ozcode

OZCODE SUCCEEDS WHERE OBSERVABILITY TOOLS FALL SHORT

Ozcode Live Debugger enables root cause analysis and resolution of age-old bugs that an error monitoring tool could not resolve

Evert Wiesenekker
Owner, CodingDutch Application Development

CodingDutch Application Development provides full-stack web development for both government institutions and the private sector. The company goes to great lengths to keep its projects running smoothly and has deployed legacy systems that have been running for over 10 years.

Trying to debug Production with no data leaves unsolvable errors for years

One of the biggest challenges the company faces when debugging Production errors is that it does not have access to the live systems on which its applications run. The customer may report an issue, but provides no data on the scenario to reproduce the issue, how frequently it occurs, the number of users experiencing it etc. In many cases, CodingDutch had to investigate issues without being able to install anything in Production or attach remote debugging tools and with no access to end users as is typical in most environments. Moreover, some customers, such as government agencies, must adhere to strict privacy regulations and cannot share the context and details of the bug, so CodingDutch was not even able to view log files. Again, trying to debug a Production error with no access to data. Consequently, some of these systems had **"known issues" which remained unsolvable for years.**

CodingDutch
Application Development

Without Ozcode

- No access to Production data
- Countless old and unresolved errors
- Error monitoring tool not helping

With Ozcode

- Code-level observability into Production data
- F10 F11 Time-travel debugging
- Age-old bugs fixed within hours

When Production systems mishandle unexpected user input

One of CodingDutch's customers is the Dutch Ministry of Agriculture, Nature, and Food Quality, which oversees the import of agricultural products in the Netherlands. One of its functions is to screen imported food products for pests. The Ministry maintains a computerized taxonomy (naming) system to classify all the different pests it monitors, and CodingDutch was managing and maintaining the taxonomy system software. Since there are different ways to name and classify pests, the taxonomy had to be flexible and allow for multiple synonyms. CodingDutch accommodated this requirement by using a NoSQL system with a flexible indexing mechanism (RavenDB).

Companies wishing to import agricultural products to the Netherlands must provide a digital manifest of all the pests which may infect the products using the Ministry's taxonomy system. Now, different language uses diacritics, special markings above and below the Latin characters that change how they sound (e.g., ã, ë, á). To accommodate diacritics, CodingDutch was using a regular expression and mapping

mechanism to map characters with diacritics to characters accepted in the Ministry's taxonomy system. The following code snippet was used to parse expressions entered by the companies:

```
...
Regex regex = new Regex("^|'|ã|ñ|ø|♀|♂|-|è|é|ë|á|ä|ü|ö|ń|ł|ï");
    MatchEvaluator evaluator = (Match match) => match.Value switch
    {
        "^" => "-", "'" => "-", "ã" => "a", "ñ" => "n", "ø" => "o",
        "♀" => "female", "♂" => "male", "-" => "-", "è" => "e", "é" => "e",
        "ë" => "e", "á" => "a", "ä" => "a", "ü" => "u", "ö" => "o", "ń" => "n",
        "ł" => "l", "ï" => "i",
        _ => throw new Exception("Unexpected match!"),
    };
...
```

When a company wanted to import goods and entered the term *"Carcharolaimus banaticus Krnjai**ć** & Loof, 1975."* , the Ministry's taxonomy system failed, throwing an exception, and would not accept the input. Apparently, the **'ć'** had been omitted from the mapping snippet. Consequently, manifests were not filled out correctly disrupting the process of importing products to the Netherlands.

Error monitoring tools did not provide enough data to solve the problem

To acquire Production data when errors occur, CodingDutch had installed an industry-standard error monitoring tool to monitor the Ministry of Agriculture's taxonomy system. The monitoring tool did provide notifications when exceptions were thrown, but CodingDutch encountered problems working with them. Configuring the tool and getting it to work correctly with the Production systems was very complex. Also, the notifications it provided were not nearly adequate to understand what caused an exception, and certainly not detailed enough to provide insights on how to resolve issues. It was impossible to know ahead of time what data the error monitoring tool should report in its exceptions, and CodingDutch had no way to see that the **'ć'** character was problematic.

The error monitoring tool did not help identify the issue with entering data in the Ministry of Agriculture's taxonomy system.

A parallel system exposes 10-year-old errors

In preparation for a major upgrade of the Ministry's taxonomy system software, CodingDutch installed a parallel system developed using more modern technologies such as Bootstrap for the front end and Blazor for the server. An Ozcode agent was installed on the same webserver running both the old and new versions. The agent effectively monitored both versions.

Ozcode Live Debugger started registering exceptions thrown by both the old and new version of the taxonomy software. **Within one day, errors that had been a mystery for years rose to the surface and were fixed within hours.** In particular, Ozcode Live Debugger brought to light the issue with text that has diacritics being entered into the taxonomy system causing it to fail. Using time-travel debugging to step through an exception, CodingDutch were able to examine the invalid variable that included the unhandled 'ć' in the snippet used to parse user input. This level of detail was never provided by the error monitoring tool and the error would have remained unsolvable without Ozcode. Moreover, Ozcode was surfacing other errors that had been occurring in the taxonomy system for years. Now, with access to Production data these errors could finally be resolved.

ozcode

The error monitoring tool did not provide enough data for debugging and required a lot of complicated setup. I installed Ozcode within 15 minutes and was immediately blown away by the level of detail it provides.

Evert Wiesenekker, Owner, CodingDutch Application Development

Time-travel debugging with code-level observability helps keep imported produce free of pests for Dutch consumers

The legacy software behind the Dutch Ministry of Agriculture's taxonomy system had errors. Since developers had no access to Production data, these errors remained unresolved for years. CodingDutch introduced Ozcode Production Debugger, which works with both legacy C# systems as well as modern technologies like .NET Core and Blazor. Ozcode caught the exceptions in the taxonomy system and provided the production data required to resolve them. Being able to step through the error execution flow of exceptions with full visibility into the call stack, locals, method parameters and return values, network requests, database queries, and relevant log entries gave CodingDutch Application Development the insights needed for a root cause analysis. CodingDutch fixed errors that had gone unresolved for years and identified new errors that had previously gone undetected.

Without Ozcode

- No Production data
- Unresolved errors
- Error monitoring not helping

With Ozcode

- Code-level observability into Production
- F10 F11 Time-travel debugging
- Age-old bugs fixed