Replatforming Models

Eugene Chang and Leanne Sunter

2021-08-25

Contents

Preface

This guide has been developed based on examples of replatforming analytical models predominantly from Excel into R within the Department for Education. It is intended to be a living document so if you have any comments or suggestions please contact the Model Improvement and Assurance Unit.

0.1 Thanks

The authors would like to thank the following people for their contribution to this document.

- DfE: Rosie Amery, Nicky Brassington, Jack Shaw, Niall Taylor, Jon Tecwyn, Zach Waller.
- BEIS: Ian Mitchell, Aubrey Kendrick, Alec Waterhouse.
- GDS: Duncan Garmonsway.

In particular, Elliott White and Stuart Andrews at NAO for initial discussions.

Chapter 1

Introduction

Over recent years Government departments have increasingly been using open source programming languages to build and run mathematical models, as well as conduct data analysis. This move towards creating reproducible workflows has a range of benefits but has resulted in a situation where some models are now being run in this new environment, whilst other older and more established models are still being run and maintained via spreadsheets or proprietary software. In this document we provide guidance on how you can determine if it would be beneficial for you to re-platform your model and give practical tips and case studies on how you can approach the move into open source software.

1.1 Who should read this guidance?

This guidance is primarily aimed at analysts considering re-platforming a model, however stakeholders who are interested in understanding the benefits and risks of re-platforming may also find this helpful, particularly the Stakeholder FAQs section. It is specifically aimed at the task of rewriting a model in a new language, which has subtle, yet important differences from a regular data analysis project.

This guidance **does not** provide details on how to learn to code or how your code should be Quality Assured but we have included some useful links throughout and provided further reading.

The contents of this document have mainly been compiled from case studies of rewriting Excel models to R within the Department for Education, although the principles discussed still apply to other software languages and Departments.

This guidance contains the following sections:

1. Introduction

2. Considerations to re-platforming
3. Practical guidance on how to implement a re-platform
4. Case studies
5. Stakeholder FAQs
6. Further reading

Chapter 2

Considerations to re-platforming

In addition to the many advantages in porting models into open source software, there are also a range of barriers. In this section we discuss the benefits, barriers and risks arising from this process. We also highlight instances where porting a model may not be the optimal solution and discuss obtaining buy-in from stakeholders.

2.1 Benefits of re-platforming

In short, there are two overarching benefits to re-platforming:

1: Reproducible Analytical Pipeline (RAP)

2: Speed

A Reproducible Analytical Pipeline is hugely beneficial and brings numerous benefits, including:

- accuracy
- useability
- readability
- transparency
- automated version control
- reduced duplication
- easier collaboration
- improved data management
- ease of updating documentation
- easy to follow audit trail

Although a significant amount of time is spent quality assuring spreadsheets, human errors are common. Evidence[1] suggests that whilst errors are rare on a per-cell basis, large programmes are likely to contain incorrect figures, these are extremely difficult to detect and correct and there is a tendency to be overconfident in the accuracy of spreadsheets. Re-platforming reduces the risk of errors, supports and enables easier version control processes (for example through "Git" or other tools) and collaboration. Clear, well structured and commented code is succinct, easy to understand and update therefore making it quicker and easier to update and quality assure. When done properly, re-platforming means that quality assurance processes are embedded **throughout** the pipeline as opposed to a time-consuming and manual task at the end of a project or model completion.

Lots of the benfits of RAP lead to the second overarching benefit of re-platforming: **speed**. A code-based approach can quickly and repeatedly access relevant data, perform relevant calculations and produce a variety of outputs and documentation **and** perform built-in checks. Automating these processes frees up time for people to focus on more challenging and interesting tasks that are likely to add more value in the long run.

A final potential benefit of re-platforming is that it can be used to develop skills for individuals and within teams. Re-platforming an existing model that's already been scoped, designed, tested and has clear outputs is an ideal way for someone to develop coding skills whilst adding value to the team.

For stakeholders, there are several benefits, some of which are outlined below and covered in more detail here:

- Reduced run times and quicker updates result in more time for analysts to focus on more complex problems.
- Risks are reduced through robust QA
- Outputs can be produced rapidly

- When done well, there is the potential for greater scenario testing across assumptions due to increased versatility of the model.

2.2 Barriers to re-platforming

The level and mixture of skills within a team/unit/division can be a barrier, with models often built in a 'common denominator' software in order to ensure everyone within a team is able to use and update it.

It may be particularly challenging to upskill individuals in relevant languages and/or re-platform a model alongside Business As Usual (BAU) work. Carving

[1] *Panko*, 2016

out enough time to re-platform and/or to learn new languages and/or techniques is difficult and often the biggest challenge. But as discussed shortly, re-platforming will often reap time-saving rewards in the longer term.

There may also be barriers to re-platforming when a model needs to be shared and/or used by non-technical colleagues. This may prevent re-platforming but there may be ways in which the model could be re-platformed with a user interface that would facilitate sharing and/or use by someone without the relevant technical skills.

For very simple models, there may be no clear need or benefit to re-platforming and this should be considered before time is invested in re-platforming unnecessarily. However, for more complex models, complexity may also be a barrier to re-platforming. All of the more complex models listed below are likely to take more time to build and may require further upskilling in order to incorporate requirements, but there are likely to be many benefits to re-platforming complex models such as these.

- A stock/flow model with a small number of stocks and flows
- A linear regression model
- A complex forecasting model involving multiple steps
- A stochastic individual level model with data preparation, status prediction via logistic regression, then using the status to predict a linear range

2.3 Risks of re-platforming

There are several risks associated with re-platforming. We'll cover each of these in a little more detail below:

- **What happens if you make a model and it's not made well?** You may have invested a significant amount of time and resource and produced something that is no better or saves little/no time compared to the existing model. This is a possibility but with clear mitigations in place this can be avoided. Carefully managing the rebuild to ensure that you have a clear scope and specification, structuring the re-platformed model well and ensuring you have the right skill sets in place should help to prevent this.

- **Does a team then constantly get in a cycle of remaking the model each year?** This is a possibility, but this is already a common issue with Excel-based models. However, with a code-based approached, existing code that has already been reviewed will need minimal scrutiny and only changes and/or new code will need full QA so this may not be as labour intensive as a non code-based model rebuild.

- **What if all the personnel with technical skills leaves the team?** Ensuring that you have good documentation in place as part of the re-platforming process should go a long way to mitigate against this risk. In practice that means that code will need to be well structured and well commented and that technical notes and user guides are completed as part of the re-platforming process, as a minimum.

It is worth nothing that as with the previous risk, this could also apply to a non code-based model but the ease with which documentation and guides can be produced for code-based models is such that this shouldn't be difficult to overcome.

- **Does it change the people you employ?** Re-platforming doesn't necessarily mean that you need to employ different people. Based on experiences at DfE to date, there is a strong appetite for analysts to develop and expand their programming skills and an equally good culture of knowledge sharing which underpins and supports that development for individuals. Whilst we have numerous Excel-based and other non code-based models, working on code-based models is an enticing propostion for lots of analysts therefore re-platforming could make it easier to recruit in future as a code-based approach may appeal to more candidates.
- **Will a re-platformed model be a black box that is difficult to explain to stakeholders?** Providing that the model is well commented and well documented, the risk of it being a black box to other analysts is low and as discussed above this risk can be mitigated by developing appropriate user guides and technical notes. It *may* be more difficult to walk stakeholders through the model initially, particularly if they are used to being able to scrutinise an Excel workbook. However, by presenting and sharing intermediate information though, for example, R Markdown outputs, at different stages of the model to demonstrate key outputs this can be overcome.

2.4 Should you re-platform?

Before embarking on re-platforming, now that you know about the risks and potential barriers, it is important to consider whether re-platforming is really the right choice for your model.

Some important questions to ask yourself before moving ahead include:

- How complex is the model?
- How frequently is the data refreshed?
- How frequently do the data inputs change (formats)?

- How many different data inputs are there?
- How frequently do the required outputs change?
- Who needs to use the model?

If a model is very simple and quick to update once a year with a single, very consistent data input and a fixed output, it may not be sensible to spend time re-platforming. However, even a simple model that requires frequent data refreshes could be considered for re-platforming. Continuing along that spectrum, a more complex model with multiple inputs would definitely be a strong candidate for re-platforming.

Ultimately, it's about weighing up whether the investment in the re-platforming process is worthwhile. In a lot of instances this will be the case, even for simple models where the main benefit of re-platforming would be to improve QA. However, for some of the simplest models, or models which need to be used by a non-technical person, it might be necessary to keep the model in Excel.

Chapter 3

Practical guidance on how to implement a re-platform

Once you have determined that your model should be ported into a code-based format there are a number of things you should plan and consider as you start to scope your project. We'll provide some general guidance here with further practical tips in the case studies section.

3.1 Pre-requisites

To ensure a successful outcome it's essential that several areas are addressed before re-platforming begins. Perhaps most importantly, you will need **buy-in from stakeholders**. This section includes some of the most commonly asked questions from Stakeholders and as this could be one of the most difficult challenges, engaging with them early is likely to be beneficial.

Ahead of carrying out any re-platforming, you should identify who your **end users** are. Are they analysts, managers, policy stakeholders or a combination of those?The answer to this question might influence the language you decide to use and the type of output(s) you decide to create. Thinking about this before you start your re-platforming will help you to shape the redevelopment in the best way. It should be one of the first steps of your planning as the answer will impact your requirements, which will in turn feed into your resource and skill requirements.

Once you have secured Stakeholder buy-in and identified your end-users, ensuring there is sufficient time and appropriate resource dedicated to the project is vital. As with any piece of analysis, a rushed job is more likely to be a poor job. This could mean postponing Business As Usual work, existing model

updates and/or securing additional resource. But, however tempting it might be to 'contract out' the re-platforming to experts who then hand 'it' over, this is *not* recommended as without *a lot* of collaboration, you're likely to end up with a team who do not understand the model. Therefore, the analysts who will carry out the re-platform need to have sufficient software knowledge for the chosen platform. This might mean they need to undertake L&D activities are needed and/or engage with experts and relevant groups for advice and support (for example, the DfE Analyst Network Data Science channel or Python Team), particularly in the event that any difficulties are encountered.

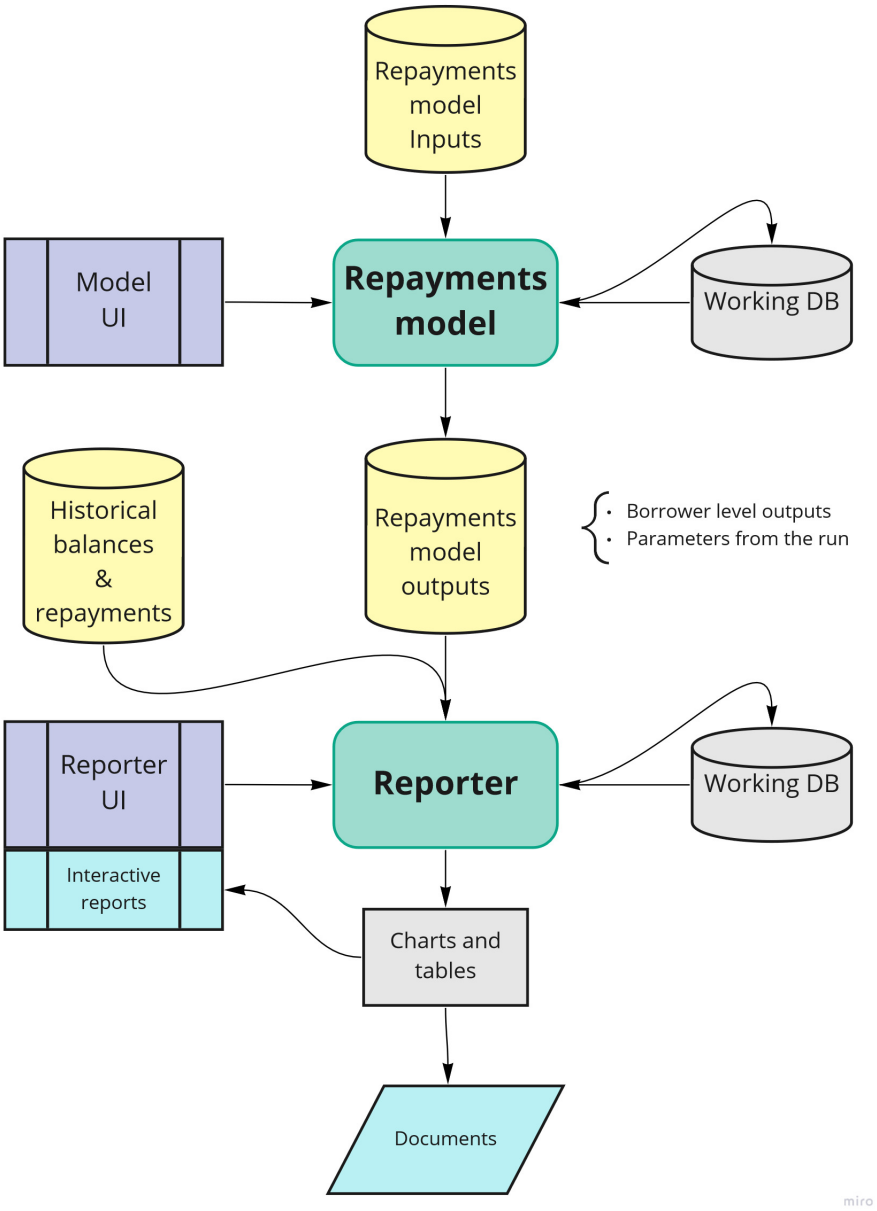
3.2 Design choices

Choosing a **language** to re-platform into should be one of your earliest considerations after deciding to re-platform. You will need to choose a language that allow you to deliver the required outputs but also one with which your team have the skills to code in, or the ability to learn. Within DfE, SQL is already well established, whilst R is becoming increasing popular and is very well supported with Analytics Academy and a number of willing experts across the Department who regularly provide support and advice through the DfE Analyst Network. Python may also be a consideration and whilst it is not (yet) as popular as R within DfE there is an active community and plenty of external resources online.

As with lots of code-based analysis, it is likely that the code **design** will evolve as the re-platform commences. However, having a clear set of objectives and required outputs from the outset is critical to ensuring that the re-platformed model is fit for purpose. This will also help to avoid scope creep and reduce the likelihood of the re-platformed model becoming unnecessarily complex.

For **clarity** you will need to think carefully about how the re-platformed model will be structured so that it's efficient and follows best practice. In fact, you may specifically want to avoid replicating the existing processes and mechanics of the non-code based model in order to improve the efficiency of the re-platformed model.

We've already covered the benefits of writing clear, well structured code elsewhere and a re-platforming project is no different to any other code based project in that respect. However, depending on the complexity of the model, in addition to ensuring that code is well structured, consistent and commented appropriately, it may also be beneficial to provide a schematic diagram of how the model works, such as the one below.



As we'll cover shortly, well documented re-usable components will help to ensure that the code is clear and easy to follow. This should ensure it's easy for others to pick up the model and update or develop it more readily in future as well as making the model as transparent as possible should you wish to go into more detail with stakeholders.

There is a strong likelihood that if you're undertaking a re-platforming project that the existing model may be repeating the same formulae or data processing numerous times. Whilst you could write some code that mirrors those processes, writing (and clearly documenting) bespoke functions will help to achieve reduced run-times and the flexibility to run a greater range of scenarios and/or uncertainty testing.

Whether the existing model takes account of uncertainty or not, we would also strongly advise you to think about building-in uncertainty (where appropriate) when you re-platform a model. There are various techniques to consider and some of these are covered in more detail as part of the Uncertainty Toolkit for Analysts in Government.

3.3 Testing and Version Control

There is more information about how to conduct **testing** in the DfE Good Code Practice but in addition to unit testing, we would recommend ensuring that the re-platformed model outputs match the previous model before you start developing the model further. Ensuring that the re-platformed model can produce equivalent or comparable outputs to the previous model will give stakeholders confidence in the re-platformed model as you will be able to demonstrate that the re-platformed model can reproduce the existing model.

As we've covered elsewhere, **version control** is a really important aspect of any coding project but it particularly adds value when teams are working collaboratively. Using Git is far superior to keeping local scripts and manually amending file names and there is unlikely to be a better time to embed Git into your ways of working than when you move away from Excel to re-platform a model. In DfE, repos within Azure DevOps are the existing code version control tool of choice and there are several resources, including how to access and use DevOps, listed in the Further Reading section.

Whatever the Version Control tool used, it is advisable to create a repository for the project and for all collaborators to use the repo to manage their code. An ideal workflow is where the analyst creates a *pull request* (AKA *merge request* in GitLab) to merge their code changes into a clean master version of the code and the approver (quality assurer or designated collaborators) then evaluates whether the change is appropriate and either approves or asks for edits to be made.

Within Azure DevOps repos, you can set branch policies such as a setting minimum number of reviewers and/or automatically including code reviews.

Once the re-platformed model is in use, new branches should be created to develop features or changes and then these should be merged back into the master branch with a pull (merge) request to seek approval from the appropriate person.

3.4 Documentation and Outputs

As with any model, good **documentation** is a key part of knowledge management; particularly to ensure that other colleagues can understand, use and update the model and so that any data and/or assumptions are recorded clearly. **QA reports** can also be automatically generated so these should be a key consideration when thinking about outputs too, for example producing basic summary statistics, a list of checks or unit tests that have been carried out or showing totals through various steps of a process. There are various packages are available to produce documentation depending on the platform used and type of outputs needed.

In R, Markdown can be used to create html **outputs** with dynamic visuals and code snippets and this can be taken one step further and published via Bookdown. This is often done to produce reports or guidance once the code has been completed as separate .Rmd scripts will be needed which means that if you make changes to R scripts you may also need to duplicate these in .Rmd files for example.

ROxygen overcomes the above issue as you can comment scripts as you go along and produce outputs from the scripts. It readily supports documentation for R packages so this may be useful if you are creating bespoke functions and/or packages. If you need to present a lot of code and less text/explanation, you may also want to consider using the Spin function within the Knitr package or rendering directly from a script using the "compile notebook" button in R Studio.

When considering outputs, think about what your users need. For example, an interactive dashboard output or RShiny App, such as this one, would work well for scenario testing, whereas figures needed for a Business Case could be output to an Rmarkdown document or even a simple table or .csv file. Whatever the final outputs are, you should also think about interim steps within your script as it's likely your users would previously have worked through a spreadsheet to look at parts of the calculation(s). A comprehensive Rmarkdown with different sections replicating the Excel checkpoints would be a good way to demonstrate what your script is doing and give Stakeholders confidence in the model.

Chapter 4

Case Studies

The case studies included in this guide were identified through a cross-Department working group. This covered analysts who were new to code-based models who had to learn R and ask experts to help up to those who were already experts in this area so we hope that the case studies showcased will provide useful learning points for everybody, no matter where you are in your learning journey. We hope to include additional case studies in future so please get in touch if you would like to add yours!

4.1 Schools Block National Funding Formula Model (SB NFF)

The SB NFF model carries out funding allocations for the schools block element of the Dedicated Schools Grant. Individual schools and pupil characteristics are used to underpin the formula allocation for more than 20,000 schools which meant that running the model in Excel involved several linked workbooks and data processing and manipulation in other files. Whilst there was a really good analytical argument to be made for re-platforming this model due to the technical challenges, one of the *biggest barriers* was that policy colleagues were initially reluctant for a re-platform as they wanted to be able to interact with the model themselves.

4.1.1 Model details

The schools block NFF Excel spreadsheet model often took most of a day for two analysts to update new changes for scenario costing. It was also very prone to error given the number of manual interventions required therefore the aim of the re-platform was to create a cleaner pipeline for scenario costing.

4.1.2 Replatform details

The first phase of the re-platform took a few months of a single analysts time to create the initial model, alongside Business As Usual work. At this stage it was a "third-build" model and the analyst was able to match outputs from the existing model to fully QA'ed models to ensure the R model was working correctly.

The second phase of the re-platform involved other members of the team implementing further QA and error checking into the model. An important change that was needed at this stage was a restructure of the code to allow for an Rmarkdown document to be created for policy colleagues. This was created iteratively, with a lot of engagement from working level policy colleagues in order to capture the right level of detail in an easy to understand way before taking more senior stakeholders and the model SRO through the walkthrough for sign-off. After this, the R model became their main model for the following allocation cycle.

4.1.3 Benefits

There were immediate benefits from the first phase - updating the model parameters became easier on the new platform than in Excel. Other analysts in the team who took on the rest of the NFF re-development were totally new to R but had experience of other languages. They were able to use what had been done in the first phase to learn the language in a hands-on way by developing the next phase.

The model is now much more efficient, running in seconds rather than tens of minutes. The new model can query directly to SQL compared to the previous Excel model, which required generating/exporting code, then loading the data in. This was a really big improvement and *greatly enhances QA* as the SQL query is directly within the model therefore it's transparent *and* by querying the data directly it removes risk of both copying and pasting errors or using the wrong file.

The team is now much better equipped and more efficient in their support to policy colleagues with development of the formula. The re-platformed model meant the team were able to provide better support to last year's SR because it was much quicker and easier to run scenarios and using the skills they'd developed, analysts produced an output dashboard that included a number of different comparisons and breakdowns, which helped the policy team understand and communicate the impact of a particular funding scenario.

The team use repos within Azure DevOps and git for tracking changes and version control. This has improved governance of the model, commissioning QA task processes and their ability to work on a single model collaboratively. The NFF team have also used their new R skills to produce other dashboards

and tools using R and RShiny to support the policy team with correspondence, PMQs. etc

There have also been wider benefits too. The knowledge developed from the model re-platform has been used in subsequent model development across the division as well.

4.1.4 Practical tips from the team

- Training - On moving to R, team members had different levels of expertise, learning trajectories and amounts of time available to dedicate to on-the-job learning. There wasn't a structured plan for what needed to be learnt, and training was very self-directed, which was manageable but tricky. Having an agreement in place for what fundamental learning needs to take place and allowing time for that training to be undertaken would be helpful for any re-platforming project.
- Governance - There were concerns from the policy team about whether they'd be able to understand the model and whether they'd ever have enough confidence in it to sign it off. The SRO would normally be walked through the Excel model step by step so the team had to think carefully about how to retain similar confidence in the R model. They achieved this by highlighting the benefits of re-platforming and by providing walk-throughs of key chunks of the code. But, it did take some effort from both sides – analysts needed to make sure code was clearly written and reasonably accessible, and the policy team needed to put in some effort to understand and follow a code-based model walkthrough. Thinking about how the re-platformed model would impact governance processes (e.g. dual builds, policy walkthroughs) and how to manage those changes was really important.
- Resilience - Whilst there is a move towards using R more widely in DfE, it's still something many people are learning and levels of expertise vary. This needs to be kept in mind when developments are made to the model and as the team upskills further. The team need to consider whether they can expect someone else coming into the team to be able to pick up the modelling and how much training they will need. Potentially this has its advantages as there is a good amount of technical and non-technical development to be had from working in their team, but it's something they need to bear in mind when making any substantive developments.

4.2 Student Loan Earnings Model

The Student Loan Earnings Model is a micro-simulation model that forecasts future annual graduate earnings at an individual level, which are used to deter-

mine future student loan repayments that the Department expects to receive on its expenditure on student loans.

The model is audited externally by the National Audit Office, who maintain a shadow-build version of the model. Whilst this is needed for NAO's own purposes, it adds value to the DfE process as well by acting as a dual build.

It was re-platformed from Microsoft Excel to R over several phases, with the re-platform bringing out nuances in both the model methodology and implementation details. The process was carried out collaboratively by a small team, using version control software to track and merge changes.

4.2.1 Model details

The Earnings Model forecasts future annual earnings using a collection of submodels which:

- For a given borrower's characteristics and history, predict next year's employment status (from 4 possible states),
- If employed, forecast earnings using the suitable regression model.

The Excel-based model implementation had several issues which made it complex to update and slow to run:

- It required manual extraction of data stored externally in SQL databases - the Excel spreadsheet contained sample SQL code to extract the datasets, before the data was copied to an additional tab in the workbook to be used.
- Key steps within the model's implementation logic were hidden within complex Excel formulae, and were time-consuming to trace and debug.
- The Excel model was only able to run a small sample (200,000) of the almost 5 million borrowers in the system, so it lacked capacity to model the entire population.
- An additional issue was that platform-specific issues in Excel (such as how random numbers are generated and stored) caused differences in reproducing simulation results on other platforms.

The aim of the re-platform was to improve readability, reliability and speed of implementation, to permit larger model runs, and set up a robust version control methodology for future model improvement.

4.2.2 Replatform details

The re-platform took place over phases by a small team of analysts - all quite new to the model but proficient in R and other coding languages.

They started by copying the workflow from the existing implementation[1] - following the logic of calculation steps displayed within the cell formulae of the Excel spreadsheet, and incrementally ported each calculation step of the methodology across.

As part of the re-platform, lookup tables from Excel were exported as datasets but external SQL databases were queried directly from the re-platformed model. This brought several benefits including ensuring changes to the code were only needed in one place, reducing the risk of copying and pasting errors by accessing the data directly and allowing the entire dataset to be used within the model, instead of only a small sample.

The team used version control within an Azure DevOps repo to carry out continuous code review during development. This meant that quality assurance occurred incrementally - potentially eliminating the need for a detailed 'full' quality assurance after the re-platform was completed. However, wider unit testing may have been beneficial to provide assurance of the end to end process.

One reocurring issue was whether the model methodology should be improved at the point of the re-platform, or whether it should be a like-for-like implementation, with future scope for model methodology improvement. It was decided to keep a like-for-like implementation to ensure that the replatformed model behaved correctly, although the tradeoff was a slower timeline for model improvement.

The re-platform was carried out prior to writing documentation. It was unclear to the team whether it was more advantageous to document along the way given the analyst time available. In hindsight this may have been a better option as it would mean analysts documenting each section they had replatformed.

4.2.3 Benefits

One of the biggest benefits from this re-platform is that the model can utilise the full dataset and model outcomes for individual borrowers based on their full characteristics, rather than scaling results from a modelled subsample of aggregated borrowers.

In addition to this, the replatform has led to other benefits:

- The model methodology is easier to understand, with model parameters and subprocesses defined in a more readable way. This makes it easier to tweak the model for model runs, and simpler for new team members familiarising themselves with the model.
- Model run times are quicker, and it is easier for a single team member to execute multiple model runs.

[1] This may have been done differently if re-platforming based directly on mathematical formulae rather than from an implementation.

- A more robust version-control system is in place which means that model changes are easier to quality assure, model versions can be tracked and compared, and the model development process is more transparent for external audit.

4.2.4 Technical tips

The team found that switching to a code-based implementation led to additional questions during the re-platform, which may recur in other projects. Some of these are listed below:

- Use appropriately named functions and variable names to simplify the readability
- Regular checkins if multiple analysts working on code (code style/naming conventions)
- Camel case or snake case for functions, starting with a verb (e.g. calcNew-Balance)
- Think about variable names - switching between SQL and R - upper case vs camel case/Pascal case
- Agree and decide on the optimum Git branching strategy - e.g. using merge vs rebase, fast forward history
- Don't store large binary files in Git if you can help it!
- Consider using lists of lists to store the output data (memory issues)
- Consider using data.table() package over dplyr for speed with large datasets

Chapter 5

Stakeholder FAQs

This section covers the key questions Stakeholders may have over the course of a re-platforming project. It is written primarily for Stakeholders but analysts may also find it useful to think about the questions their Stakeholders may have and how they can work together to develop confidence and secure buy-in to the re-platformed model.

What *is* re-platforming?

Re-platforming is the process of converting a model from a piece of software to a code-based platform. In practice, that usually means turning an Excel model into one which runs using code. Within DfE, the most commonly used coding languages are R, SQL and Python and a re-platform may involve a combination of the two, particularly R and SQL together.

Do I need to be able to understand code to understand a re-platformed model?

Absolutely not. Although the calculations will be run using code, there are various user-friendly output formats that can be generated which will allow analysts to walk you through what the model is doing at various stages and what the outputs of a model are at those stages too.

How will I know that the model is doing what it should do?

There are a few things you can do to check this, including:

- asking for a **dual-build**. This should demonstrate that the re-platformed model matches and can produce equivalent outputs to the previous model. However, it's important to be aware that there may be times when the re-platformed model may not be an exact match. For example, if either the model needs to be updated and/or the re-platform needs to be built differently or incorporate changes. In such instances, you can ask analysts

to reconcile and explain any differences to ensure that the two models are aligned.
- asking for breakdowns of outputs at key steps within the model to ensure that interim figures are in line with expectations
- asking lots of the same questions you would usually ask around assumptions
- sense checking outputs, comparing outputs to other data and your own expectations

What do we mean by automated outputs?

What we mean by an 'automated output' is any output that is created without any additional manual intervention when the script is run such as text-based reports, a slide-deck or a html file. In short, if there is something that you need to see on a regular basis, code can be written which produces this every time the script is run or on demand.

What things can I look at myself?

In the same way that an Excel model might have an output sheet with key figures, tables and/or charts, these outputs can produced by a script-based model too (as set out above). Similarly, if there are intermediate steps in an Excel model that you use as a sense-check of for other purposes, these can be generated either as individual outputs or as part of a package of outputs. There are several added benefits of producing these outputs directly from the re-platformed model including:

- more interactive outputs than Excel, for example you may be able to hover over charts and tables for further information or use toggles and drop downs to visualise different options without having the model itself
- reduced risk of copy and paste or data source errors as outputs produced directly from the model
- autochecks can be included which highlight or flag issues, for example if a figure is outside an agreed range or tolerance
- inclusion of key parameters, model information for easy audit trails

What are the main benefits for me as a Stakeholder?

As we cover elsewhere within this guide, there are numerous benefits of re-platforming but as a Stakeholder it is *likely* to mean that **robust QA** is in place and can be demonstrated to you quickly and easily. That in turn means that analysts can spend more time on complex problems and other analysis, instead of on routine checks and it *should* mean that you can still be as, if not more, confident that the model is as accurate and error-free as possible.

Re-platforming *should* lead to much **quicker** run-times so you're likely to get answers to your questions sooner. This is true for data refreshes as well, and

since QA should already be built in, this means that *when done well*, updating the model and producing outputs can happen at the click of a button!

Automated reports and **outputs** can also be set up which means you can expect to see consistent reporting, and where relevant, interact with the outputs through dashboards or Apps. In addition to all of the above benefits, there is also the potential to include more sophisticated analysis and incorporate **uncertainty** much more easily within a code based model. If this has been built in, it means you may be able to ask more challenging questions and develop your evidence base even further.

What are the risks?

As a Stakeholder, the biggest risk is likely to be that the re-platform takes longer than expected. That could be due to unforeseen challenges when writing the script, scope creep or both. This could occur as part of any new piece of analysis and therefore the advice and mitigations are the same: agree a clear specification at the outset and have regular discussions on progress as this should help you to work together with analysts to deliver the re-platformed model to schedule.

You might be tempted to bring in additional resource to deliver the re-platform and leave existing analysts able to fulfill their current roles. We would advise against this as there is a risk that the new model will not be as well understood by the team who will own it which means you won't reap the key benefit of re-platforming.

The next biggest risk is that the re-platform might not deliver all of the benefits outlined here, particularly if there are delays which mean the scope has to be revisited in order to deliver a minimum viable product on schedule. Over time, as with other types of models, additional features and functionality can be added and developed which should enhance the model and realise more of the benefits of a script-based model.

How long will the re-platforming take?

This will vary from model to model and is dependent on a number of factors, such as the level of experience and available resource. However, if you're able to hold off commissioning analysis or can postpone BAU work it's likely to be delivered sooner as analysts can focus more of their time on the re-platform.

Chapter 6

Further Reading

6.1 Internal resources

- DfE Good Code Practice
- DevOps for Analysis
- Analytics Academy

6.2 External resources

- Reproducible Analytical Pipeline (RAP)
- RAP MOOC
- Setting branch policies in DevOps
- ROxygen
- Spin function within the Knitr package