manifest the behaviour

Your response on ANY symptoms you can think of which
behaviour. These are not intended to be comprehensive; base
some cards give examples to help clarify the described

Example:

names

e.g. Uses camel case for variable

A description of developer behaviour:
Declares a local object right before its first usage.

Example:

Gives each variable the smallest possible lifetime and scope.
Example:

```
write functions which retain state
between calls
```
Seems to write a lot of lines of code to achieve a given outcome.
the device drivers

* Resident expert who always wants to be the one to write
* Does not like others to work on code they wrote

Example:

Tends to "own" code
Uses a complex conditional expression with multiple boolean variables rather than nested conditions or intermediate operators, rather than nested conditions or intermediate operators.

Example:

Uses long multi-part conditions
their code

is willing to discuss suggestions about

Example:

- Approach makes a change or explains a clear rationale for the existing approach
- Debates pros and cons
- Asks for more information about suggested approach

their code

is willing to discuss suggestions about
Final String exitMessage = "Goodbye!"

const float PI = 3.1415926535;

Example:

Relevant

Makes data immutable whenever
other languages and uses its idioms when coding. Pretsers prefers <a href="">a programming language</a>
etc
- Generating documentation
- Building code
- Running of tests

Example: 

Automates tasks
An empty line between two sections

Alienment of related structures

Example:

align and separate code

Uses horizontal and vertical spacing to
bug reports
and how to reproduce the bug in their
includes accurate details of symptoms
Avoids deeply nested code
Uses break judiciously
Avoids GOTO

Example:

Keeps the flow of control easy to follow
Data structures and the processes that operate on them are

Example:

follows encapsulation principles
which will compile and run

Defines APIs and implements some minimum functionality

Example:

- can use
- functionality that other team members
- tries to provide early outline
Example:

Checks return value when opening a file for write calls which may return error codes from function return values.
Follows formal methods to the letter
keyboard and just fixing it
Explains how to fix a problem rather than taking the
patient with less experienced developers
Willing to answer questions

Example:

Is good at helping others
Is rigorous about deallocating allocated resources

* malloc - free
* file open - file close
* new - delete
etc
Example:

Let's existing code constrain future code
Ignores build warnings
suitably informative message

Accompanies each commit with a
- Finds it easier to implement than check if required
- Thinks it might be needed in future

```
Example
```

Currently needed
includes code features that are not
Use of the STL in C++
* Ternary operators in Java
* List comprehensions in Python
* Lambda expressions in functional languages

Example:

Uses the idioms of the programming language
Writes long if... else if... else if... blocks
Example:

\[
\text{Fahrenheit} = (\text{Celsius} \times 9.0 / 5.0) + 32.0;
\]

precedence (BODMAS)
denoted by mathematical operators
includes brackets which are not
Quick and dirty change to get customer's system working

Example:

Change
when knowingly making a sub-optimal
logs it in the issue-tracking system
more parameters
Commonly writes methods with 6 or
Catches exceptions at a level of the code where they cannot be resolved.
display a "not implemented" message

* improves a variable name
* splits up an over-long function

Example:

then when they checked it out

Tries to leave a module a bit better
Assuming no unexpected events, the program could still run correctly if all exception handlers were removed.

Example:

Rarely uses exceptions as part of a program's normal flow.
Is often the person who breaks the build

Example:

* Missing files
* Missing steps
* Broken code
* etc
Willingly to give priority to helping fix a problem which is holding others back but does not affect their own current task.

Example:

Put project goals over individual goals.
Prioritises performance in the design of their code

Example:

Code is optimised for speed over readability without any benefit to overall system performance
I'm getting an exception. Do you know what the problem is?

Example:

Asks questions without giving context.
Perform a single task.

Writes short, simple functions which
Rather than:

```plaintext
a = 3 / (b = c + 1) % d;
```

Example:

Uses assignments within expressions

```plaintext
a = 3 / b % d;
```
Does not assume that a complex problem necessarily results in complex code
Fixes the symptoms without discovering the root cause of a bug.
Finds out whether functionality is already available before writing their own implementation

*Example:*

* Uses project libraries
* Uses language libraries
* Uses frameworks
Quantity of messages is sufficient but not verbose.

Content of messages is informative and succinct.

Errors, warnings, and info are clearly distinguished.

Example:

| Their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages | their code includes useful logging messages |

their code includes useful logging messages
Example:

When changing code, doesn't always update or add tests.
things

Espousés "one true way" of doing
if ((day == 31) && (month == 12) && (year == 1999))

result = operand > (a + b);

Example:

precedence demanded by the language's operator
includes brackets which are not
Assumes that things which "can't happen" won't
* Does not leave TODOs for others to deal with
* Removes temporary debug statements

Example:

Practices good housekeeping
assumptions about the correct interpretation
* ASKS for clarification of a requirement rather than making
  * ASKS for explanation of domain-specific concepts
  * ASKS for advice on programming issues
  * ASKS whether there is existing code to do X

Example:

Is willing to ask questions
Rather than domain terms

Example:

```python
if (user.isauthorized())
    # code
```

Codes using implementation terms

Rather than domain terms
common use of functionality
* Contributions code to project libraries when they notice
* Avoids "copy-and-paste" coding

Example:

You yourself principle
Follows the DRY (Don't Repeat...
Example:

Tends to work in isolation

* Raresly integrates their work with others

* Checks in their work inequently
convenience of the underlying implementation

* Designs APIs which seem natural and obvious, not for the
  Documents the API

Example:

Makes APIs easy to use correctly
Regardless of context, tends to apply a favourite pattern.
* Comments only what the code cannot say
* Does not obscure the structure of the code
* Updates existing comments to match changes to the code
  * Explains the task done by the following section of code,
  * Assumptions, alternatives discarded, trade-offs, workarounds
  * Documents design decisions, e.g.
  * Explains the domain logic of the code

Example:

understanding
Uses code comments in ways that aid
Nouns for function names: conversion() vs conv() * 
Verbose: performance() vs performance() * 
Superfluous: temperature vs temperature * 
**Cute**: int avoidkeckedavra; // exit code when system dies * 
**Misleading**: getdata() actually writes data to disk * 
**Misleading**: String s; float ex; int foo; dostuff(); * 
Silent differences: old$pwd & old$pwd; userinput & userinput * 
Spelling mistakes: String passsword! * 

Example:

succinct, meaningful and distinct
Chooses identifiers which are not