well.

Implementing such a system test will drive future updates to be consistent; when we incremented the test year to `2019` to simulate an update in the next year, the test failed because the year given to the test is different from that in the `metadata()`. To fix this test, we have to increment the year in the `metadata()`. Following this fix, another test fails because the year values for the `metadata()` and those for the data differ. To fix this test, we have to add data from 2019. After all the fixes, when all the tests as passed, project updating is completed with quality assurance.

4. Discussion

In the present study, we showed how to develop a stock assessment project as software. Because we implemented the report as a buildable document under the CI system, every update to the source files triggered a build on a cloud computer that uploaded the report in html format to online storage. This keeps our document up to date with the source code and data.

In the present paper, we followed the standard development procedure for generic software and the R package Hadley (2015). The standardized procedure helped us to maintain a clean project. We also implemented modularized functions under a directory with a standard structure. The organization of the program and directory was maintained because we branched our scenario using a version control system. Issue-driven iterative development

Hall et al. (2008) made our work traceable as it related to specific issues. The branch-based workflow allowed us to compare multiple scenarios on pull requests and to review all decision-making processes; it also helped us to correct fundamental defects even after scenario branching. By clarifying our tacit knowledge about the year lag between data and assessment, we made our software maintainable for future updates.

The method we propose can not only speed up the calculation-communication cycle of stock assessment but also ensure the stability of the project. To support sustainable fishery management, we, the scientists engaging in stock assessment, also need to make our working methods sustainable.

4.1. Lean documentation

The essence of stock assessment is value creation. Spotlighting this aspect of our role, we applied lean manufacturing Womack and Jones (2010); Earley (2016), a proven method in many business domains, to the documentation of stock assessment. Despite originating in the car industry, lean manufacturing has been applied to other fields of manufacturing, including software development Alves et al. (2015, 2014) due to its general effectiveness for value creation. Our method complies with the five principles of lean manufacturing: (1) specify value, (2) identify the value stream, (3) flow, (4) pull, and (5) pursue perfection.

The value of a project must be distinguished from its product; it is the customer satisfaction provided by the product. Taking the car industry as an example, the value is not the car itself but the pleasurable experience brought by driving the car. The value that stock assessment provides to management is a smooth decision-making process, which is provided by the product: a comprehensible assessment report. All of our work contributes to customer value because we arranged the value stream, the focus of the second principle.

The value stream is all the actions that are needed to provide value to the customer. At the beginning of the first iteration, we built a CI documentation system that converts our work to improve customer value. Our only task is to edit source code, commit our work, and push it to GitHub. Committing is not a task without value; it works as an input control for the quality of the work. Once the modification is registered as a commit, there is no room for human error because all subsequent processes are automated.

The third principle "flow" is about keeping the process moving at a steady pace from start to finish Earley (2016). This principle emphasizes the importance of a steady, not an as-fast-as-possible, pace. Our CI system, which stops product delivery if even a small typo exists in the code, follows the flow principle. From a myopic viewpoint, a workflow that does not allow any defects may seem vulnerable compared to that based on the GUI word processor, but our CI system prevents a pile of small deficits from building up after several time lags and causing severe problems; bugs should be removed at the moment they are introduced because the cost of debugging increases with time. A successful build proves that the issue, which arose from user requirements, was converted into value without problems.

The fourth principle, "pull", is achieved by our coding style. To eliminate the goods-in-process that do not contribute to value, every process should be suspended until the next production process pulls its result. The techniques that contribute to these principles are test-driven development and functional programming. Test-driven development is a technique to start coding from a test to ensure that the code fails the test properly Beck (2003). Although the famous benefit of test-driven development is that it helps clarify the behavior of the program component, it has another benefit in terms of lean production: the failing test we wrote at the beginning of each iteration works as a "pull", telling us that the latter process requires the new function. This method prevents us from writing not only programs without tests but also unnecessary programs. Functional programming—a technique that designs a program to use the return of other function rather than an object state—also commits to the pull principle. Without creating any R object in the report, we converted data into value with functions in a single flow using a pipe operator. This means that we constructed our program based on "pull" from the previous function. Programs with minimum side effects keep us from cluttering our directory with goods-in-process files.

The last principle, "perfection," is a continuous "kaizen" attitude towards user satisfaction. Although our report is simplified for demonstration, it can be improved without degradation of the product quality because the framework we created enables our direct commitment to the user value. To increase project productivity, it is necessary to not only increase the speed of improvement but also avoid regressions. We also suppressed the recurrence of a bug by adding a test. These small but steady incremental improvements are necessary for project stability.

The workflow proposed here is not the sole solution for every project: workflows should be designed considering the context of the project, such as the skill levels of those involved and the available tools. Nevertheless, following the lean principles helped us to design an effective workflow, and it has proven to be successful in varying business domains Alves et al. (2014).

4.2. Lessons from the history of software development

Our novel approach to stock assessment was influenced by our knowledge of software development. The more our work relies on the computer, the more similar the problems we face become to those that software developers have already addressed. When we stuck to the area of our own expertise, we were unable find a good exemplar to solve our problem. Although the field of software development varies, software developers have compiled excellent methodologies for information management because they focus on abstracting problems into their essence. With this ability, software developers can employ knowledge beyond the field of their expertise. The practices of developers such as CI offer general advantages to work that involves computers: repeatability, reliability, recoverability, and so on. Such "ilities" **?** allow us to focus on essential business problems.

Developing reports under the CI system will bring benefits to not only scientists but also model programmers. When trying to resolve technical problems such as model features and interfaces, we need effective communication; miscommunication during the specification is one of the major factors lowering project productivity Kortum et al. (2016). Behavior-driven development Pereira et al. (2018); Scandaroli et al. (2019) is recognized to be a solution to this issue. Communication based on documents generated by the CI system and its source code works as form of behavior-driven development, and it helps to avoid miscommunication because everyone sees exactly the same behavior in the rendered report on the cloud server.

Our method provides benefit in the context of knowledge management Lethbridge and Skuce (1992). Code is not only a source of value creation but also a form of communication between the project members de Souza et al. (2005) because the source code is the only accurate description of the software de Souza et al. (2005); McConnell (2004). Since literate programming enables reporting based on the source code, the writing procedure as a visible code is repeatable because it is executable by any person. As a result, making the procedure visible helps find errors as well.

We have tacit expectations for not only our procedure but also our data. We can make our expectations visible by writing tests that allow us to focus on unusual data as well, such as unbelievably low or high amounts. The accumulation of explicit knowledge leads to the building of a knowledge base, which will have strong appeal in the case of peer review of stock assessment.

The standardization of the method eliminates project-specific rules and methodologies. We developed our program as an R package, which standardizes our program and makes it comprehensible among the R user community. Standardization distinguishes essential differences in business knowledge between projects. Although our method required us to learn how to develop an R package, the technique can be applied to other projects and bring the benefit of organized workflows.

4.3. Future perspective

In this article, we defined our user as management. We confess that this definition was so arbitrary that we premised the development of a report as our product. The lean principle suggests that the proper definition of the end user is indispensable for business success; without proper end-user definition, value is distorted by the middle of the organization because it adds complexity of no interest to the end-user Womack and Jones (2010). For successful value creation, we must realize who our true end-user is.

Stock assessment is part of a stratified business activity: fishery scientists give management an assessment report to support smooth decision making; management gives the fishing industry a fishing policy to provide sustainable fisheries. Hence, our true end-users are those people who expect sustainable fishery management to maintain their fishery business and consumption activity. In other words, the scientists engaging in stock assessments, the people in fishery management, and the people working in the fishery industry are the co-creators of sustainable fisheries.

Considering our true purpose, value co-creation, the tool we need is not a report, but a business intelligence (BI) tool de Carvalho et al. (2016); von Hippel (2001) that supports discussion between co-creators. Putting into words, the BI tool in stock management will provide stock assessment as a service. This expectation is inferred by the trend of servicification in the software business, which has persisted since the 2010s. Although the closest example is the “Assessment Tools” [9] hosted by the International Council for the Exploration of the Sea (ICES), no BI tool targeting managemental discussion has been developed at this time. BI tool platforms such as the R package Shiny [10] and the commercial service Tableau [11] will help us build our own BI tools, and cloud computing services such as AWS and Google Cloud

[9]`https://www.ices.dk/data/assessment-tools/Pages/default.aspx`
[10]`https://shiny.rstudio.com`
[11]`https://www.tableau.com`

Platform [12] will provide us with an environment for establishing microservice architectures from scratch. If our future BI tool includes narratives, it should be served by an independent MS service; this will solve a weakness of the proposed method: the time cost of the entire build is triggered by a small modification of narratives.

We recognize the imperfection of a report as a product for higher end-user experience. However, as a small step towards change in the business model, we introduced a method to resolve the methodological problems of the current stock assessment project. Not only technical but also cultural aspects are a major problem in the transformation of workflows Pechau (2011); even in the software industry, the migration to agile methodology is difficult for projects traditionally managed by the waterfall methodology Kusters et al. (2017); Theocharis et al. (2015); Kuhrmann et al. (2017); Theobald and Diebold (2018); Clear (2003). Although build-based documentation requires a cultural shift for projects away from using the GUI word processor, the investment will pay off because stock assessment is a continuous project. The method can be applied to all projects, not only stock assessment but also other research activities.

References

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., Iannone, R., 2020. rmarkdown: Dynamic

[12]`https://cloud.google.com/gcp/`

Documents for R. URL: https://github.com/rstudio/rmarkdown. r package version 2.3.

Alves, Anabela, C., Kahlen, F.J., Flumerfelt, S., Siriban-Manalang, Anna, B., 2014. The lean production multidisciplinary: From operations to education, in: 7th International Conference on Production Research Americas 2014, Lima, Peru. URL: https://doi.org/10.13140/2.1.1524.0005, doi:10.13140/2.1.1524.0005.

Alves, Anabela, C., Sousa, Rui, M., Dinis-Carvalho, J., Moreira, F., 2015. Production systems redesign in a lean context: a matter of sustainability. FME Transactions 43, 344–352.

Beck, K., 2003. Test-driven Development: By Example. Kent Beck signature book, Addison-Wesley.

de Carvalho, D., Rocha, R., Fernandes, V., Neves, S., 2016. Business intelligence: Future perspectives (april, 2016), in: Proceedings of the Ninth International C* Conference on Computer Science &; Software Engineering, Association for Computing Machinery, New York, NY, USA. pp. 89–92. URL: https://doi.org/10.1145/2948992.2949011, doi:10.1145/2948992.2949011.

Clear, T., 2003. The waterfall is dead..: Long live the waterfall!! SIGCSE Bull. 35, 13–14. URL: https://doi.org/10.1145/960492.960500, doi:10.1145/960492.960500.

Duvall, P., Matyas, S., Duvall, P., Glover, A., 2007. Continuous Integration:

Improving Software Quality and Reducing Risk. A Martin Fowler signature book, Addison-Wesley.

Earley, J., 2016. The Lean Book of Lean: A Concise Guide to Lean Management for Life and Business. Wiley.

Fitzgerald, B., 2000. Systems development methodologies: the problem of tenses. IT & People 13, 174–185. URL: `https://doi.org/10.1108/09593840010377617`, doi:`10.1108/09593840010377617`.

Fluri, B., Wursch, M., Gall, H.C., 2007. Do code and comments co-evolve? on the relation between source code and comment changes, in: 14th Working Conference on Reverse Engineering (WCRE 2007), pp. 70–79.

Forward, A., Lethbridge, T.C., 2002. The relevance of software documentation, tools and technologies: A survey, in: Proceedings of the 2002 ACM Symposium on Document Engineering, Association for Computing Machinery, New York, NY, USA. pp. 26–33. URL: `https://doi.org/10.1145/585058.585065`, doi:`10.1145/585058.585065`.

Gandrud, C., 2020. Reproducible Research with R and RStudio. Chapman & Hall/CRC The R Series, CRC Press. URL: `https://books.google.co.jp/books?id=hDnSDwAAQBAJ`.

Hadley, W., 2015. R Packages: Organize, Test, Document, and Share Your Code. O'Reilly Media.

Hall, J., Rapanotti, L., Jackson, M., 2008. Problem oriented software engineering: Solving the package router control problem. IEEE Trans. Softw.

Eng. 34, 226–241. URL: https://doi.org/10.1109/TSE.2007.70769, doi:10.1109/TSE.2007.70769.

Hayashi, A., 2020. Stock Assessment As A Software. URL: https://doi.org/https://doi.org/10.5281/zenodo.4054164, doi:https://doi.org/10.5281/zenodo.4054164.

Hayashi, A., Kinoshita, J., Manabe, A., 2019. Stock assessment model in japan: future perspective, in: CAPAM workshop on the creation of frameworks for the next generation general stock assessment models, Wellington, New Zealand.

van der Hoek, A., Hall, R.S., Heimbigner, D., Wolf, A.L., 1997. Software release management, in: Proceedings of the 6th European SOFTWARE ENGINEERING Conference Held Jointly with the 5th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Springer-Verlag, Berlin, Heidelberg. pp. 159–175. URL: https://doi.org/10.1145/267895.267909, doi:10.1145/267895.267909.

Huang, F., Liu, B., Song, Y., Keyal, S., 2014. The links between human error diversity and software diversity: Implications for fault diversity seeking. Science of Computer Programming 89, 350 – 373. URL: https://doi.org/https://doi.org/10.1016/j.scico.2014.03.004, doi:https://doi.org/10.1016/j.scico.2014.03.004.

Huang, F., Liu, B., Wang, S., Li, Q., 2015. The impact of software process consistency on residual defects. Journal of Software: Evolution and Process 27, 625–646. URL:

https://doi.org/10.1002/smr.1717, doi:10.1002/smr.1717, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1717.

Ibrahim, W.M., Bettenburg, N., Adams, B., Hassan, A.E., 2012. On the relationship between comment update practices and software bugs. J. Syst. Softw. 85, 2293–2304. URL: https://doi.org/10.1016/j.jss.2011.09.019, doi:10.1016/j.jss.2011.09.019.

Ichinokawa, M., Nishijima, S., Okamura, H., 2019. Stock assessment model in japan: past to present, in: CAPAM workshop on the creation of frameworks for the next generation general stock assessment models, Wellington, New Zealand.

Ichinokawa, M., Okamura, H., 2014. Review of stock evaluation methods using vpa for fishery stocks in japan: Implementation with r. Bulletin of the Japanese Society of Fisheries Oceanography 78, 104–113.

Kajko-Mattsson, M., 2005. A survey of documentation practice within corrective maintenance. Empirical Software Engineering 10, 31–55. URL: https://doi.org/10.1023/B:LIDA.0000048322.42751.ca, doi:10.1023/B:LIDA.0000048322.42751.ca.

Kaminski, D.W., Hall, J.G., Wermelinger, M., 2008. Relating problem oriented engineering to current development processes: A research agenda, in: Proceedings of the 3rd International Workshop on Applications and Advances of Problem Frames, Association for Computing Machinery, New York, NY, USA. pp. 78–81. URL: https://doi.org/10.1145/1370811.1370827, doi:10.1145/1370811.1370827.

Kano, N., Seraku, N., Takahashi, F., Shin-ichi, T., 1984. Attractive quality and must-be quality. Journal of The Japanese Society for Quality Control 14, 147–156. URL: `https://ci.nii.ac.jp/naid/110003158895/en/`, doi:`10.20684/quality.14.2_147`.

Kortum, F., Klünder, J., Schneider, K., 2016. Miscommunication in software projects: Early recognition through tendency forecasts, in: International Conference on Product-Focused Software Process Improvement, pp. 731–738. URL: `https://doi.org/10.1007/978-3-319-49094-6_62`, doi:`10.1007/978-3-319-49094-6_62`.

Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., McCaffery, F., Linssen, O., Hanser, E., Prause, C.R., 2017. Hybrid software and system development in practice: Waterfall, scrum, and beyond, in: Proceedings of the 2017 International Conference on Software and System Process, Association for Computing Machinery, New York, NY, USA. pp. 30–39. URL: `https://doi.org/10.1145/3084100.3084104`, doi:`10.1145/3084100.3084104`.

Kusters, R., Leur, Y., Rutten, W., Trienekens, J., 2017. When agile meets waterfall - investigating risks and problems on the interface between agile and traditional software development in a hybrid development organization, pp. 271–278. URL: `https://doi.org/10.5220/0006292502710278`, doi:`10.5220/0006292502710278`.

Lenarduzzi, V., Taibi, D., 2016. MVP explained: A systematic mapping study on the definitions of minimal viable product, in: 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 112–119. URL: https://doi.org/10.1109/SEAA.2016.56, doi:10.1109/SEAA.2016.56.

Lethbridge, T.C., Skuce, D., 1992. Beyond hypertext: Knowledge management for technical documentation, in: Proceedings of the 10th Annual International Conference on Systems Documentation, Association for Computing Machinery, New York, NY, USA. pp. 313–322. URL: https://doi.org/10.1145/147001.147056, doi:10.1145/147001.147056.

LeVeque, R.J., Mitchell, I.M., Stodden, V., 2012. Reproducible research for scientific computing: Tools and strategies for changing the culture. Computing in Science Engineering 14, 13–17.

McConnell, S., 2004. Code Complete. Developer Best Practices Series, Microsoft Press.

Methot, R., 2009. Stock Assessment: Operational Models in Support of Fisheries Management. pp. 137–165. URL: https://doi.org/10.1007/978-1-4020-9210-7_9, doi:10.1007/978-1-4020-9210-7_9.

Nuseibeh, B., Easterbrook, S., Russo, A., 2000. Leveraging inconsistency in software development. Computer 33, 24–29. URL: https://doi.org/10.1109/2.839317, doi:10.1109/2.839317.

Pechau, J., 2011. Rafting the agile waterfall: Value based conflicts of agile software development, in: Proceedings of the 16th European Conference on Pattern Languages of Programs, Association for Computing Machinery, New York, NY, USA. URL: https://doi.org/10.1145/2396716.2396731, doi:10.1145/2396716.2396731.

Pereira, L., Sharp, H., de Souza, C., Oliveira, G., Marczak, S., Bastos, R., 2018. Behavior-driven development benefits and challenges: Reports from an industrial study, in: Proceedings of the 19th International Conference on Agile Software Development: Companion, Association for Computing Machinery, New York, NY, USA. URL: `https://doi.org/10.1145/3234152.3234167`, doi:`10.1145/3234152.3234167`.

R Core Team, 2020. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria. URL: `https://www.R-project.org/`.

Roy, D., Balszun, M., Heurung, T., Chakraborty, S., Naik, A., 2018. Waterfall is too slow, let's go agile: Multi-domain coupling for synthesizing automotive cyber-physical systems, in: Proceedings of the International Conference on Computer-Aided Design, Association for Computing Machinery, New York, NY, USA. URL: `https://doi.org/10.1145/3240765.3243500`, doi:`10.1145/3240765.3243500`.

Royce, W.W., 1987. Managing the development of large software systems: Concepts and techniques, in: Proceedings of the 9th International Conference on Software Engineering, IEEE Computer Society Press, Washington, DC, USA. pp. 328–338.

Scandaroli, A., Leite, R., Kiosia, A.H., Coelho, S.A., 2019. Behavior-driven development as an approach to improve software quality and communication across remote business stakeholders, developers and qa: Two case studies, in: Proceedings of the 14th International Conference on Global

Software Engineering, IEEE Press. pp. 95–100. URL: https://doi.org/10.1109/ICGSE.2019.00016, doi:10.1109/ICGSE.2019.00016.

Sharma, S., 2017. The DevOps adoption playbook : a guide to adopting DevOps in a multi-speed IT enterprise. Wiley, Indianapolis, IN.

Shi, L., Zhong, H., Xie, T., Li, M., 2011. An empirical study on evolution of api documentation, in: Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software, Springer-Verlag, Berlin, Heidelberg. pp. 416–431.

Sommerville, I., 1996. Software process models. ACM Comput. Surv. 28, 269–271. URL: https://doi.org/10.1145/234313.234420, doi:10.1145/234313.234420.

de Souza, S.C.B., Anquetil, N., de Oliveira, K.M., 2005. A study of the documentation essential to software maintenance, in: Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information, Association for Computing Machinery, New York, NY, USA. pp. 68–75. URL: https://doi.org/10.1145/1085313.1085331, doi:10.1145/1085313.1085331.

Swierstra, W., Löh, A., 2014. The semantics of version control, in: Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Association for Computing Machinery, New York, NY, USA. pp. 43–54. URL: https://doi.org/10.1145/2661136.2661137, doi:10.1145/2661136.2661137.

Theobald, S., Diebold, P., 2018. Interface Problems of Agile in a Non-agile Environment. pp. 123–130. URL: `https://doi.org/10.1007/978-3-319-91602-6_8`, doi:`10.1007/978-3-319-91602-6_8`.

Theocharis, G., Kuhrmann, M., Münch, J., Diebold, P., 2015. Is water-scrum-fall reality? on the use of agile and traditional development practices, pp. 149–166. URL: `https://doi.org/10.1007/978-3-319-26844-6_11`, doi:`10.1007/978-3-319-26844-6_11`.

von Hippel, E., 2001. Perspective: User toolkits for innovation. Journal of Product Innovation Management 18, 247 – 257. URL: `https://doi.org/https://doi.org/10.1016/S0737-6782(01)00090-X`, doi:`https://doi.org/10.1016/S0737-6782(01)00090-X`.

Williams, L., Cockburn, A., 2003. Guest editors' introduction: Agile software development: It's about feedback and change. Computer 36, 39–43. URL: `https://doi.org/10.1109/MC.2003.1204373`, doi:`10.1109/MC.2003.1204373`.

Womack, J., Jones, D., 2010. Lean Thinking: Banish Waste and Create Wealth in Your Corporation. Free Press.

Xie, Y., Allaire, J., Grolemund, G., 2018. R Markdown: The Definitive Guide. Chapman and Hall/CRC the R Series, Taylor & Francis, CRC Press, Boca Raton, Florida. URL: `https://bookdown.org/yihui/rmarkdown`.

Zolkifli, N.N., Ngah, A., Deraman, A., 2018. Version control system: a review. Procedia Computer Science 135, 408–415. doi:`https://doi.`

`org/10.1016/j.procs.2018.08.191`. the 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.