Making Code Generation Real

Five Requirements for Effective Code Generation

The promise of effective code generation is tremendous. It can accelerate development, increase productivity, streamline maintenance, and improve system quality by orders of magnitude.
But, the demands of effective code generation are also significant. There are basic requirements that must be met in order to ensure a productive environment. An approach and toolset that supports these basic requirements can greatly increase project productivity and quality. However, one lacking these critical capabilities will become either an expensive drawing tool or an obstacle to project success.

Outlined in the following article are the five basic requirements for effective code generation that will make each project a success.

Complete control of how the code is generated -- every system is different.

Embedded systems require architecture, design, and implementation decisions specific to the system under development. Projects cannot afford to be constrained by vendor-enforced, pre-defined design and implementation decisions.

Effective code generation requires the development team to have full control of all software architecture, design, implementation, and code characteristics. Code-generation capabilities must be fully customizable so that vendor-provided options can be modified or changed in any way necessary to meet the specific demands of the system under development.

The model compilation phase is provided in a fully customizable format. Off-the-shelf model compilers can generate highly optimized code as is, or they can provide a good starting point for any customizations or extensions specific to the target under development.

Granular control of generated code -- all code is not created equal.

Different parts of a single system may demand different design or implementation characteristics. The system as a whole may be constrained by size, while certain threads may be critical to performance. Sub-optimal code, required project workarounds, or abandonment of the generated code can be caused by the lack of ability to define different design characteristics for different parts of the same system.

The model compilation phase provides the ability to “mark” or “color” any threads or portions of a UML application model so that different parts of the system can be generated with different design and implementation characteristics.

Integration of generated code -- no code is an island.

Many times it’s neither practical nor efficient to model and generate all parts of the system. Generated code may often need to co-exist and integrate with legacy code, COTS, handwritten code, and code generated with other tools. Generated code must integrate easily and automatically.

Completeness of generated code -- partial code generation provides only a partial solution.

Effective code generation needs to produce complete target code in order to provide full productivity and quality benefits. Tools that generate incomplete code frameworks require developers to hand write the missing code bodies. This partial generation approach creates a need for project procedures and tool utilities to help manage synchronization between the generated framework and the hand-written code bodies. The time and processes involved keeping these intermixed pieces of the system synchronized can impact a project's tight schedule and reduce project productivity.

Quality in/quality out -- Quality code requires quality models.

Effective code generation relies on quality models as input. The earlier errors are found and eliminated, the cheaper and easier they are to fix. Defects should be eliminated before ever producing code while models need to be syntactically, semantically, and behaviorally correct before code generation. Deferring elimination of these defects to integration and test would result in reduced project productivity and system quality.

# # #