# Concurrency at Microsoft – An Exploratory Survey

Patrice Godefroid
Microsoft Research
Redmond, WA 98052
pg@microsoft.com

Nachiappan Nagappan
Microsoft Research
Redmond, WA 98052
nachin@microsoft.com

## ABSTRACT

Concurrent programming is gaining significant prominence in the software industry, especially due to the advent of multi-core architectures. In this report, we present the results of a survey deployed inside Microsoft in January 2007 to assess the state of the practice of concurrency at Microsoft. Our survey polled 10% of the Microsoft technical staff and collected data for each of the three major business units, namely Microsoft platforms and services division, mobile and embedded devices division and Microsoft business division. Our major findings indicate that the use of concurrency is widespread at Microsoft. Of our 684 respondents, over 60% of our respondent population had to deal with concurrency issues frequently (on a monthly basis). The most popular platforms for concurrent programming inside Microsoft are Win32 and CLR (Common Language Runtime), which are equally popular. Also, multi-threading and message-passing forms of concurrency appear to be equally pervasive. Concurrency bugs take on average several days to detect, reproduce, debug and fix. Most of these bugs are of high severity. Most engineers feel concurrency issues will be more of an issue going forward, and would welcome additional help in terms of language support, libraries, tools, processes and training.

## 1. INTRODUCTION

Many research papers are published each year on analyzing concurrent software systems. Yet, most researchers have little idea of how widespread concurrent programming actually is in the software industry, and in what specific forms concurrency is used. Without a better knowledge of the typical end customer for concurrency analysis tools (microview) and of the total addressable market for such tools (macroview), any widespread technology transfer for such tools is doomed to fail.

There are few external studies that discuss concurrency related issues presenting a company-wide perspective. This paper attempts to provide a high-level view of the use and practice regarding concurrency in a large company like Microsoft. A primary motivation is to understand better the types of concurrent programming currently being used. For instance, what are the most popular platforms for concurrent programming inside Microsoft? Are most forms of communication based on message-passing or multi-threading? This information may help target the scope of new tools to detect concurrency bugs. Another motivation is to understand the current processes and tools used to detect, debug and fix concurrency bugs. During what phase(s) of software development are concurrency bugs detected? Finally, we wanted to get a consensus from the engineers on how hard they felt concurrency bugs were to fix, their severity, and collect general suggestions from respondents on how to better deal with concurrency.

To address those broad questions, we performed a survey inside Microsoft in January 2007. The purpose of this short paper is to share with the external research community some of the key (non-proprietary) insights we learned by analyzing the results of this survey.

The rest of this report is organized as follows. Section 2 briefly presents some information about the organization of the survey we conducted. Section 3 discusses various forms of concurrency used in Microsoft products, while Section 4 discusses concurrency bugs. Section 5 concludes with some general comments on future directions.

## 2. DESCRIPTION OF THE SURVEY

In order to assess the current state of the art/practice of concurrency at Microsoft, we conducted an on-line survey. For this purpose, 10% of all employees belonging to three primary activities (development, test and program management) were selected randomly from the employee database of Microsoft. This set includes managers, leads, architects, etc. We deployed the survey for a period of 2 weeks. Overall, 684 people responded to the survey. Before interpreting the survey results, we discuss whether our respondent population is sufficiently experienced. For this purpose, we assessed their work experience at Microsoft (to understand their familiarity with the Microsoft environment and culture) and their total work experience. Table 1 presents this data, from which we see that our sample population comprised people with a significant level of work experience. The self-reported

average size of the respondent's team was around 100 employees.

| | Min. | Max. | Mean | Std. Dev. |
|---|---|---|---|---|
| Years of Experience at Microsoft | 0 | 23.67 | 7.33 | 4.16 |
| Total Years of Experience | 0 | 48 | 10.48 | 7.81 |

**Table 1: Experience of respondent population**

# 3. CONCURRENCY AT MICROSOFT

In this section, we discuss, at a high level, the various forms of concurrency used in Microsoft products. Figure 2 shows responses to the question "what form of concurrency do you deal with in your product?" We observe that both multi-threading and message passing are almost equally pervasive in Microsoft. (We expected multi-threading to strongly dominate message passing, but this is clearly not the case.) Therefore, tools for concurrent program analysis should ideally support both multi-threading *and* message passing. The other most commonly type of concurrency reported was "database concurrency".

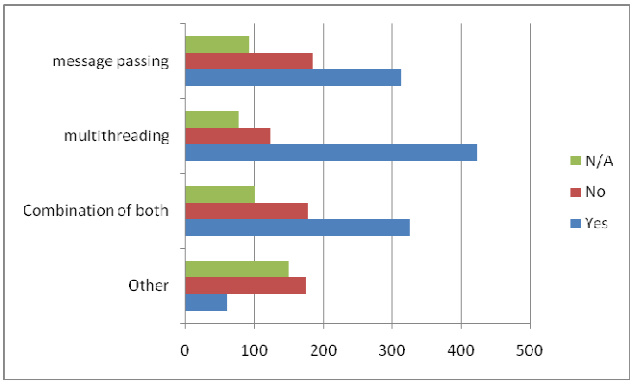


**Figure 2: Different forms of concurrency in Microsoft products**

Table 2 below presents the reasons why concurrency is used in Microsoft products. We observe that performance and responsiveness are the primary drivers. Reliability, robustness, supporting multiple clients, scalability, and security are the most common other reasons mentioned by the respondents.

| Performance | Responsiveness | Modularity | Other |
|---|---|---|---|
| 56.8% | 51.2% | 27.4% | 8.8% |

**Table 2: Reasons for concurrency in Microsoft products**

Next, we wanted to assess what concurrency platforms figure predominantly in Microsoft. Based on previous focus interviews with members of Windows, Visual Studio and Windows Live Core, we gave respondents the options of "Win32, CLR or other". Their responses are shown in Table 3.

| | Total |
|---|---|
| Win32 | 47.1% |
| CLR | 42.5% |
| Other | 10.4% |

**Table 3: Concurrency platforms in use at Microsoft**

We observe that Win32 is slightly more popular than CLR, but not by much. (We expected Win32 to dominate CLR more.) Note that 22.1% respondents replied "both Win32 and CLR", so the overlap is significant.

# 4. CONCURRENCY BUGS

An interesting observation from this survey is that about 66% of our respondents deal with concurrency issues in one form or another. This looks like a sizable part of the Microsoft population, rather than a small and confined group of concurrency experts, as we had expected. As a follow-up, we asked respondents to select how frequently they detect/debug and fix concurrency bugs. Most respondents who face concurrency issues deal with them on a monthly basis (Table 4).

| | |
|---|---|
| Daily | 2.8% |
| Weekly | 13.3% |
| Monthly | 38.2% |
| Rarely | 43.7% |
| Never | 2 % |

**Table 4: Frequency of facing (detect/debug/fix) concurrency bugs**

This leads us to a discussion on how concurrency bugs are currently detected. Figure 4 shows how effective various phases of the development cycles are in detecting concurrency bugs, according to our respondents. Most respondents said that they find the majority of concurrency bugs during system/integration, performance and ad-hoc testing. Code review also scores relatively well, while code scanning tools do not. (This is a known area for improvement for code scanning tools as there are currently few industrial-strength practically-usable static analysis tools targeted at detecting concurrency bugs).

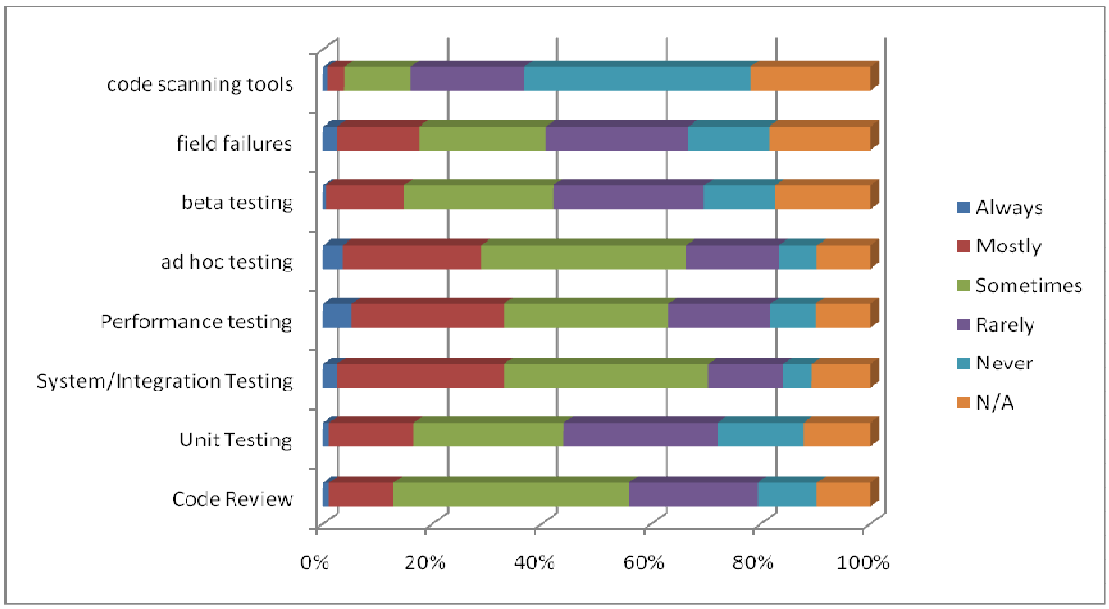


**Figure 4: Perceived effectiveness of various techniques for concurrency bug detection**

Note that the perception does not change if we compare the developer population and the non-developer population. (One might have thought that developers would rank code scanning tools or unit testing as more effective ways to detect concurrency bugs; that is not the case).

When asked about how easy or difficult concurrency bugs were to reproduce, there seems to be a consensus that most concurrency bugs are hard to reproduce, as shown by the data in Table 5.

| | |
|---|---|
| Very hard | 19.2% |
| Hard | 53.7% |
| OK | 20.9% |
| Relatively easy | 5.8% |
| Easy | 0.4% |

**Table 5: Reproducibility issues**

The self-assessed average time needed to analyze a concurrency bug is given in Table 6. It often takes days of work to analyze a single concurrency bug.

| | |
|---|---|
| Hours (but less than 1 day) | 27.4% |
| Days (but less than 1 week) | 63.4% |
| Weeks (but less than 1 month) | 8.3% |
| Months | 0.9% |

**Table 6: Time to debug**

Most respondents state that on average, concurrency bugs are of Severity 1 and 2 (see Table 7).

| | |
|---|---|
| 1 - Most severe | 25.8% |
| 2 | 58.7% |
| 3 | 13.5 % |
| 4 - Least severe | 2.0% |

**Table 7: Average severity of concurrency bugs**

Note that the overall time spent by all the respondents of this survey to debug and fix all their self-reported concurrency bugs represents thousands of days of work.

# 5. DISCUSSION

We also asked respondents to select what they thought was the most common source of concurrency bugs amongst the possible options of “design”, “coding”, “more of a design issue and less of a coding issue”, “more of a coding issue and less of a design issue”, and “an equal mixture of both”. Most respondents thought that concurrency bugs were either due to coding issues or a mixture of design and coding issues.

Going forward, 65% out of 428 concurrency-facing respondents feel that concurrency issues are going to be more problematic.

Also, most respondents said they would strongly benefit from:

- Better tools: debuggers for multithreaded/process programs, capture/replay tools (to help reproducibility), static analysis, etc.;
- Better libraries with pre-packaged concurrency mechanisms/patterns;
- Better compilers (that would automatically take care of concurrency);
- Better programming languages, designed for concurrency;
- Better code reviews, guidelines, developer training, education.

The bottom-line: a lot of improvement is possible in all those directions, and seems necessary to master the challenges of concurrency. Furthermore, many of the current tools used are not well adapted to concurrency.

## ACKNOWLEDGEMENTS

We thank several senior engineers in Windows, Windows Live Core and in Visual Studio for their early feedback during the preparation of this survey. We also thank Tom Ball, Andrew Begel, Jim Larus and David Molnar for their comments on previous versions of this report.