CHAPTER 19

Metrics and ALM Assessment for Developer Practices

Developer practices are really what Visual Studio 2012 is all about. Microsoft offers several reports and metrics we can use for analyzing and assessing our code base. There are also many assessment questions in the online assessment targeting this area. Let us start with the metrics.

Metrics

Using the metrics from TFS we can find KPIs that will help us understand if we are successfully working to improve our code. These are useful from both architecture and design viewpoints as well as from a developer viewpoint. Using them will help us improve how we design our application or system.

There are several important metrics we can get automatically from Visual Studio and TFS to get a good understanding about the quality of the development work, including

- Code coverage
- Code metrics
- Compiler warnings
- Code analysis warnings

Code coverage

Code coverage shows us how much of the code has been covered by our automated unit tests. We get the value as a percentage of the whole code base. The difficulty is often to decide what percentage is enough. Should we always strive for 100%, or is 80% enough? This is something the team has to discuss with the product owner in Scrum or any other similar decision maker in other processes. This value is input for the Definition of Done (DoD).

Code metrics

There are different code metrics we can get:

- *Lines of code* is an approximate number based on Intermediate Language (IL) code. A high count might indicate that a type or method is doing too much work and should be split up. This might also be a warning that code will be hard to maintain.
• **Class coupling** measures the coupling to unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration. Low coupling is better to strive for because high coupling indicates a design that is difficult to reuse and maintain because of its many interdependencies on other types.

• **Depth of inheritance** indicates the number of class definitions that extend to the root of the class hierarchy. The deeper the hierarchy, the more difficult it might be to understand where particular methods and fields are defined and/or redefined.

• **Cyclomatic complexity** is created by calculating the number of different code paths in the flow of the program and shows the complexity of the code. A high complexity makes the maintainability suffer, and it can also make it hard to get good code coverage.

• **Maintainability index** is an index value between 0 and 100 that represents the relative ease of maintaining the code. The higher the better. A rating above 19 is good. Below that, maintainability suffers.

### Compiler warnings

Errors and warnings should be avoided in a project. Allowing more than zero errors or warnings tends to result in the team accepting a lower quality on the codebase, which over time will cause the code to lose maintainability (commonly known as the broken windows theory [see http://en.wikipedia.org/wiki/Broken_windows_theory]).

Track this metric to make sure the number of errors is zero. This should ideally be enforced by automatic build policies (as described in Chapter 26).

### Code analysis warnings

Code Analysis in Visual Studio performs static analysis on code, which will help developers identify potential design, globalization, interoperability, performance, security, and a bunch of other categories of potential problems.

The Code Analysis tool provides warnings that indicate rule violations in managed code libraries. The warnings are organized into rule areas such as design, localization, performance, and security. Each warning signifies a violation of a Code Analysis rule.

Code analysis can be used to enforce company policies on the code developers write. We can extend the ones Microsoft offers by writing our own rule set or we can suppress the ones we do not want. Definitely discuss this with your development team and the product owner, as the warnings will have an impact on the effort required before the Definition of Done is fulfilled.

### Standard Reports

The Quality Indicators report (Agile, CMMI templates) (see Figure 19-1) gives us an overview of the code quality metrics over time. Based on automated builds, the report shows how code churn, code coverage, test results, and the bug count vary over time.