Theme	Code	Explanation	Examples	Partial-fit Examples
Bugs in practice	Identifying bugs	Methods used by interviewees to identify bugs in code	• Read the code • Trace debugging • Print statements / Printf debugging • Staring at the code • Questioning invariants • Already knowing what the problem is • Debugging tooling for profiling • Hints by GHC • HLS integration in VS Code [...] red wiggles • Skim the area where the bug should come from • Write a test to replicate the problem • Hard coding some values • Dissect the area • Print the data and properties of data. • If tests have failed [...] then you know a place to start looking at • Follow up of guess • Look at the types involved • Figure out what parts of the program are involved • From that context (e.g. from the bug report), kind of figure out if it has to do with this part of the program • Replicate the bug	• It feels like there is a hurdle to really go into the weeds and use the complete debugging tooling because that requires some set up and that's kind of time intensive. And even if you use it [...] you need to actually be quite experienced to always find... to quickly find the problem in there. • I'm pattern matching here, and I expect this list to be non empty, and it's empty... That's a good point to start • I know, at this point, something is going wrong. But is it going wrong because of this function is doing something wrong? Or is it going wrong, because I receive wrong data? • I will just aggressive try to remove all laziness from the code and then see, if the system behaves correctly. Usually, if it doesn't, that means that I have maybe some infinite list that I am generating, but then the behaviour from the system changes from "my program is slowly leaking memory" to "my program is not running at all", in this case, because some infinite computations. I think that actually points you to somewhere where you need to be more careful.
Bugs in practice	Fixing bugs	Methods used by the interviewees to fix bugs in code	• Throw blank patterns at some places and hope it sticks • Investigate and try around • Rewrite the logic • Understanding the program state • Understanding the bug: ○ Git blame ○ Reading old commit messages ○ Understanding intention of code • Chesterton's fence – Question why something [piece of code] is there • Correct the incorrect types • Fully understand the problem and what the correct value would be in that location • Trial and error	• What you thought would be the right value isn't the right value and because the end result isn't correct • The way I coded at that location to introduce that bug might have cause me to introduce bugs at other locations because I programmed there in the same kind of way • So try not to be defensive and do something on this small part of the code • If it is an easy bug, I would probably fix it and then write the test instead

			• Moving checks to the type checker • Parse all SQL queries that are embedded in the Haskell code […] so that I know it’s valid SQL syntax and prevent that kind of bug (Typos in Haskell automated tools that generates SQL queries). • Try to fix the real problem, but because in Haskell we have the strong type system, sometimes I also look at ways in which this can be avoided or directly reported by the compiler. • Very aggressively adding bans or somehow forcing the evaluation of things and hope that this will fix the problem. • Trace statements or Print statements to see what the values involved are • Write a test • Strict Haskell, and there's strict versions of containers that you can use • Profile the application and look at the memory allocation	
Bugs in practice	Complexity of bugs	How hard different types of bugs are to fix as considered by the interviewees	• Timing / Optimization are probably the bane • Wrong understanding of the problem [...] this is kind of a nice bug to have • (Configuration type) it’s quite an annoying one [...] it’s so hard to debug because you cannot just do printf debugging or introspect the state of the pipeline • External systems […] sometimes it's actually hard to replicate why they have failed • These (bugs related to laziness in Haskell) are also quite problematic and usually are very hard to debug in Haskell because, usually, if you try to inspect what's going on, you're actually changing their laziness behaviour. • Most tricky (bugs) are the ones which are memory leaks related to laziness. • That's what makes it tricky in Haskell, because you can't write in the type system, or in the test, you can't say this function shouldn't take more than this much memory, so because we can't model that in the language itself, then you have to kind of use these external tools, which makes those kinds of	• It's (Memory leaks) basically the downside, the trade off that you have from using Haskell because it's really hard to reason about. • The really annoying bugs are the ones where you're like “Why does it not work? It should work! I don't know what's going on!” • It's good to check all of them because you never know where else the same kind of problems could have caused bigger issues in the code base. • So if we're in the lucky case in which we have some tests that has failed... this is maybe one case scenario, right? Because then, you know, a place to start looking at. • (Program segfaulting), was a very tricky bug, not just because the solution was weird, but also because other people have been working on it before and we weren't really understanding

			bugs (Performance) trickier than maybe they should be.	
Bugs in practice	Most time spent on bugs	How much time the interviewees spent on fixing certain types of bugs	• Requirement and environment bugs [...] sometimes it takes a lot • In terms of spent time per bug, performance and space leak issues are probably the most intensive • Understanding the requirements is 80% of the work • I spend most time on [...] wrong understanding of the problem • I've spent a lot of time [...] (when) you interact with external systems • I've spent a lot of time [...] on everything (bugs) which has to do with the fact that Haskell is lazy • I always think of them (bugs related to laziness in Haskell) as a real sink of time. • That's the bugs (Performance) I've spent most time on in Haskell	• If it takes two minutes, then it's not a complicated bug. But if it's taking a half an hour, or an hour, my evaluation of the complexity is building up.
Bugs in practice	Frequency of bugs	How often certain types of bugs appear	• Most bugs I fix are some kind of logic bugs • Most common bugs are wrong assumptions about environment • Space leaks [...] the king of Haskell bugs • Name shadowing [...] not very often • API bugs does not happen that often, because it's hopefully well documented • Rare kind of bug , or not very rare [...] type error • Most common is where I have some wrong understanding of the problem • Often have it in CI pipelines where something is misconfigured • Most of the bugs [...] involve some kind of external interaction • Most common thing I encounter is just off by one errors or I think actually for me, it's more performance bugs that I have to fix	• Bugs which have reached my testing or production [...] that's not very common • A lot bugs are prevented by type checker • IO runtime errors which can just bubble up • Haskell code definition of the (SQL) query [...] it's much easier to make just a typo in and will only figure out at runtime when a particular query is executed. • Usually bugs don't come all at the same time. They come bit by bit • I see often that you get wrong results, not because of that specific function, but because you actually got some wrong data coming in. • often I find that the kind of functions we write in Haskell are simple enough that if the data is consistent, they will give you the right results. • They (bugs related to laziness in Haskell) are not happening as often • Because most of the other things (other than Performance bugs) are taken care of by the type system and the tests, but what you really need to,

				kind of manually do is make sure that the evaluation is correct.
Bugs in practice	Bug types	Types of bugs mentioned by the interviewees or used by them in practice	• Performance • Memory / Space leaks • (real) Logic bugs • Type errors [not caught by compiler] (Runtime bugs) • Parsing bugs [bugs happening when parsing data to get it in the application] • Regression • Name shadowing (accidentally used a variable from higher up and not the current state) • Requirement and environment bugs [wrong assumptions about the environment the code is running in] • Expectations of the behaviour of an API • Type Tetris [someone picked a function which seemed to fit the complicated types they needed, but it does something completely wrong) • IO runtime errors • Compile time errors • Wrong understanding of the problem • Problems on the interface • Security relevant • External interaction • Validation issues • Tricky bug, where it takes you a long time to kind of figure out why it's being caused • Off by one errors	• Wrong variable • Wrong type • Forgot one edge case • Not working how you expect it • Wrong thing you implemented • Missing stuff • String conversion that didn’t work • Something out of scope of the Haskell type system • Interactions with the other programs other file formats other interfaces and that's where it's the most easy to make mistakes • Haskell type safety erodes at the edges
Bugs in practice	Bug classifications	Different ways of classifying a bug explained by the interviewees	• Implicit classification • By importance [not a problematic bug, or very rare, then we decide we can’t prioritize this] • Small / Serious bugs • No classification system • Safety critical • Label which area of the program it concerns [...] Backend [...] Cloud functions feature [...] server [...] GUI [...] sometimes we combine security and performance with like GUI [...] Infrastructure (new	• Treat bugs mostly with the same diligence and priority • It's very flexible just combining tags of kind of bug and which area. • We have priority, and we will say, this is a refactoring, or this is a feature,

			servers, more CPU) [...] Dashboard [...] SQL [...] User page • Easy vs tricky bugs (based on time spent)	
Bugs in practice	Pushing buggy code	Different behaviours of what interviewees do with buggy code before pushing their code to a remote branch	• Yeah […] that's just a business trade off • We don't have time to work on this, so we just leave it (the bug in the code) • Leave a TODO • Stash it • Don't push buggy code • Remove / Comment out the buggy part of the code • Build something simpler that is correct as far as the features it supports are • Push it onto a work in progress branch. • Wouldn't merge it to main • Push and say that I know this doesn't work	• Wait until a customer complains • I use GitHub also as a way to just sync the code between machines
Bugs in practice	Wild bugs	Examples of bugs the interviewees encountered and potential fixes	• There were some design decisions involved and I simply didn't realise that there is a situation where the user could actually even click on something which would display an invoice for another year. So, in that case, we decided to rewrite the logic for how to discover which invoices to cache and that's what we do now • The strong structure and the types prevent a lot of bugs, but when you get the data in, you need to somehow process this. Often that's parsing. Maybe it's with an existing library, if it's JSON or you write your own parser and the stuff you get in that's inherently noisy and you can just have a bug in your parser. And those are the situations where like validation of incoming data fails. • Sometimes you want to parse a document, but you use it as a maybe document and then the parser tries to parse a maybe document, but for that instance does not actually fit what comes out of the database • I used show somewhere in the code to convert something to a string [...] later, I changed the data type, but because show converts any data type to a	

			string, it still converted to a string and the new string was partially correct for some of the cases, but it didn't work for the other cases and it took me forever to realize that at that place like the show now created another string which was only correct for some use cases, which made a really weird error you get, because most of the time it works and it suddenly doesn't. […] By now if I use the show function, I always use […] “show space at text” for example or like “at user” […] so you can annotate which type the show function should receive. Then if you later change the type of the variable you get an error because the “at user” wouldn't work anymore if the new type is now “at person” • I learned there's two tools which basically do almost the same thing, but a little differently and I was using the wrong one. And then the pipeline crashed at that moment with a very inconclusive error message • If we had a list and we expected it to be non empty, maybe I can change the type from the regular list to some type of non empty list if I think that this is an invariant, which happens throughout the whole program, and then this will help me locate the places where I'm actually working with this value • In [project] there's an unsafe perform IO for performance reasons. And what was happening is that sometimes the program would segfault […]. Then I was tracking down the segfault, and then I was kind of trying to print out values. And what happened was that once I printed out a value, it would not segfault anymore. And then I think the solution to that bug was just fmap-ing ID or like... so kind of not doing anything but I think that causes the evaluation of the value, so it would not segfault anymore.	
Taxonomies	Frequency of bugs	How often do the interviewees consider each of the types to appear	• Algorithm / Method can of course happen • Assignment / Initialisation seems unlikely in Haskell • Checking is something that can go very wrong in Haskell	• Some categories, they're largely reduced in Haskell • Assignment/Initialization and that seems a little bit more low level

			• Data seems a little less likely • Internal Interface can happen • Configuration is annoying and the ones you encounter often • Network I try to avoid as far as possible • Database-related also common issue • GUI related issues also happen often, but they are mostly not critical • Rarely had Performance issues with Haskell • [Permission / Deprecation] that's something you can plan for […] so not really important • Every other type of issue might become a Security issue • (Program Anomaly) is always something you try to avoid as much as possible […] this happens quite often • Most of them are Logic bugs • Many of the bugs although not the ones that take the most time to fix are Program Anomaly issues • Another important source of bugs for me is what it appears to be network issues • Configuration issues... I don't think I've dealt with too much • Database related issues […] I haven't found that often that people will have problems writing SQL […] And databases, in my experience are not things you fail for a connection […] it's something that you control • GUI related... I wouldn't often do UI application in Haskell • Performance issues, as I mentioned, I think they are very... they are rare in Haskell • I've never had to deal with a security issue • May not apply too much to Haskell is scanning of Data • Type conversions is very much non existent in Haskell, you always go through some kind of functions. • There is not often that you can define it (modelling of data) wrong	• If buttons don't work anymore, buttons are hidden, that is more severe • Anything stylistic, I don't even count as a bug • But all this laziness, or this potential to do infinite computation in Haskell, really means that you could have for performance issues which are difficult to track • You can see all of these in practice, but I think it really depends on where you are in the stack.

			• It (non functional defects type) feels like too broad of a category to me • Which of these appears most [...] definitely Performance bugs • I usually work in the back end, so GUI issues are not really my forte or permission issues. That's more like something that the front end would have to deal with • I've had database related issues for sure • Configuration issues... it's very much part of the initial setup, but we don't have it so much during development. • Optimisation appears most • Checking appears a lot less because the type system really forces you to handle everything that can happen • But I think once the types are in place, the types make sure that you're handling data better so the defect (Data) appears, but at a different stage of the project.	
Taxonomies	Suggestions	Suggestions given to improve the taxonomies, e.g. adding types.	• Misunderstanding requirements • Split that one up into more categories, because GUI related doesn't tell you anything, just that it's happening in the GUI. But is it stylistic? Or is it impacting the user flow? • (Program Anomaly) needs to be split up […] because what kind of logic error is very important • When I think of the (Database-related) problems, I think of them, and if you have a problem with the connection, that's network issue, because that's a server you are not being able to talk to. So in my mind, it should be dealt with as any other external communication you do. • And a bug in your SQL statement, well, that's an issue in your logic • Validation issues are a big source of problems […] that would be a different kind of problem from, what I think here Program Anomaly would be covered by this. • separate those between validation and proper program logic. So if you make a mistake in the actual	• (Logic bugs) sounds very broad • I feel the categorization non-functional defects and functional defects, the opposite… sounds quite important as well […] but here it's somehow on the same level • (Taxonomy from Seaman et al.) seems to be very code-specific • I fail to see the difference between assignment / initialization and checking for potential errors • Logic and Algorithm / Method sounds to me like a strange separation • I feel like I somehow like more the separation of non functional defects from the previous taxonomy (Catolino et al.) in several kinds of issues like permissions and performance • That can be difficult when you have a very strict model and very specific model of something, it can be difficult to make something that fits into that model. But once you do that, you avoid a lot of the other issues. I think that's a trade off there.

			logic seems to me different than failing to validate something that was coming from some external source. • I don't see it (Data type) as big as to have their own part in taxonomy. • Separate the performance part where it would include kind of memory leaks and things like this in Haskell, • Modelling issues is a big problem […]we're having a problem with figuring out the right type for some operation or function, so that it matches what we expect to happen. […] that means both writing the model and also implementing something that fits the model.	• I think this (taxonomy) is more relevant at the later stage of a project […]these are more for when you're when you're taking it from the prototype to an actual production system • So talking about type conversions, and then if you have a strict type system that doesn't really make sense. Or it has a very different meaning. • Because sometimes it's a blend... like, in the first one you have configuration issues and then you have network issues. But most network issues are kind of configuration issues of the network stack or something. Then you would have to decide which one should I assign.
Taxonomies	Usefulness	How useful the interviewees perceive the taxonomies	• So a few categorizations (from the first taxonomy) definitely help but like this doesn't help me at all • Find out that most of your bugs maybe belong to one or the other category here and you might then come up with some steps to avoid them in the future. • I think the first one (Catolino et al.) is more useful than the second one (Seaman et al.) • I don't see too much usefulness in programming… in research yes, but not in programming in day-to-day coding. • A great way to document things • Could be useful as a way to figure out where you as a programmer or your team need to learn more • If you all the time see problems with a particular piece of your software because you see this particular types of issues, then maybe you should spend more time on this and I like that this could be potentially guided by bug classification instead of just being a gut feeling which is mostly how these things are done now • I think if you have a large project with lots of different teams, then this could be good for kind of triage, when you're like assigning different people to different bugs • External, internal interface, can be very useful because usually there was someone that	• Maybe you see a lot of your bugs are external interface, then you should be more diligent with your tests for interfacing with the external systems. • Whatever taxonomy you use or whatever you analyse your bugs for, you should first ask yourself what do you want to do with it afterwards. What's the purpose of organizing or classifying it? Because otherwise I think there's just too many options or ways you could classify it, to be of any help • we sometimes lack this vocabulary when talking to colleagues, to say "Oh, I have a problem." But what kind of problem? It's a bug, right? But the bug is a very broad thing. • These are very Java-like programs, for these taxonomies • They don't mention all these other things that you have in, like a strictly type language, like Haskell

			implemented that interface in your system, so you can kind of point towards the right person • I think if you're a solo developer, or you're in a small team, then I'm not sure that they help. I'm not sure that they help enough that it's worth labelling every issue as one of these • It could be useful, but could just suck away time from the actual fixing. • It could be useful, especially if you're using some stack... if you chose a specific database or a specific GUI framework […]we should reconsider our choices or hire more people for that. • It really depends on the size of the project and the time scale of the project	