APPENDIX B: JAVA SPECIFICATION CHECKING: SURVEY AND INTERVIEW RESULTS

Anton.Dil

THE OPEN UNIVERSITY

Appendix B: Survey and interview results – D3

This appendix contains more information about the survey and interview process with tutors, as well as more detail on the coding and thematic analysis.

1 The tutor survey

A survey was used in this study to explore tutors' use of the CheckM250 software and their attitudes towards marking tools and other resources we currently make available to them, as well as resources we might offer in future. Tutors were asked to rate the usefulness of tools both to themselves and to students.

The survey was created on Bristol Online Surveys (now known as Online Surveys) [1], and used a mixture of open and closed questions, with open questions providing an opportunity to expand on reasons for the closed responses. An export of the full results is available in the Online Survey Responses PDF.

Some more information on the responses is provided below.

1.1 Does students' code compile?

We asked tutors how often students' code compiled and whether they were in the habit of fixing students' compilation problems. This is significant from the point of view of automated marking, because the CheckM250 tool, in common with other tools such as style checkers and unit testing tools, requires code to compile in the first instance. If students' code doesn't compile very often, automated marking tools of these sorts are not very useful.

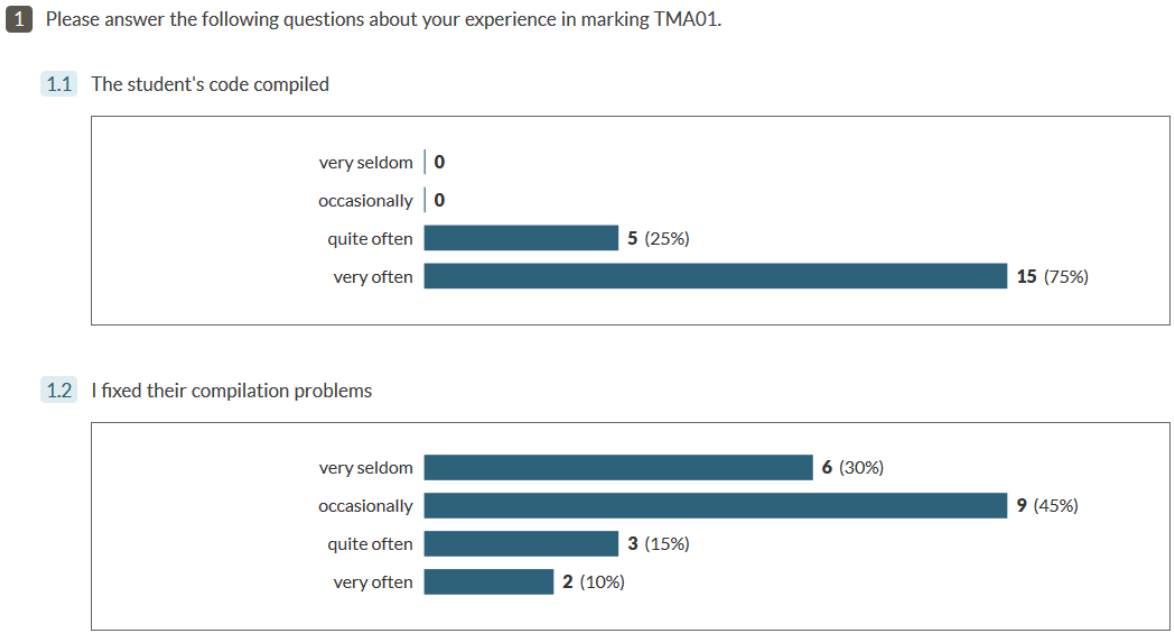

Figure 1: Compilation errors

Tutors suggested that students' code compiles most of the time (75%), but that they only occasionally or seldom fix those problems (75%). Interviews revealed that this was related to the amount of effort required to fix the code. If a quick fix seemed possible, tutors were more likely to correct the code.

1.2 How long does it take tutors to mark TMA01?

We wanted to explore whether use of a marking tool would speed up marking, so times to mark the TMA on which the tool was first used (TMA01) were queried.

The reported marking times for TMA01 had a median of 39 minutes. Considering this is a short assignment, worth only 10% of OCAS, this was surprisingly high.

Considering only tutors who indicated they used the CheckM250 tool (N=8) marking time had a median of 47 minutes, compared to 44 minutes for non-tool users (N=12), suggesting using the tool incurred a small amount of extra time or that tool-adopting tutors spend more time on marking in general. There was not enough data available to infer statistical significance of differences between the tool using and non-tool using groups responses.

A time difference is not unexpected, since tool use requires an extra process to be included when marking, involving opening the tool, selecting the 'checks' (specification) file, and clicking a 'Run Checks' button. The results of the tool would then need to be interpreted and (potentially) included in the marking of the assignment in some way.

However, two tutors during interview suggested that with more experience the CheckM250 tool might help speed up marking, depending on how it is adopted during the workflow. It can be used to determine where to focus marking effort, or as a check after marking. In the former case, it may help save time.

1.3 Reasons for not using the marking tool

The majority of tutors did not use the CheckM250 tool and the majority of survey respondents did not use it. The reasons reported by the survey respondents (N=12) for not using the tool were categorised and their frequencies counted. A categorisation of the responses is given below:

1. Didn't know about it or know enough about it (50%)

2. Too busy in other areas (33%)

3. Didn't feel the need for it (33%)

The third category might be further broken down as follows:

> 3a The TMA was simple enough to mark without the tool
>
> 3b The tutor feels experienced enough to mark the TMA without the tool.

1.4 Frequency of use by the tool users

Half of the tutors using the tool (4/8) did so on all of their assignments. Three tutors chose to use the tool in a few cases only, while one used the tool on most of their assignments.

It was unclear from the survey to what extent non-compilation of students code played a part in this, but we can infer from the previous responses that it would account for some of the non-usage.

1.5 Did the tool help spot errors?

Most tutors (75%) believed the tool was somewhat or very likely to spot errors they'd have missed.

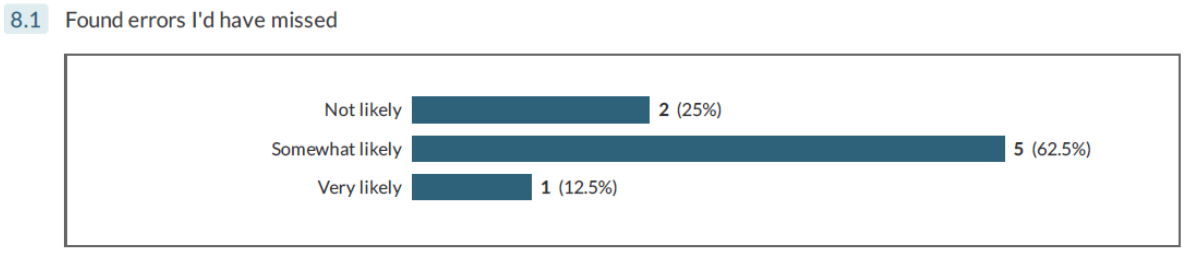

Figure 2: Found errors I'd have missed

Since the responses to this question were based on usage in the first TMA, which was a fairly simple exercise, it is quite possible that a tutor could have spotted all the errors students made without having to use the marking tool. However, as the complexity and length of the assignment increases, the tool can be expected to outperform human markers consistently.

1.6 Would tutors recommend the tool and use it in future?

Most CheckM250 users (5/8) would be very likely to recommend it to other tutors, while 2/8 were somewhat likely to recommend it.

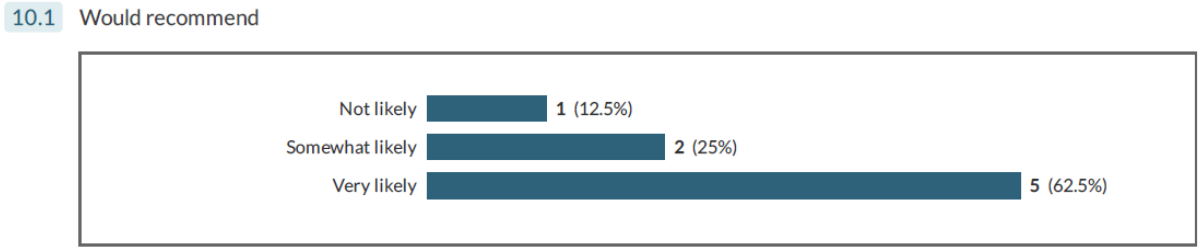

Figure 3: Would recommend

The main concern with recommending the tool is the time it takes to use (3/8 respondents) although one tutor thought the tool helped them mark faster. Some tutors using the tool had missed an update that resolved a usability problem which led to a slower workflow. The way in which the tool is incorporated in the workflow also impacts on the time it takes to use.

One tutor suggested that they would be more likely to use the tool if it included unit testing, so soliciting another layer of automated marking be added to it.

The fact that CheckM250 will spot errors that tutors may miss is valued by almost all. One tutor appreciated *confirmation* that they had spotted an error or had not missed errors, so was using the tool to monitor their own work.

The same distribution of responses is given for a desire to use the tool in future.

1.7 Does the tool change the approach to marking?

Mostly, the tool did not change the way tutors marked, although it functioned for some as

1. A backup check that no specification errors in the student's solution had been missed
2. A way to locate parts of student's code to be commented on in their solution document.
3. An interactive tool

This indicates that the tool could be used before, during, or after marking.

Two open responses to this question are provided below:

> *I tend to only open their code if necessary - mostly errors can be spotted from eyeballing their solution documents.*

> *Usually I review the solution document first then look at the code if necessary. This time I ran the code through CheckM250 before running it with my test class and finally examining the solution document. This meant that by the time I looked at the solution document I knew where the errors were.*

This confirms our anecdotal understanding that some tutors do not usually examine students' code in its software form (in Bluej), and instead concentrate on Microsoft Word document submissions, but other evidence from this study suggests that the option of clicking a button to examine students' code in their projects could be useful to markers.

1.8 Code style concerns

In preparation for producing a style-marking tool as part of further work planned after this project, we asked tutors about their concerns with regards to code style.

Issues tutors raised that were amenable or somewhat amenable to automated style checking were:

- Formatting of code, with appropriate spacing
- Use of local variables that shadow instance variables
- Missing method comments
- Use of if-else constructs rather than single-line return statements

Also raised as issues, but not fitting automated checking of style very well were:

- Use of obscure or complex code
- Use of meaningful variable names

Some aspects of obscure or complex code might be detected by a style checking tool, but others might require bespoke code to be written. For example, use of the ternary ? operator might be detected by a style checker and flagged. Complex code might possibly be picked up by a complexity measure such as McCabe's cyclomatic complexity, but metrics such as this are not typically included in standard style checking tools such as PMD [2] and CheckStyle [3].

Meaningful variable names are highly context dependent, and so it is unlikely a rule could be written to detect less meaningful names. It is, however, possible to detect short variable names, which are likely to be less meaningful, depending on the context in which they are used, and these might be flagged.

Other comments on tool support for assessing code quality

One tutor suggested that whilst we could flag violations of coding conventions (i.e., style issues), students may ignore such advice unless there is a summative aspect to them.

Some tutors were concerned that use of tools, if student-facing, could increase students' study time and be confusing to weaker students, or dispose students to rely on tools in future. We consider that it is debatable whether using tools to spot errors is really problematic, however. The module team considers them to be part of a programmer's toolbox, not a substitute for understanding. We also feel that bringing such issues to students' attention, whether through tools or tutors, is a step to helping students recognise that there is an issue to be addressed.

There was also the suggestion that automated tests might be seen as comprehensive, when in fact they can only cover some aspects of code quality. This means we would need to explain how the tool fits into the work and marking process for students and tutors.

1.9 Tutor evaluation of tools

We asked about the utility of the following resources:

- The module text
- Existing software activities
- Practice TMA quizzes*[1]
- Debugging activities*
- Unit tests*
- Walk-through videos on TMAs*
- Style checking tool*
- Compilation error help*[2]
- Structural specification checking[3]

Note, starred (*) items were not available to tutors at the time of the survey, so the responses here were based on tutors' professional judgement.

A Likert scale was used to solicit ratings as 'not at all', 'slightly', 'moderately', 'very' or 'extremely' useful. Based on sums of percentage ratings, tutors valued these resources as follows, ordering by very + extremely, then by moderately + very + extremely. Responses were received both from tool users (N=8) and non-tool users (N=12). Tables 1 and 2 show the results.

Table 1: Overall resource utility to students, judged by tutors

	V+E	**M+V+E**
Existing software activities	85	100
Debugging activities	80	100
Module text	75	100
Practice TMA quizzes	74	95
Walk-through videos on TMAs	60	85
Unit tests	58	79
Compilation error help	40	70
Style checking	39	72
Structural specification checking	**33**	**72**

The items in the top half of the table are resources that tutors might not engage with deeply, although students might ask questions about them and some might be used for tutorial purposes. They include the main resources we already provide: the module text and the existing software activities, with which tutors are already very familiar. It is notable that tutors felt strongly that more debugging activities would be helpful.

All the suggested tool-based resources come in the bottom half of the table for utility to students, as judged by tutors, when considering the sum of very + extremely ratings. Of these tools, unit tests are considered the most useful to students, while the remaining three are of similar usefulness to one another, but specification checking help comes last.

[1] Available to students as formative quizzes using CodeRunner questions
[2] Available to students on formative Practice for TMA02 and Practice for TMA03 quizzes.
[3] Available to students on Practice for TMA03

I believe this indicates a misunderstanding on some tutors' part of how unit testing would work in practice. Typically, a unit test will check the functionality of a method, by supplying certain inputs and checking whether expected outputs are achieved. However, it is not possible to perform unit tests such as these unless the structural specification for methods is correct, because such tests would simply not compile, due to the absence of required methods. I concluded therefore that tutors were simply more familiar with the idea of unit testing and recognise its usefulness already. It is worth bearing in mind also that the majority of respondents had not used CheckM250 and so were less likely to understand how the structural specification checking tool operates in this respect.

The CheckM250 tool checks a number of features demanded by the questions we set that would not normally be covered by unit testing, such as the presence of instance variables with certain names or inheritance relationships.

Further, I argue that the separation of concerns between structure and functionality is desirable, both for students' understanding of how to write a correct program, and for markers to assess whether a program is correct.

The rating of tools was different for users of the CheckM250 tool, as shown in Table 2.

Table 2: Tool use to students, **as rated by tutors who used the tool**

	V+E	**M+V+E**
Existing software activities	88	100
Walk-through videos on TMAs	75	100
Unit tests	75	88
Debugging activities*	63	100
Module text	63	100
Structural specification checking	**63**	**88**
Style checking *	57	100
Practice TMA quizzes	57	86
Compilation error help	38	75

It is clear that tool users rate the CheckM250 tool more highly, suggesting that their experience has been a positive one, even though unit tests are still rated more highly by this group.

We can conclude that tutors overall were more concerned with the functionality of students' software than with its structure or style. However, as already noted, one cannot test whether methods behave appropriately if students have not provided those methods.

Further, this study has reminded us that passing unit tests does not of itself ensure correctness of structural specification. Structural correctness is required in order for unit tests to be passed, but passing unit tests does not confirm structural correctness.

1.9.1 Correlations

This section explores correlations between ratings of tools by tutors. We found that compilation error help is not well correlated with ratings for the other tools. This is to be expected, as tutors would not particularly need this tool, which is more appropriate for students.

Correlation between ratings of resources as useful to students

Table 3 concerns utility of resources to students as judged by tutors, correlated using Spearman's rank order coefficient, rho.

Folder: \OneDrive\Research\CheckM250\Survey\responses

File: results-for-code-marking-2018-04-11-1310.sav

Table 3: Correlations between ratings of resources by tutors, all ***for student use***

Correlations

		the module text	existing software activities	compilation error help (like BlueJ's '?' facility)	specification checking (like CheckM250)	unit tests	code style checking	practice TMA quizzes	wa v
the module text	Correlation Coefficient	1.000	.262	-.168	.216	.290	.384	-.053	
	Sig. (2-tailed)	.	.264	.479	.390	.229	.116	.831	
	N	20	20	20	18	19	18	19	
existing software activities	Correlation Coefficient	.262	1.000	.498*	.273	.406	.528*	.514*	
	Sig. (2-tailed)	.264	.	.025	.274	.084	.024	.024	
	N	20	20	20	18	19	18	19	
compilation error help (like BlueJ's '?' facility)	Correlation Coefficient	-.168	.498*	1.000	.265	.225	.316	.346	
	Sig. (2-tailed)	.479	.025	.	.288	.354	.202	.147	
	N	20	20	20	18	19	18	19	
specification checking (like CheckM250)	Correlation Coefficient	.216	.273	.265	1.000	.801**	.648**	.291	
	Sig. (2-tailed)	.390	.274	.288	.	.000	.005	.257	
	N	18	18	18	18	18	17	17	
unit tests	Correlation Coefficient	.290	.406	.225	.801**	1.000	.669**	.216	
	Sig. (2-tailed)	.229	.084	.354	.000	.	.002	.389	
	N	19	19	19	18	19	18	18	
code style checking	Correlation Coefficient	.384	.528*	.316	.648**	.669**	1.000	.297	
	Sig. (2-tailed)	.116	.024	.202	.005	.002	.	.248	
	N	18	18	18	17	18	18	17	
practice TMA quizzes	Correlation Coefficient	-.053	.514*	.346	.291	.216	.297	1.000	
	Sig. (2-tailed)	.831	.024	.147	.257	.389	.248	.	
	N	19	19	19	17	18	17	19	
walk-through videos on TMAs	Correlation Coefficient	.262	.421	.082	.326	.244	.465	.403	
	Sig. (2-tailed)	.265	.065	.730	.187	.314	.052	.087	
	N	20	20	20	18	19	18	19	
software debugging activities	Correlation Coefficient	.124	.367	.204	.509*	.233	.400	.114	
	Sig. (2-tailed)	.604	.112	.388	.031	.337	.100	.642	
	N	20	20	20	18	19	18	19	

* are statistically significant at the 0.05 level (2-tailed)

** are statistically significant at the 0.01 level (2-tailed)

Points to note:

1. There are no significant correlations between ratings of the module text and other resources.

2. The highest correlations are between ratings of tools, with rating of specification checking and unit testing (Spearman's rho = 0.801) the highest, which is significant at the 0.01 level. The 95% confidence interval for this result is 0.556 to 0.918, indicating a moderately high degree of correlation. Also significantly correlated at the 0.01 level are specification checking and style checking.

This suggests that if a tutor rates one of these tools as useful, they will also rate the others as useful, and vice versa. In other words, tutors appear to rate tools in general as useful or as less useful in general. Table 4 presents a subset of the correlations, for tool use, rated by tutors for student utility.

The exception to this rule is Compilation Error help. (Rating of compilation error help is correlated at the 0.05 significance level with rating of existing software activities on the module.) This is not surprising, as experienced coders could be expected to be capable of understanding compilation errors without tool support.

So, there is a tendency to rate all of the three new tools being proposed either well or poorly by tutors, suggesting a disposition either for or against tools or for or against new tools.

Table 4 presents a subset of these results – the ratings of the tools.

Table 4: Correlations between ratings of tools for student use

Correlations

		spec	unit tests	style	compil.
specification checking	Correlation Coefficient	1.000	.801**	.648**	.265
	Sig. (2-tailed)	.	.000	.005	.288
	N	18	18	17	18
unit tests	Correlation Coefficient	.801**	1.000	.669**	.225
	Sig. (2-tailed)	.000	.	.002	.354
	N	18	19	18	19
code style checking	Correlation Coefficient	.648**	.669**	1.000	.316
	Sig. (2-tailed)	.005	.002	.	.202
	N	17	18	18	18
compilation error help	Correlation Coefficient	.265	.225	.316	1.000
	Sig. (2-tailed)	.288	.354	.202	.
	N	18	19	18	20

**. Correlation is significant at the 0.01 level (2-tailed).

Correlation between ratings of tools for tutor use

Table 5 presents the same results, this time according to tutors' ratings of the tools for their own use.

Table 5: Tool ratings for tutors' own use

Correlations

			compilation error help	specification checking (CheckM250 tool)	unit (functionality) testing	code style checking
Spearman's rho	compilation error help	Correlation Coefficient	1.000	.079	-.342	.021
		Sig. (2-tailed)	.	.756	.165	.931
		N	19	18	18	19
	specification checking (CheckM250 tool)	Correlation Coefficient	.079	1.000	.650**	.494*
		Sig. (2-tailed)	.756	.	.003	.032
		N	18	19	19	19
	unit (functionality) testing	Correlation Coefficient	-.342	.650**	1.000	.368
		Sig. (2-tailed)	.165	.003	.	.121
		N	18	19	19	19
	code style checking	Correlation Coefficient	.021	.494*	.368	1.000
		Sig. (2-tailed)	.931	.032	.121	.
		N	19	19	19	20

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Again we can see that the correlations between ratings of the tools indicate that compilation error help is not highly correlated with choices on the other variables, but unit testing, specification checking and style checking are all correlated at a statistically significant level (0.01 two-tailed sigma). The highest correlation is between unit testing and specification checking at 0.752. Style checking is correlated more weakly, but still significantly correlated as useful or not useful to tutors, compared to specification checking.

Just as tutors appeared to be disposed for or against tool use for students, these results suggest that tutors are disposed for or against tool use for themselves.

Taking these two sets of correlations into account, the results suggest that the utility of the proposed tools themselves may be less significant than the predisposition of the tutor towards tools. In the case of personal use, this may be accounted for by unwillingness to spend extra time on using a tool, belief that the tutor will anyway notice issues that tools find, that it is not necessary to be completely accurate in marking, or that there are other more important things for tutors to be commenting on than issues that these tools might discover in students' code. In the case of student use, it may be accounted for by a worry that students have too much to do, or will have too much to do (students' time concerns), or that the tool may somehow interfere with the students' learning in another way. These ideas are explored further in the interview analysis.

1.9.2 Associations

To provide further evidence for the assertions just made, we can look for an association between the rating of a tool for use by tutors and by students. In particular I am interested in whether tutors rate tools higher or lower generally, or whether they rate them according to utility to the tutor differently as compared to utility to students. This can be tested using several association statistics, and Somers' d is an appropriate and conservative one in this case.

Somers' d may be calculated on the assumption that the relationship between these two ordinal variables is symmetric. (We have no compelling reason to believe that either of these variables depends on the other, although it is possible that a high rating of using the tool as a tutor leads to increased rating of utility to students.)

Table 6: Associations between ratings of specification checking tools by tutors for student and their own use

Directional Measures

			Value	Asymp. Std. Error[a]	Approx. T[b]	Approx. Sig.
Ordinal by Ordinal	Somers' d	Symmetric	.625	.103	5.449	.000
		specification checking for tutors (like CheckM250) Dependent	.609	.114	5.449	.000
		specification checking for students (CheckM250 tool) Dependent	.642	.094	5.449	.000

a. Not assuming the null hypothesis.

b. Using the asymptotic standard error assuming the null hypothesis.

The symmetric score for Somers' d is 0.625, indicating a 62.5% improvement in prediction of the student utility rating if we know the tutor utility rating of the tool. This is a statistically significant result ($p < .000$). In other words, it appears that tutors are inclined to value the tool for use by both themselves and students, or for neither, or consider them of moderate utility to both themselves and students.

We can calculate the same measure of association for ratings of the other tools and this shows that similar relationships exist for Unit testing (Somers' d symmetric = 0.687, $p < 0.1\%$) and for Style checking (Somers' d symmetric = 0.706, $p < 0.1\%$). As expected, this relationship does not apply to Compilation Error help, as tutors should not particularly need help with compilation errors, whereas students may well do.

2 Tutor interview analysis

Interviews were conducted by the author with six experienced tutors over a period of two weeks. This section provides more information on the interview process. Anonymized transcripts of the interviews are included in a separate zip file.

2.1 Thematic analysis

Initial thematic analysis was performed using open responses from the survey questions, leading to the following themes:

1. Time (tutors find it slower to use the tool; they don't have time to use it).
2. Accuracy - The tool will always equal or outperform tutors within the scope of its operation. Some tutors thought that this was not so, or that the difference the tool might have made was not sufficient to warrant tool use. One aspect of this is the complexity of the marking task.
3. Complexity of the assignment being checked. If the assignment is very simple, some tutors will say the tool isn't worth using.
4. Self-sufficiency – some tutors may think they are as accurate as the tool or that using a tool is otherwise problematic. Similar comments are made regarding student use.
5. Focus – The tool highlights a particular aspect of code quality, but this may be misleading if other areas of code quality are not also highlighted.

Point 5 raises the question of what a tutor's focus in marking should be. Is it to detect the kinds of errors that tools can spot, or is it to help explain to students why their code is not working as expected? If we have tools that can spot certain kinds of issues with code quality, but spotting other kinds of issues is easier for humans, should tutors be focussing on those aspects of code quality that are less open to automated testing?

Ideas that did not fit well in this framework were:

6. Tutors saying they were not aware of the tool.

7. Comments on usability and enhancements of the CheckM250 tool.

I explored these themes further via a set of hour-long interviews with six tutors who used the tool.

These themes were then used as a basis for further exploration and analysis of interview transcripts, leading to a refined set of five themes:

1. Time available to engage with the tool
2. Quality of marking
3. Attitude towards tools
4. Focus of teaching and testing
5. The need for marking tools

In addition, there were questions asked during the interviews about other learning resources available in M250 that elicited comments not covered by the above themes, so extra themes were introduced to take note of these comments. There were also many unsolicited comments on

assessment in general on M250, which were of interest to the module team, but not of relevance to this report.

The key themes are discussed in more detail in the following sections.

2.2 Time

Remarks about time can be further subdivided into comments on tutors' and students' time.

There were 12 references to time in open responses in the survey, some arising from tutors explaining why they did not use the tool, typically suggesting they might not have enough time.

More detailed comments from the interviews are given below:

> *If I thought that a TMA was already taking me far too long, that would be a corner I might cut, because although it does help me find more issues, if I'm already spending hours marking a TMA, I might feel it wasn't helping. That's a shame, but that really depends how well the TMA is set.*

> *It might be a good idea if you got them to paste in the screenshot of what the tool has told them so we can just mark that we've seen it. That would probably save some time.*

> *It did mean it took a little bit longer but I didn't think that was a bad thing*

> *I thought the extra time that was spent there [on TMA01] came back in TMA02 where the code you got was better*

> *although I commented before and said that it took me longer, that wasn't a negative comment, … it made me recognise there were some things I had missed, which was good, and I think it was a good investment of time*

> *I think there's actually a saving in time as well that I hadn't really realised because what I did was I marked them and then ran the code to test them as it were and consequently when I'd missed something I had to go back and sort it out [i.e. if the workflow was different, time might be saved]*

> *I suspect you want me to say that the more tool support you give us the more time we've got for interaction with students, but it doesn't work like that, because … however long it takes us to mark a TMA, it takes us that long, you know. You're going to do it to the same standard. ….Those of who are retired or semi-retired have the luxury of being able to spend the time that it needs, I suppose, so it doesn't really make that much difference. …. We don't have 10 minutes per TMA and you do it as well as you can in that 10 minutes; it doesn't work like that.*

The following themes are reiterated or introduced:

1. If the TMA is taking too long to mark, the tool may be in the way.
2. The tool might actually improve speed if used appropriately in the tutors' workflow.
3. Students may produce better results on subsequent TMAs as a result of increased accuracy in marking on an earlier TMA.
4. Tutors may take as long as it takes to mark accurately.
5. The increased accuracy resulting from the tool use may compensate for the extra time taken

General remarks were made about how long it took to use the tool and the impact this had on marking.

2.2.1 Students' time

There is some worry about students' ability to cope with the extra information provided by tools such as CheckM250.

> *you don't want too much or you'll overload them. I thought what you had was just the right level.*

> *Most of these things would actually be very/extremely useful to quite a lot of students - but would increase the amount of study time required - and most students don't seem to have sufficient time for their studies.*

> *For very weak students more tools might actually be more confusing.*

There is a feeling that, for some students, specification checking and other testing tools (style checking and unit testing) would be useful, while for others they might confuse things.

2.3 Quality of marking

As the comments on time indicate, tutors are weighing up the utility of the marking tool with respect to the time it takes to use it and the increased accuracy it may bring to their marking. For some tutors this trade-off of increased time for increased accuracy is worth making, while for others it is not.

We know from our own experience in moderating exam marking that between (and within) marker variance can be a significant issue and we know that when it comes to marking code, different markers spot different errors in students' code.

We can at least be sure that the tool (if it works) would spot all of the errors it should. We can be sure that human markers will miss some of these errors, unless they are very diligent. (This is not to say that the quality of any marking on M250 is sub-standard. When a student makes a particular error, they often make it repeatedly. A marker may well pick this up in one place but not in another, and over the length of an assignment these omissions may not be significant.) For a recent survey paper on marker reliability see [4]

> *it gave me more of an idea of whether what they'd written was what they were expected to write, but it gave me an idea of how much they'd written as well because I think it tells you when methods are missing. So I already had an idea of if you like what was missing before I'd even gone to mark it*

2.3.1 Spotting errors otherwise missed

Four tutors commented on the survey that using the tool improved the quality of their marking, through bringing to their attention errors they had missed, and helping them improve their feedback to students accordingly.

In interviews it was also reported that the tool found errors that the tutor had missed. These errors particularly fell into the following categories:

- Misspelling of names of variables or methods, including in the case of the name
- Misspelling of types, e.g. in use of wrapper types versus primitive types.
- Incorrect access modifiers (e.g. public instead of private).
- Inappropriate use of the 'static' modifier

These errors may not prevent code from functioning, and the code may even pass unit tests. However, the presence of any of these errors indicates that the student has not followed the question specification.

The first of these issues is not very significant from a pedagogical point of view, but indicates a lack of care and attention to detail. It is worth noting also that if names are misspelled consistently, the student's code may work perfectly well, but it will not necessarily pass unit tests.

The use of wrapper types in place of primitive types in Java is an easy mistake to make, because they are generally interchangeable from the compiler's point of view. However, there is a pedagogical point to be made here because wrapper types and primitives are qualitatively different kinds of types and swapping of one for the other may indicate a lack of understanding of the distinction between them.

The last two points are interesting because both may indicate a student solving a problem in a way that circumvents question requirements and object oriented principles. This often arises because students have a difficulty in implementing the code correctly, and taking these approaches provides a way around the problems they've encountered.

2.4 Attitude towards tools

This section explores tutors reactions to using the Checkm250 marking tool and how they interacted with it.

There are several sub-themes within this category, including coming to terms with how tools may perform better than the marker, what the marking tool is telling us about quality of code and how to mark it, the effort required to use the tool and how the tool can be incorporated in the overall marking task.

2.4.1 Beaten by a tool?

Tutors commented on how they felt about the tool picking up errors they'd missed. Reactions included embarrassment at having missed an error, to acceptance that this was what tools were good for, as well as recognition that they'd seen similar errors in other tutors' work when they were undertaking monitoring duties.

> *I did find that there were errors when I ran it after I'd marked the first time it was there the errors I hadn't spotted, which was embarrassing, but there were one or two that I didn't spot so I did find more with them*

> *You know what you're looking for and you think you've seen it, whereas the computer will say 'whoops'. That's what computers are useful for as far as I'm concerned, you know, they pick you up now and again and say well you missed this.*

> *I would have liked to say to you that I'd spotted all those errors anyway, but I would be lying. It did make me spot things that I would almost certainly have missed, and that was good, that was an improvement.*

> *There were ones where I didn't get the set of results I expected and when I initially looked at the code I couldn't see why, but then when I looked a little closer you found there was an issue*

> *Because your mind sort of set up to think it's going to be okay, you don't notice them. And it's… you know, those sort of bugs in code are … if we could all spot them there wouldn't be any bugs in code.*

> *If you'd asked me previously do you ever miss things I'd have gone 'of course not'*

> *I do monitoring as well, and I see that other people do as well, but this made me*

more conscious [of things I'd missed]

For my own part, I have no doubt that every marker will miss many errors when marking. Does this matter? For some tutors it is important to pick up all issues, because commenting on them should help students perform better in future. For others, it may feel like they are being picky, or discouraging students if they mention all their mistakes.

2.4.2 Seeing the big picture

Sometimes a tutor may feel that it is more important for a student to get the 'big picture' correct.

> *I am not like a real stickler for semi-colons and stuff like that, but I want to make sure that they understand the gist of everything and if they kind of know what they're doing with their if loops and their for loops … and I would probably be more lenient about things like naming things correctly*

> *I would be letting them call it get-whatever and not exactly the name of the instance variable and that was one of the things I got caught with…. I'm more focussed on whether they've got the whole idea of it.*

> *I did like the specification checker, because it did pick up things that on first glance I had missed. One of those was somebody creating a private method when the requirement had asked for a public method, and … that's reasonably common to do that. I don't consider that to be a major issue really, but the specification checker picked that up.*

These comments suggest that tutors may regard marking code on paper as acceptable, rather than checking that the code actually works or meets the specification we set. The last comment provides one example of when code might pass a unit test, but not meet the structural specification requirements.

2.4.3 Is it worth the effort?

Is it worth it to find out when something is wrong, and you missed it?

> *[It is] actually addressing some issues that otherwise might not have been addressed. I thought the extra time that was spent there came back in TMA02 where the code you got was better.*

> *If we've got to comment on that I'd like to be accurate obviously.*

> *I quite like tools and I think that if you've got a tool for a job that works that saves a lot of hassle then use it.*

> *Some of the scripts it made me give the student feedback on things which otherwise I might not have done. Things that I might have glossed over, either not spotted or if I'd spotted them maybe just put a very brief comment, I probably put something more extensive.*

> *It is important to follow the specification. You know in industry if you start doing your own thing you aren't going to be popular. I think they do need to learn those kinds of things.*

These comments are supportive of marking tools picking up issues that we would like tutors to comment on. The first comment suggests that providing feedback on these issues may have resulted in improved performance by a student on the following TMA.

2.4.4 Helping rather than replacing tutors

Most tutors are comfortable with fitting the tool into their workflow, as the following comments indicate:

It gave me a bit of confidence I was looking for the right things.

It was a good test for me, of my marking as well, so I was quite pleased I picked up most of it.

It was nice to be able to do that test and have some kind of reassurance that you know at least those headers were right.

I suppose it gave you a feeling of safety.

I didn't know how much faith to put into the tool, and also I wanted to really test myself and see whether I spotted everything.

I like the confirmation that I didn't overlook something obvious.

2.5 Focus of marking

Tutors expressed their personal values, and what motivated them to tutor. The results were not surprising, but may serve to reinforce the idea that tutors wish to remain attached to their students, and not be replaced by tools.

That's rewarding too, if you can solve the problems for them.

I do quite often try and fix the student's code.

My biggest problem is I'm not teaching my own group anymore.

I'd rather that a student sent in bad code that showed complete lack of understanding, because then we can get in touch with them and do something about it, than where they just don't send anything in, then we have a problem where they then go quiet and evaporate.

The final point is interesting because there is a fear that when faced with a tool telling them their code is incorrect then students will not submit it.

Tutors have indicated that their students' code usually compiles, and an interviewee reported that it is more likely to find a feature or part of a question missing than partially implemented. Students may be making a choice to provide code that runs, when they could get extra marks for incomplete code, because non-running code appears to be (and in some sense is) less complete than running code. This has a knock-on effect in that the tutor may have been able to provide useful feedback on the incomplete or incorrect code, but the student will not get that feedback.

Ideally we would be able to separate the parts of the students' code that compile and run, and mark those automatically, while allowing tutors to provide feedback on incomplete parts, but for automated marking this is a complex task. Tutors are more able to cope with this kind of incompleteness.

It's not all about specification checking

There is also a concern that receiving 'tick marks' might lead students to believe they are fully correct, when a tool only covers part of an aspect of code quality.

> *It's not going to test that they have got a sensible layout for their code.*
>
> *It's not going to test whether they've got perfectly sensible arguments that just aren't course style, so they're not using substitution correctly, because it will be picked up if that's not right.*
>
> *[It's not testing] whether they're using code that could have been done in one line over several lines. It's not testing whether they're using if-else statements sensibly.*
>
> *It's not picking up whether they've commented everything successfully is it?*
>
> *It's not going to test for code reuse.*
>
> *Students can do an awful lot of inefficient and inelegant things … no sensible amount of tool use is going to pick up.*

These points raise other issues of code quality, which are also important, but they are supplemental to structural specification, and often less amenable to automation. For example, a style checking tool can pick up whether everything is commented, but not whether everything has a sensible comment. The latter really requires a human.

In some cases it felt to me as if tutors were not making a clear distinction in their own minds between these aspects of quality and what kinds of tools could handle them. The focus of this project was particularly on specification checking, but not to the exclusion of other kinds of tools.

Also, the intention is not to exclude other kinds of marking by humans, but to support marking by humans of a particular aspect of code quality that is amenable to automation.

Students often don't understand how creating methods relates to reuse, and marking reusability would be hard to automate in general, although there are code refactoring tools that can provide a programmer with hints about places in which code might be reused. It is not within the scope of current plans to attempt to address this aspect of code quality.

2.6 The need for tools

Ultimately, for some tutors, marking tools are not needed, because they consider the assignments are simple enough to mark and that they are good enough as markers to notice errors that students make. For me, this misses the point. The tool can assist tutors, but I do not intend for it to replace them.

For more taxing assignments, we assume that the value of a marking tool would be clearer, and that tutors would recognise that. However, this is tied up with the amount of time needed to use the tool, and the trade-off in accuracy achievable, and whether tutors consider that complete accuracy is needed or even desirable.

Tuning out and turning off

Concerns were expressed sometimes about relying on tools. Does it mean turning off, or a loss of attention? Some tutors are concerned that if you give students (and perhaps tutors) a tool that issues tick marks you will lead them to believe that all is well, when there are many other aspects of quality.

> *I've noticed that with the iCMAs if students think their code has passed the tests then they stop; they don't go on and think whether the tests are sufficient.*

> *I think you still need some assessment that doesn't have automated checking that the students can see so that you can see how much they've actually learned.*

> *It's awfully tempting to then go through the assignment and go 'tick, tick, tick, this is great,' so maybe there would be a downside to adding these kinds of features.*

> *If you just regard it as a tick-box that does exactly what it says on the tin and it has the right signatures for all the methods it's right, then there's a danger that you might miss all sorts of things.*

> *I am probably just as sort of fussy with it as I would be without it.*

Of course, the tool is only intended to help with one aspect of code quality. Although the intention is to provide support for marking other aspects of code quality eventually.

Sometimes tutors think that students will not learn to do it themselves if they've been used to having a tool to help them in the past.

> *You've got to be wary about passing students who actually couldn't do it without all the tools.*

> *It would make it very difficult to answer the questions if they went for certification because they wouldn't have them to rely on.*

> *At some stage I think there needs to be some sort of blind testing where they're not told particularly to see what they've actually learned.*

> *I think the drawback could be they come to rely on it, and then when the exam comes around.*

I am puzzled by this concern, because I think that a significant part of learning is through making mistakes and having them pointed out. When tutors mark, they point out mistakes. Why does this become problematic when it is a tool pointing out mistakes? Feedback is crucial to the learning process, and immediate feedback is preferable to delayed feedback, according to the literature [5].

Perhaps the concern is in part expressed by the following comment:

> *I've got a slight concern that you're trying to get rid of tutors!*

This is certainly not my intention, although I do believe that there is a place for greater use of automated marking within the MOOC environment, and that CodeRunner questions in the VLE will provide a good way to realise this.

And in some cases tutors may feel that what is okay for them to say is less okay for a tool to do:

> *It makes it sound a bit… a bit… sort of dictatorial and sort of 'you do it our way or else', you know, which isn't really desirable is it? This is what we say you shall do. It's one thing saying tutors would prefer you to do it this way…*

Tutors also realised that a specification tool would not be much use if the code was very far off the mark to begin with. This is true, but it is not within the scope of the tool to handle this extent of departure from requirements.

> *Their code was just gobbledegook really, and there was nothing that … no tool would have managed to handle it; it just didn't bear much relation to the question.*

This is where we need the human marker to take the lead and help set a student on a path to writing something more accurate.

> *I suppose there's a danger that if you do have tool support people think 'Oh well if it passes all the tools it's fine,' and that's not necessarily so.*

3 Conclusions

The survey and interviews revealed tutors' disposition for or against tool use, for themselves and for students. The tool was found to be useful by the majority of those who trialled it, and by all of the interviewees. Time available to use the tool is a major obstacle for some tutors, while others might use it if it could be more easily integrated in their workflow.

We know that although the CheckM250 tool should pick up all specification issues within its scope, it does not account for many other quality issues in code, e.g. functionality and style, including appropriate reuse of code, and many tutors commented on this limitation of the tool at some time or another during their interviews. This suggested that tutors were aware of other aspects of correctness they assessed students on. Support for automated assessment of those other aspects of correctness was solicited.

It is necessary to understand the separation of concerns of testing correctness at different levels. In particular, structural correctness is required to perform unit testing, and unit testing will not necessarily detect all aspects of correctness, depending on the coverage of the tests. It is quite possible that a piece of code could pass specification checks and fail unit tests, or pass unit tests and fail specification tests. Therefore these aspects of correctness testing support one another.

The CheckM250 tool is not intended to be a replacement for other kinds of testing or a licence to ignore other aspects of code quality when marking. Some of these aspects of code quality assessment are amenable to automation, while others are relatively complex to perform through tools, and are better suited to tutor feedback. Even when automated tests are possible, tutor feedback would be useful to add nuance and value to the test results.

4 References

[1] Online Surveys [Online] https://www.onlinesurveys.ac.uk/ Accessed 12th October, 2018

[2] "PMD an extensible cross-language static code analyzer." [Online]. Available: https://pmd.github.io/. [Accessed: 25-May-2018].

[3] "Checkstyle." [Online]. Available: http://checkstyle.sourceforge.net/. [Accessed: 25-May-2018].

[4] Tisi, J. et al. 2013. A review of literature on marking reliability research (report for Ofqual). *Nfer*. June (2013).

[5] Hattie, J. and Timperley, H. 2007. The power of feedback. *Review of Educational Research*. .77, 1 (2007), 16–7. DOI:https://doi.org/10.3102/003465430298487.

Appendix 1: Interview plan

2.1 Interview Plan

1. Skype for business meeting call. Backup recording via Skype.

2. **Introduction**:

a) Thank you for volunteering to be interviewed. Shall we enable video?
b) Who am I? SL at the OU, involved in production and presentation of M250.
c) **Context**: Esteem project on tool use on M250 and in particular on a tool to check students' code compared to a specification.
d) **Aims to explore:** How useful specification checking is 1) as a marking tool for tutors and 2) For students, to provide quick feedback on their responses. But the scope of the project has been broadening to consider tool use more generally in teaching Java at a distance.
e) **Purpose of interviews:** receive valuable feedback and input from tutors on M250, on pedagogical issues and how we support tutors and students with appropriate tools. I'd like to know what you spend your time on, and how we can make best use of your time.
f) **Time:** 30 minutes up to an hour.
g) **Details on confidentiality and anonymity:** I will be recording the interview and transcribing it, but will not disseminate any personal details. It is likely that I will want to use quotes from the interview, but they will be anonymized.

2.2 Questions

a) Before we go into more detail, may I ask how long you have been an OU tutor and how long you've worked on M250, and similar modules?
b) Can you tell me about your experience in teaching Java on M250 generally? What do you like or dislike about the module and supporting it as an AL?
    a. What motivated you to become an AL on M250?
c) Considering tool use on M250 in particular, we have BlueJ, the OUWorkspace, the MicroWorlds and many practical activities, as well as a variety of iCMA questions, including CodeRunner questions.
    a. Do you have any comments on these tools? How appropriate and useful are they?
d) Turning to the central idea of my Esteem project, the focus of the project is mainly on code specification. By this I mean students providing various features in a class that we have asked for in a question, such as instance variables or methods with particular signatures. But there are other aspects of the correctness of students' code. What particularly concerns you while you are teaching and marking Java on M250?
e) As you know, I made available a tool I called "CheckM250", for code specification checking. What was your reaction to the tool?
    a. Did you encounter any issues in using the CheckM250 software?
    b. Did you find checking of code specifications useful to you when marking?
    c. Did the tool change the focus of your marking?
    d. Did the tool give you a sense of which errors students most commonly made?
    e. If you would use the tool in future, how would you improve it?
    f. If you would not use the tool in future, why not?
        i. Would fixing that problem change your mind?

f) Do you think this kind of specification checking help would be useful to students when preparing their solutions?

    a. [Assuming they think it's useful to students] The feedback from the tool I provided to tutors is quite detailed. I've been thinking about the level of feedback that a tool like this can provide to students. For example, it could provide Boolean feedback (specification correct / incorrect), say how many errors there are (without identifying them), say how many method errors and instance variable errors there are, and so on, up to the point where it's being quite specific about what a student's code is missing such as methods with certain headers, for example. What sort of feedback do you think would be most useful to students?

g) What kind of feedback do you give to students about their code when you are marking?
    a. Do you try to fix students' code if it doesn't compile?
    b. Do you look at their code in BlueJ, or just mark the solution document?

h) I'd like to talk a bit about code style; for example, correct formatting and appropriate choice of variable names. How do you approach teaching students about style and marking their code in relation to style?
    a. Would automated style-checking of students' programming assignments be useful to you when marking? [this help is currently not available in the CheckM250 tool, but could be added]
    b. Do you think automated style-checking would be useful to students when preparing their solutions?

i) I'd like to end with a very general question: How can we make best use of your time as an AL on M250?
    a. Suppose we made more of our assessment through iCMAs rather than TMAs. How would that affect things?

4. Summing up / winding down

j) Summing up. Where is the work on CheckM250 going from here?
    a. Esteem conference coming up in April where I'll be summarising my experience so far with this and the feedback I've received on it.
    b. Continue to develop the specification checking tool according to feedback.
    c. Embedding in Practice for TMA quizzes, and in iCMA CodeRunner questions.
    d. Use on the Gateway to Coding in the same way
    e. Consider how tutors' role may alter in light of higher tool usage (if so)

k) Answers given in confidence.

l) Thank you for your participation!