Automatic Code Generation Through Model-Driven Design

Patrick W. Burke
Principal Software Engineer
1 May 2008
Motivation

Software Engineering Industry

- GOAL (Same as 40 years ago)
  - How to avoid introducing error
  - Worst case: Find errors early
    □ Dr. Barry Boehm: “Finding and fixing a problem AFTER delivery is often 100 times more expensive than finding the problem in the requirements phase”

- How Bad Is the Software Industry: (Jerome Krasner, EMF)
  - SW development is responsible for 80% of program/design delays
  - Over 33% of SW developments produce a FINAL PRODUCT that is NOT within 50% of its predesign expectations.
Automatic Code Generation technology is mature enough to improve the software development process..

FASTER, BETTER, CHEAPER

Model-Driven Design as a front end
Agenda

- Software Development History
  - Those who do not study history are doomed to repeat it

- Model-Driven Design

- Automatic Code Generation
  - Requirements
  - Current practices
  - Problems
  - Areas in need of research

- Where do we go from here?
Figure 1. Software Engineering Evolution
(by Dr. Barry Boehm)
Software Engineering: History

- Software Development Processes are highly people-dependent
  - Quality proportional to skill level and availability of key reviewers
  - Minimal automation beyond archiving and versioning
  - Order of Magnitude improvements are unachievable in an expert-dependent system

- True automation within Software Engineering is the key to quantum level improvements
Software Engineering: History

Software vs. Hardware Engineering Disciplines
- Software end product is not physical
  - Binary representation of HUMAN THOUGHT
- Hardware capability has experienced exponential growth
  - Software struggles to use those capabilities
- What is the difference in the disciplines
  - Integrated Circuits
    - TTL Book
      » Software does not have a “Holy Grail”
  - CAD Tools
    - Independently tested components
- Smart, innovative, skilled humans develop tools to allow:
  - Exponential growth
  - Error-free software
  - by engineers who are experts in the application domain, not in software engineering
SW Engineering: Future

- Future of Software Engineering (Dr. Boehm)
  - Model-Driven Design
  - Integrated Software-Systems Engineering
  - System of Systems
  - Enterprise Models
  - Autonomy
  - Bio-Computing

- All can significantly benefit from UML, simulation, and code generation
Process

- Current Software Engineering mantra
  - Software Engineering Institute
    - SEI Levels
    - CMMI
  - Continuous PROCESS improvement
    - PRODUCT improvement is an ancillary benefit
  - Generally, coding starts after design ... one line at a time
    - Just like we did it in the 70’s

- Grady Booch: Don’t let process stifle innovation. (SSTC’06)
Model-Driven Design

- Predominantly Non-textual representations which provide an abstraction of a software system
  - A picture is worth a thousand words!!!

- Benefits
  - Shorter design cycles
  - Early error detection through model execution
  - High quality products
  - Continuous validation

FASTER, BETTER, CHEAPER
Model-Driven Design

Requirements

- Simple, unambiguous, user interface
- Universally accepted modeling language (UML/SDL)
- Abstract enough to be platform independent
- Expressive enough to allow sufficient detail to describe how the design can be realized
- Must be VALIDATED
- Allow direct model execution as a prototype
  - Automated verification/validation sequence support desirable
    - Allows validation at any point in the development
    - Constant validation is the key to producing high quality systems very quickly (Jerome Krasner, EMF)
Requirements (cont.)

- Allow incorporation of hand-written code segments
  - Highly specialized domain
  - High efficiency or tight timing constraints
  - Legacy
  - Hardware-dependent

- Reverse-engineering support
  - Incorporate legacy/external code/design
  - Automatically generate corresponding design/code

MORE AUTOMATED CODE = LESS HUMAN-INDUCED ERRORS
Model-Driven Design

Model Validation .. NASA Style

– Traditional “Input-Processing-Output” sets for testing cannot exhaustively test anything but the most simple models

– NASA is studying a “Proof of Correctness” System
  – Requirements to Design to Code (R2D2C – Patent Pending)
    □ Requirements written as scenarios
    □ Formal model derived with guarantee of equivalence to requirements
      » Currently using Hoare’s language of Communication Sequential Processes (CSP)
    □ Requires development of
      » Independently provable inferences
      » Knowledge rules to implement laws of concurrency
    □ Cost may be justified for
      » Small project
      » Human life at stake
      » Extremely expensive national resource at stake
Model Validation .. NASA Style

- Weakest Link:

  “… While engineers are happy to write descriptions as natural language scenarios, or even using semiformal notations such as UML, they are LOATH to undertake formal specifications.”

[Hinchey, Rash, and Rouff]
MDD Process Problem

- Cost Estimation

- May not match current/historical software development processes
  - Where is the expected “stepwise refinement”
  - What is delivered at SRR, PDR, CDR, TRR ...

- Model Driven Architecture Guide from the Object Management Group (OMG) provides:
  - Computation Independent Model (CIM)
  - Platform Independent Model (PIM)
  - Platform Specific Model (PSM)
MDD Process Problem (cont)

- “Design Review” replaced with “Model Review”
  - Verify that textual specified requirements are realized in the model
  - Compliance with modeling guidelines
  - Quality criteria (portability, maintainability, testability, etc.) are met
  - Check that model meets requirements for safe and efficient code generation

- Effectiveness of this review is highly dependent upon the skill of the reviewers.
MDD Research Opportunities

- Apply Proof of Correctness to a safety-critical thread in a larger system
- Formal specification techniques
- Formalization of semantics in modeling languages
- Reuse of model components (software reuse)
- Development of reusable models as “Patterns”
- Domain-specific modeling languages
- Systematic Software Testing
- Compilation technologies for models
Historic Code Generation Process

- Look at design artifacts, start coding
  - Begin .... End
- Peer Review of Code
- Unit test
- Functional integration and test
- System integration and test
- Dry run acceptance test, dry run acceptance test, ...
- Get final code build
- Acceptance test
- Deficiency report
- Dry run acceptance test, ...
- Acceptance test
- Done
Automatic Code Generation

- Just press the Generate Code button?

- Research indicates
  - Higher productivity
  - Uniformly high quality
  - Generation of 80-90% of final application code from model

- REALITY CHECK
  - Unless “provably correct” by design, some degree of testing will still be required
ACG Requirements

- User friendly
- Allow direct insertion of external code segments
- Reverse engineer external code into model
- Generated code must be human readable
- Generated code must be VALIDATED
  - If human review of generated code is deemed necessary, the efficiency of automatic code generation will be quickly lost
  - Accommodate automatic testing to requirements to the maximum extent possible
ACG Process Problem

- Cost Estimation
- “Peer Review” of code required until code generators are trusted
  - Must review code and model simultaneously
  - Find errors introduced by inappropriate modeling
  - Identify errors which are difficult to detect in the model but more easily found in the code
  - Ensure that custom (inserted) code segments are properly integrated
  - Identify possible efficiency improvements

Effectiveness of this review is highly dependent upon the skill of the reviewers.
Other ACG Problems

- Cycle time for change is significantly greater
  - MDD/ACG favored if number of problems is low

- Code generated and tested for a generic platform must be regenerated and tested for the target platform
  - Double testing cost can be avoided with automated testing

- Scalability to large programs not yet established
Automatic Code Generation

Research Opportunities

- Debugger technology at the model level
- How to create Human readable and documented code
- Unambiguous formal requirement specification techniques
- Automated testing to requirements techniques
  - R2D2C approach is prohibitively expensive for normal programs
- Study of scalability of ACG
Conclusion

- MDD/ACG is an emerging technology with significant potential for improving the efficiency and reliability of the Software Development Process.

- Needs large-scale experience to identify any latent deficiencies.
Where Do We Go From Here

- Government:
  - Start funding MDD/AGC Projects
  - Work with industry to tailor processes
  - DARPA MDD/AGC projects

- Industry:
  - Start investing in research
  - Start prototyping MDD/ACG technologies
  - Start training engineers

- Academia:
  - Add rigorous modeling training to curricula
  - Encourage MDD/ACG research
List of References

- Hinchey, Rash, and Rouff, A Formal Approach to Requirements-Based Programming: 5.
- Elizabeth A. Strunk, Xiang Yin, and John C. Knight, "Echo: A Practical Approach to Formal Verification," FMICS'05 Lisbon, Portugal 5-6 September 2005: 44.