Rigour for Agent-Based Modellers

– an accessible guide for the perplexed (*as well as for the over-confident who might* **not** *be perplexed but should be*)

Bruce Edmonds[1][0000-0002-3903-2507] and others (TBA)

[1] Centre for Policy Modelling, Manchester Metropolitan University, Manchester, UK
bruce@edmonds.name

Abstract. A guide to doing agent-based modelling in an appropriately rigorous way is presented. This tries to be (a) accessible to non-experts and (b) appropriate to the stage and purpose of the modelling. Thus, what is suggested is aimed at four different levels: (1) for one's own understanding, (2) when presenting a model to an audience for discussion, (3) when publishing in a journal article and, (4) where the modelling may influence decisions that affect people's lives. Thus, the highest level of rigour is not demanded at earlier stages. As one moves to the next level, the suggested steps needed for rigour increases, making it possible to incrementally learn increasing rigour as one gets more serious. Level (4) will not be discussed here but left to future versions of this document.

Keywords: rigour, agent-based modelling, modelling purpose, introduction.

Rigour is a set of norms and procedures a field has for ensuring their conclusions are reliable. People outside the field do not want to know the details of how we achieve such reliability, just that they can rely on the results and understand for what they can be relied on for. People inside the field, only communicating with each other, are often a bit more relaxed but they still want models that '*do what it says on the tin*'. One can always be more rigorous, so this is not an all-or-nothing decision but a continuous journey "*onwards and upwards*". All modellers are somewhere on this journey – none of us is perfect and the standards continue to evolve. All of these have been discussed elsewhere but not collected and presented in an organized and accessible manner.

The guide is presented as a series of steps to implement at each level – some extra things to do alongside your growing modelling abilities. We have tried to make these appropriate to each stage – not demanding a gold-standard from begginers. Concentrate on the steps at the level appropriate to the task you are engaged in. If you feel you have got on top of the steps at one level, then move on to those at the next level. This should not be a tick-box exercise – racing through all the steps in a superficial manner will not make you more rigorous – rather the best reliability comes from carefully developed thinking and habits. There are four "levels" for different modelling circumstances: **Level 1** – *for your own understanding*, **Level 2** – *for presenting a model to an audience for discussion,* **Level 3** – *when publishing in a journal article, and* **Level 4** – *where modelling may influence decisions that affect people's lives*. These are cumulative, so

each level builds upon (and includes) the previous levels. It is important to realise that the level of rigour is not related to the level of your skill at programming, but rather what is necessary to the kind of task you are engaged upon.

We understand that declaring standards for rigour is a ("small-p") political act. Promoting norms that mean modellers have extra obligations in their already busy lives will not be universally popular. We are trying to make only modest suggestions here, but adopting them will mean that modelling is a slightly slower and more cautious process. It will involve more of the social processes of science: checking and critique by others, and more care about how the results might be interpreted by non-modellers etc. In other words any work (beyond just playing with a model [17]) will be slightly less fun. We admit that we, as modellers, have not always done all the steps that we describe here, but hope by declaring these standards we can also motivate ourselves to improve. What is in here is a snapshot of some of the practices and issues you might want to consider – we do not think it lists *all* those possible, that would make this into a book. If you are serious do the follow-up reading suggested in each section.

1 Modelling for your own understanding (*level 1*)

This section is aimed at those who are new to Agent-Based Modelling (ABM) to guide them on the early stages of this journey. This level is aimed at those using ABM in a more rigorous manner so as to reliably increase your own understanding. Before taking up anyone else's time (other than teacher, fellow student, advisor etc.) in understanding the model or its results, one should also implement level 2.

General Advice. The first barrier when coming to ABM is getting to grips with the technology: how you write code, run it, find errors, see the results etc. During this initial phase one is simply glad to get anything to work and one's whole energy spent understanding the mechanisms and facilities of the system you are programming with. During this phase one is continually going back and forth from manual/tutorial and one's developing model – reading a bit more, trying things out and then revisiting what one coded. At this stage it is good to look at and play with other people's code – trying to understand what they have done (and why), then messing with it to see what happens when you run it. Ideally, one should have someone to ask when one has beat one's head to a pulp trying to get something to work but making no progress.

It is only worth worrying about rigour when one has started to get past this initial phase, so one can concentrate a little on *what* one is trying to achieve. The key to this stage is not just doing stuff, but reflecting on what one is doing and thinking how one might do it better. Sure, it can be helpful to copy what another modeller is doing, but then try to also work out *why* they have coded in the way they have.

Think about what you are trying to achieve with your model. One does not do ABM *just* for fun – usually one is trying to gain some more understanding of something. The thing you are trying to understand can be many different things: an observed system, the theoretical consequences of some mechanism, some ideas, someone else's model, etc. In this case the point is to *learn* something, something that one would not learn *without* the model (Even if in retrospect what you learnt is obvious – it may only

be ‘obvious’ only after having done the modelling). Thus it is worth keeping in mind what one is aiming to achieve at each stage of modelling (even if this changes over time). The more precise this is the better – vague aims are generally less useful than more explicit aims. One of the troubles with ABM is that it is a very flexible tool that can be used for lots of *different* purposes [10, 9], which can cause confusion. Limit yourself to *one* aim at a time (more risks achieving none well).

Keep track of your model code as it develops. Computer simulations are often simpler to understand than many real world systems. However ABMs can quickly become sufficiently complex that the programmer does not completely understand them. In other words, it is easy to fool yourself using your own code. Part of the problem is that the human brain can only hold so much in mind at any one time (this is the point of an ABM). Thus, although it may seem obvious to you at the time, a week or two later you will have forgotten the reason you implemented bits of your code in the way you did. For this reason it is good to document your code as you go along. In other words, to give modules, procedures, functions and variables long meaningful names and to write comments into the code concerning what each bit does and why. Nobody likes to go back and do this later, and it is quicker to do it whilst it is fresh in your mind. These comments will also be useful in Level 1 when you want to present your work. Such documentation has more than an administrative function – describing what you are doing and why aids reflection on your actions and thus encourages precision. It is something about imagining someone else reading your code and its comments that aids a more objective perspective on what you are doing.

Find out what is happening in your model by messing with it. Once you have a model that seems to be doing something interesting, the next step is to understand what is happening in the model – find out why, how and when the interesting results occur when you run the model. Do not underestimate how difficult this can be! Even when we think we know what is happening this understanding is often incomplete or mistaken – just because we intended our model to work in a particular way does not mean that it does (this is not because you are not clever enough, there are basic limits to *any* understanding [24]). Thus one has to take *active* steps to ensure one has an adequate understanding of your model – just running it a few times, watching it and getting out a few numbers or a graph, is insufficient. You should do two things: (1) program as many different measurements, traces, graphs, indicators etc. of what is happening in your model as possible, a good visualization really helps understanding, (2) do lots of different experiments with it: changing parameter values (including setting them to silly values), change rules in minor ways, run it about 20 times with the same parameters to get an idea of the variation in output etc. etc. Use your imagination.

Further reading and resources. A good general introduction to modelling, aimed at the social scientist is (Gilbert & Troitzsch 2005). This sets the context and motivation for simulating social phenomena, as well as giving useful modelling examples.

If you are just starting out to model, then the NetLogo modelling language is very accessible, with a clean, easy-to-read syntax and good quality documentation. Many of the entities, tasks and tools you might use for modelling are already built in to it. This is a fully powerful programming language – it does lack a couple of features advanced

practitioners might want (step-by-step debugging facilities and user-defined hierarchical classes of object) but is otherwise complete. NetLogo comes as a complete package, with a good library of example models and many extensions. Furthermore the NetLogo website has many useful pointers and other resources, including pointers to user groups where you might find further help.

The CoMSeS.net website is a wide-ranging repository of resources for serious agent-based modellers. It has discussion forums, tutorials and a large library of simulation models with their code and documentation. It covers all systems for modelling.

For an idea of some of the range of different purposes for modelling, read Epstein [10] who lists 16 reasons, other than prediction, why one might build a model.

2 For those who want to present their model to an audience for discussion (e.g. workshop, discussion paper, etc.) (*level 2*)

So you have a model that seems to show interesting results. You are all enthusiastic and want to (or have to) present it to others. However, it is easy to waste other people's time in this way if you have been sloppy, made a mistake in your code or just not sufficiently understood your own model to answer questions about it. Here we outline some modest steps that will give you more confidence in your understanding, and which will make presenting your model more productive. Workshops are for discussion and comments upon *emerging* work, what is called 'work in progress' – if you are intending to publish mature work in a journal or top-level conference you should also do level 3.

General advice. Here we recommend that you adopt a much less friendly attitude to your model. That is, it is helpful to think of your model as a con-artist that is trying to deceive you – you are trying to probe it sufficiently that any such deceit is revealed (and if necessary fixed). One reason why this is necessary is that modelling affects one's perceptions – one starts to see the world 'through' one's ideas about the model – one selects what one notices by what is consistent with that model so you may well not notice things (e.g. aspects of the model behaviour) that is not part of that story [15]. However, even without this, computer programs can be *fundamentally* complex [24] so that things might be happening in your model that you have not appreciated.

One approach to this is using a kind 'threat analysis'. In this approach you first identify limits/possible weaknesses in your own modelling, then plan what you are going to do to mitigate/deal with these, and finally assess the success of those measures and remaining limits [25]. Of course, the weaknesses are relative to what you are trying to achieve with your model, so before anything you need to be clear on your modelling goal. This approach is also helps being honest about what your model achieves.

Document your modelling clearly. The first step in documenting your model goes way back before you start coding – starting with your purpose in doing the modelling. This is the declared goal that you are suggesting that your model be judged against – sometimes this is framed as having a clear "research question" you are trying to answer. It is common to have several different destinations in mind for one's model, but for any one presentation/report/paper you should focus on just one goal. Then you need at least some description of how you plan to achieve this purpose and a high-level description

of the conceptual model (that is the intended ideas, structures, processes etc. that the model is supposed to implement). It is helpful to briefly document the sources that informed the design of the model (assumptions, theories, etc.).

Ideally, someone else will want to dive into the details of your work and maybe even do an independent reproduction of your model, but this requires a lot of time by the person doing this, so it is not common at this stage. On the whole, others will only bother about how your model is made when they have seen interesting results or radically new ideas – the proof of the pudding is in the eating, not the recipe. To facilitate others to get into the details of your modelling, make your code easily available to others and provide good documentation.

Actively prevent bugs in your code. Bugs can be very subtle so they do not necessarily come to light during the exploration of model results. Rather, one needs to take some *active* steps to check that your code corresponds to your conceptual model..

Key to preventing and then identifying any bugs is structuring your code. The most important aspect of structuring is dividing up your code into independent (or semi-independent) chunks that make sense (represent a particular process, entity, calculation etc.). Having a clear conceptual model is really helpful for this – making it all up as you go along might result in "spaghetti" code that is very hard to de-bug or understand later. The main strategy for finding existing bugs is then to test each of these chunks separately. You can "turn off" other parts of the code by adding temporary statements into other parts to short-circuit their operation or to make them return a default result (e.g. a constant or a random value) and then, once you think each part works as intended you gradually "turn them on" again until you have the whole simulation working again. Also try turning them on in a different order.

In computer science there are a whole lot of other strategies for planning/checking code and eliminating bugs, some of which we will mention in higher levels.

Do systematic experiments. If you are going to present results then this cannot be just cherry-picked examples. It is best to divide your results into two parts: in the first aim to give a more general idea about how the model behaves under a variety of conditions and the second focusing on the results you think are interesting. Thus systematically explore the impact of the important parameters and look at the individual runs that result as well as the average of a set of independent runs (to get an idea of any central trends but also the variation between runs). Check that changing parameters or processes that you do not think are important to the results does not change them significantly (including the random number seed). Keep a record of the experiments you run, recording: the model version, the parameter values and the data that resulted from them and keep this somewhere safe (you may need to refer to them or recreate them later). The most common question at such forums is "but what if you did X?" – so it is good to be ready for some of these.

Assess your progress and be honest in reporting this. It is difficult to do, but before you write your slides or the conclusions of your discussion paper it is good to step back and try to objectively assess the extent that the results you have demonstrate that you have achieved your stated aim or that they support your conclusion. In particular, try to distinguish between what you hope to achieve with your model in the future and what you have currently *proved* (i.e. beyond audience doubt). Try to think of ways

in which you might be wrong about this, or alternative hypotheses that would explain the same conclusion. A workshop is a good venue for eliciting critiques of current weaknesses and helpful suggestions for how to address these – not for showing how perfect a modeller you are. Although other modellers might forgive you for being over-enthusiastic about your results, they will prefer honesty. If there are non-modellers attending to your results they will be annoyed when they find out they have been conned (as it will feel to them) – they expect more from us than this.

Further reading and resources. A good, general motivation for the anticipating, mitigating and being honest about possible weaknesses in research is found in [25]. [9] is a example of this approach, discussing seven common modelling purposes along with an analysis of some of the ways each of these might fail.

3 For those who want to publish an ABM paper in an established journal or at a top-level conference (*level 3*)

You are getting serious, thinking you have some robust and significant research to report and so are thinking about publishing in a journal paper or top-level conference where the main audience is other academics, modellers or your peers. If your results might affect people's lives (even indirectly) you should also do the steps in level 4.

General advice. We are all, to some extent, specialists – those reading your work will not have the time to check it, which means that any such paper is far more useful if they can simply trust it. Once a paper starts to get cited in the literature this is hard to undo, even if the results turn out to be wrong because papers tend to be cited even after being refuted. Thus, there is a strong obligation to *not* present flawed models or over-hyped conclusions – such papers impair scientific process and cause harm if later used to influence decision makers. For example, even if you did not intend it, your code or modelling structure might be used in a future model, developed by others, that does affect people's lives. This is a social dilemma – everybody has short-term, individual reasons for rushing to publish (they need a paper quickly for their career, they can't be bothered etc.) but this can be detrimental to the whole field in the longer-term. For these reasons, at this level, one should be careful, cautious and methodical. If the model was developed in a somewhat chaotic or exploratory manner, then you need to go back and re-do it properly [17]. Reporting cautious but reliable results that is better than hyped results that may disappoint later when limitations come to light.

Make your model "open" – freely available with complete, multi-aspect documentation and appropriate licensing. Publishing at this level means that you are entering a dialogue with your peers – inviting discussion and critique. It is almost impossible to see all the flaws and limitations in one's research, so we have to rely on others to do this. This "checking each other's homework" is one of the pillars of good science, and this distinguishes it from many other endeavors. In particular, the analysis, reproduction [8] or comparison of each other's work [4, 6] is important for establishing its reliability – without it you just do not know if your code is as you intended it. Unfortunately getting someone else to spend the time and effort to do this is not easy, and this

probably has to happen post-publication (if you are lucky enough to get someone to do this).

However, regardless of whether someone actually reproduces or further develops your model, it is your responsibility to make your model *as accessible as possible* to other researchers. There is a problem here – understanding someone else's model is hard and needs a lot of time and effort. For this reason one needs a *variety* of kinds of model documentation of the same model to help them do this, including the following.

- *A narrative account of the model* which gives the authors account of the important aspects of the model, including its declared purpose (by which they want it to be judged) and its original context. Knowing *what* the author thought was important is a good starting point for understanding a model as long as one bears in mind that the authors might be mistaken (e.g. [8]). This should be supplemented by *a few example results* which give the reader a feel for what the model typically does.
- A precise description of *the conceptual model* that the authors aimed their implementation at. This is a relatively high-level but complete account of all the aspects deemed intentional or significant for the model [22]. This should only include aspects thought to be *essential* for producing the significant model results and not conflate this with what was implemented. Techniques such as UML [1], more formal specification languages [21] and formal "ontologies" [19] can be helpful for this.
- *A complete description of the implemented model*. Standards such as the "ODD" standard are helpful here as they can remind you to include aspects you might otherwise forget about. ODD descriptions can be long so they are much better as an appendix or other kind of supplement supplied with the main paper or model code.
- A *description of the important experiments done* with the model with some example results (see step below for more details).
- Some *account of how the model was developed* indicating variations tried that turned out not to be so important to the declared results. This helps other modellers know what else has and has not been tried with the model. The "TRACE" documentation standard is helpful in this regard [2]. Parameter settings/variations of the model that are not important are usefully included in appendixes or supplementary material.
- A *list of assumptions behind the model formulation* is very useful to others, especially if this indicates the authors' assessment of their reliability and justification. If some aspects of the model come from existing theory, the appropriate references should be included, however only citing such sources is not sufficient as theories are often vaguely formulated – additional assumptions are usually necessary in order to obtain an implementation. If some elements are derived from data or other evidence then how this was done should be described (e.g. using the RAT-RS list of questions [1]. Whilst highlights from these are use in the main text, a more complete account should be left to an appendix or supplementary material.
- Finally, the code itself, including comments, *should be made available on a public archive*. This is important so that (a) others can try out and mess with your code but also (b) it is the ultimate reference document for what was implemented – not many people will read the code, but if there is a question that is not answered by the other documentation then one can sort this out using the code itself. The code should be

accompanied with some instructions as how to run it (e.g. libraries or extensions that are needed) and an appropriate open license so that others can legitimately re-use and play with your code [18].

Use well-structured code which includes internal checks. The code should be easily comprehensible. It should be

- *well structured*, having a logical structure with related code close to each other and using a limited set of control structures – chucks of code should not be too long but divided into separate, appropriately labelled sub-procedures or blocks;
- *well formatted*, with spacing between procedures, indenting that indicates the inside of loops or alternative paths in conditional statements and informative headings to indicate separate sections of the code – collecting related code together;
- *with meaningful variable/procedure/function names* that would indicate what they are to people unfamiliar with them;
- *with lots of comments* telling a reader what that bit of code does and why – this should include references to the other documentation that link to this – in particular anything important, non-standard or are something the author has doubts about.

Signs of well-structured code is that is reads as close to ordinary language as the programming language allows.

Bugs can be subtle creatures, and it is likely that many of them are never identified. This may or may not be important for the results discussed, but can often have an impact when the model is used in a different context to the one it was developed for. Thus as well as actively testing the code (described in step 2.2 above), one should implement code that implements internal checks. That is code that does not aid its normal running that checks internal, intermediate results produced by parts of the code for consistency with the conceptual model. Some of these might be automatically done by the programming language you are using, but it is better to be explicit, so that a re-implementation in another language might include equivalent checks. Such inline checks might be for:

- that an intermediate result be the right type of entity (e.g. distinguishing the letter "O" and the number "0"),
- that it is in the right range (e.g. all prices should be positive),
- that conserved entities (money, number of agents etc.) total to the same amount,
- that numbers that should be truly zero be exactly that (and not something like 0.0000000000000000001 due to a rounding error).
- That a filename be of the right format.

Having many statistical measures, visualisations and graphs concerning what is happening in the model can also help, since it may become immediately obvious if something has gone wrong just be looking at these.

If these checks slow down the running of your model too much, then introduce a switch in your code to turn them on and off. Whilst developing and testing your model you can have these on, but whilst doing many experimental runs you can turn them off.

Justification and full description of runs and results. The steps above are concerned with the description, provision and checking of the model as a static object of

concern. Now is the time to describe and document what you did with the model. In other words, what runs you did and what results you obtained. This is necessarily partial – there are, almost always, far too many possible parameter/starting configurations to try them all out, let alone describe them all. We recommend that you separate out possible runs/results as follows, for each 'run' careful describe the parameter/starting configurations/input data you use and the corresponding significant outcomes that result.

1. Describe some runs that give the reader some idea of the typical behaviour or behaviours of the model in the main text. This gives the reader a reference point.
2. Exhibit some runs that show the behaviour that you are primarily interested in.
3. Do a 'sensitivity analysis'. In other words, systematically vary the key parameters (or combination of parameters) showing how some key outcome measures change as a result [23]. This gives the reader a more general idea about what the model does given different parameters, but also indicates which settings the model is very sensitive to, and which only make a gradual or minor impact.
4. Finally, describe the key runs that test or otherwise demonstrate your key explanation/hypothesis. The best papers explicitly formulate their key hypothesis concerning the behaviour of the simulation (how outcomes relate to its structures, processes, etc.) and then seek to falsify that hypothesis in a series of experiments.

There are two comments concerning all these descriptions of runs and outcomes. *Firstly*, that it is very useful for those seeking to reproduce the simulation to have some precise outcome measures for precise sets of outcomes. This enables those reproducing the model to check they have got it exactly right. *Secondly*, there is a lot of knowledge implicitly 'packed into' what concerning the simulation outcomes is considered significant and what is not. In deciding what to show and what to pass over, authors are making a statement about what in the simulation is core and what happenstance. To take a trivial example, exactly which pseudo-random number generator is used in the simulation is probably not important, and one should get *essentially* the same results if another was, in fact, used. The essence of the results might be the average of a value over many independent runs or the qualitative behaviour in the longer term, and not lie in the precise value of an outcome measure. It is good practice to explain why one is highlighting some aspects of a simulation run and not others.

Strong validation of any conclusions. You have a presented a fully documented and tested simulation model, and then you have described sets of simulation runs. Now comes the most important (and often most neglected) part of the paper – the argument from the results to your conclusion. This argument should be as water-tight as possible, and if it is not then moderate the strength of the conclusions until it can be. The conclusions should relate strongly to the declared model purpose. If you were aiming to explain some phenomena then the simulation experiments should show that the workings of the model strongly support that explanation. However, validation is a much contested term, especially in ABM. One way of thinking about validation is the process by which you rule out the different ways in which you might be mistaken about your conclusions. This involves the following steps:

1. Be precise about your intended conclusions, which should be an example of your declared modelling purpose.
2. Describe your main reasoning, starting with your results and the design of your model and ending with your conclusions.
3. Think of all the ways in which you could be mistaken about these conclusions.
4. Design experiments or mitigating measures to prevent or rule-out each of these ways. The most important of these should be described in the main body of the paper.
5. If there are any ways in which you might be wrong, that have not been ruled out, then you should moderate your conclusions to take these into account.

The exact steps taken and the arguments used depend upon your conclusions, so it is not possible to be more specific here, but see Edmonds et al (2019) for an analysis of some of the main risks and mitigating measures that might be relevant to different kinds of modelling purpose, whose summary table of risks is reproduced in Table 1 below.

Table 1. Summary of Different Modelling Purposes, their features and risks (from [9]).

Modelling Purpose	**Essential features**	**Particular risks (in addition to that of not achieving the relevant essential features)**
Prediction	Anticipates unknown data	Conditions of application unclear
Explanation	Uses plausible mechanisms to match outcome data in a well-defined manner	Model is brittle, so minor changes in the set-up result in bad fit to explained data; bugs in the code
Description	Relates directly to evidence for a set of cases	Unclear provenance; over generalisation from cases described
Theoretical exposition	Systematically maps out or establishes the consequences of some mechanisms	Bugs in the code; inadequate coverage of possibilities
Illustration	Shows an idea clearly as a particular example	Over interpretation to make theoretical or empirical claims; vagueness
Analogy	Provides a way of thinking about something; gives insights	Taking it seriously for any other purpose
Social learning	Facilitates communication or agreement	Lack of engagement; confusion with objective modelling

Using honest and cautious language. Scientific language – that in a formal scientific publication – should be careful and precise. That is, it should communicate the research honestly and transparently. It should not hype the importance, results, or conclusions. This is difficult to do for two reasons: (a) in your enthusiasm you may not have noticed all the possible weaknesses and limitations and (b) even if you are crystal clear in your mind it may be hard to find the best language to communicate this in.

- For each statement written you should think "is this precisely true/justified?" and "is this crystal clear?" and if there is any doubt about these, reword it so it is.
- Language used about the model should be clearly distinguished from that about what it models (the intended target of the modelling) [5]. It is good practice to have different sections where you talk about the intended phenomena or ideas of what the model is about and those where you describe the implemented model and its results. It can also be helpful to use different words to indicate entities in the model and what they represent. Never make one statement that you think holds for both model and target phenomena – even if avoiding this involves some repetition.
- We all hope that our models will achieve great things in the future, but should limit ourselves to describing what has currently been achieved. For example, we should not say something like "This model may be helpful to Y in preventing X" if we have not shown that the model can reliably predict X or the conditions when it is more likely. This includes optimistic anticipation about the empirical correctness.
- Whether you have made clear the weaknesses or limitations of your research.
- It is easy to *imply* achievements or uses, even if one does not make any such claims explicitly but one should make an effort not to do this. A common approach that can cause confusion is to motivate the work in strong language and then present the research. For example, stating that solving a particular problem is of high importance, then present a model leaving the impression that the model is a significant step towards solving that problem. Ideally, one should anticipate possible misconceptions and correct these using explicit statements (e.g. "this is just a proof of concept").

If the reader has to work hard to understand what the research has achieved by reading footnotes, diving into model detail or reading between the lines like a lawyer, then you have not done a good job of describing your research. Rather, the work should be easy to understand and assess, with the authors making it clear what was achieved.

Further reading and resources. The starting point here is [9]. For the systematic design of experiments see [16]. To check you have run your simulations enough times to justify you conclusions see [20]. For more on UML see [1]. More about keeping the conceptual model and implementation distinct see [22]. For active steps to prevent 'bug' or model artefacts see [11]. For more on sensitivity analysis see [23].

4 For those whose modelling might influence decisions that affect people's lives (*level 4*)

Many grant applications promise "policy relevance" in some form or other. When funded, the resulting projects are under an obligation to, at least, try to interest policy actors in their results. Others want their work to have relevance outside academic circles because they want to be useful or they feel an obligation to do so. Due to the potential dangers, this level calls for the highest standards of consultation, validation, transparency, rigour, and caution [7]. We are not going to describe this level here because it would take up too much space and we would like to build more consensus upon it before publishing. A future guide could be written for policy actors interacting with modellers.

Acknowledgements

Thanks to: (1) all those who gave me useful comments on this paper, (2) all those Agent-Based Modellers (most of whom entirely well-intentioned) who have presented flaky or flawed research due to a lack of rigour (something we have been guilty of, on occasion), (3) the Grumpy Old Men and Women[1] who are our friends and with whom we have spent many a happy hour at conferences complaining about the current state of modelling, and (4) all those special people participating in the roundtable on rigour at Social Simulation 2021. This would not have been written without you.

References[2]

1. Achter, S., Borit, M., Chattoe-Brown, E., Palaretti, C. and Siebers, P.-O.(2019) Cherchez Le RAT: A Proposed Plan for Augmenting Rigour and Transparency of Data Use in ABM. *RofASSS*, 4th June 2019. https://rofasss.org/2019/06/04/rat/
2. Augusiak, J., Van den Brink, P. J., & Grimm, V. (2014). Merging validation and evaluation of ecological models to 'evaludation': a review of terminology and a practical approach. *Ecological Modelling*, **280,** 117-128.
3. Bersini, H. (2012) UML for ABM. *JASSS*, **15**(1), 9. http://jasss.org/15/1/9.html
4. Bithell, M., Chattoe-Brown, E. and Edmonds, B. (2021) The Systematic Comparison of Agent-Based Policy Models - It's time we got our act together!. Social Simulation Conference, Warsaw, 2021. http://cfpm.org/model-comparison/
5. Edmonds, B. (2020) Basic Modelling Hygiene - keep descriptions about models and what they model clearly distinct. *RofASSS*, 22nd May 2020. https://rofasss.org/2020/05/22/modelling-hygiene/
6. Edmonds, B. (2020) Good Modelling Takes a Lot of Time and Many Eyes. *RofASSS*, 13th April 2020. https://rofasss.org/2020/04/13/a-lot-of-time-and-many-eyes/
7. Edmonds, B. & Adoha, L. (2019) Using agent-based simulation to inform policy – what could possibly go wrong? In Davidson, P. & Verhargen, H. (Eds.) (2019). *Multi-Agent-Based Simulation XIX*. LNAI, 11463, Springer, 1-16. DOI: 10.1007/978-3-030-22270-3_1
8. Edmonds, B. and Hales, D. (2003) Replication, Replication and Replication - Some Hard Lessons from Model Alignment. *JASSS,* **6**(4), 11. http://jasss.org/6/4/11.html
9. Edmonds, B., le Page, C., Bithell, M., Chattoe-Brown, E., Grimm, V., Meyer, R., Montañola-Sales, C., Ormerod, P., Root H. & Squazzoni. F. (2019) Different Modelling Purposes. *JASSS*, **22**(3), 6. http://jasss.org/22/3/6.html
10. Epstein, J. M. (2008). Why model?. *JASSS*, 11(4), 12. https://jasss.org/11/4/12.html
11. Galán, J. M., Izquierdo, L. R., Izquierdo, S. S., Santos, J. I., del Olmo, R., López-Paredes, A. and Edmonds, B. (2009). Errors and Artefacts in Agent-Based Modelling. *JASSS*, 12(1), 1. http://jasss.org/12/1/1.html
12. Gilbert, N. and Troitzsch, K. G. (2005) *Simulation for the Social Scientist* (2nd Edition). Open University Press.

[1] Actually, many of them are neither old nor *very* grumpy and some may be non-binary.

[2] In the references below, *JASSS=Journal of Artificial Societies and Social Simulation*, *RofASSS=Review of Artificial Societies and Social Simulation*, and *SSC=Edmonds, B., Meyer, R. (eds) Simulating Social Complexity. Understanding Complex Systems. Springer.*

13. Grimm, V. et al. (2020) The ODD Protocol for Describing Agent-Based and Other Simulation Models: A Second Update to Improve Clarity, Replication, and Structural Realism. *JASSS,* **23**(2), 7. https://www.jasss.org/23/2/7.html
14. Hales, D., Rouchier, J., and Edmonds, B. (2003) Model-to-Model Analysis. *JASSS*, **6**(4), 5. https://www.jasss.org/6/4/5.html
15. Kuhn, T. S. (1962) *The Structure of Scientific Revolutions*. Chicago, IL: University of Chicago Press.
16. Lorscheid, I., Heine, B. O., and Meyer, M. (2012). Opening the 'black box' of simulations: increased transparency and effective communication through the systematic design of experiments. *Computational and Mathematical Organization Theory*, **18**(1), 22-62.
17. Norling, E., Edmonds, B. and Meyer, R. (2017) Informal Approaches to Developing Simulation Models. In *SSC*, 61-79.
18. Polhill, J. G. and Edmonds, B. (2007) Open Access for Social Simulation. *JASSS*, 10(3), 10. http://jasss.soc.surrey.ac.uk/10/3/10.html
19. Polhill, G., Salt, D. (2017). The Importance of Ontological Structure: Why Validation by 'Fit-to-Data' Is Insufficient. In: *SSC*, 141-172.
20. Seri, R., Secchi, D. (2017). How Many Times Should One Run a Computational Simulation?. In: *SSC,* 229-251.
21. Siebers, PO., Klügl, F. (2017). What Software Engineering Has to Offer to Agent-Based Social Simulation. In: *SSC*, 81-117.
22. Stepney, S., and Polack, F. A. (2018). *Engineering Simulations as Scientific Instruments: A Pattern Language*. Springer.
23. ten Broeke, G., van Voorn, G. and Ligtenberg, A. (2016) Which Sensitivity Analysis Method Should I Use for My Agent-Based Model? *JASSS*, **19**(1), 5. http://jasss.org/19/1/5.html
24. Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, **2**(1), 230-265.
25. Wilson, J. L., & Botham, C. M. (2021). Three questions to address rigour and reproducibility concerns in your grant proposal. *Nature*, **596**(7873), 609-610.