5.4 Data Science, Artificial Intelligence, and Machine Learning

One of the themes of early work in artificial intelligence (AI) was referred to as "automatic programming". The essential diversity and complexity of software and the underlying tacit and difficult to precisely express technical and domain-specific knowledge required to build software are still beyond the feasibility of current techniques.

While the more mundane aspects of software development have been susceptible to greater automation, the more creative aspects have been satisfied, in an industry focused on low-investment/short-term economics, with what continues to be labor-intensive practices. Although AI efforts have not supplanted traditional software-based development practices, AI-inspired advances offer opportunities for improvements in how software is developed, described here in the context of the DsE methodology.

Build a Sample-Based Component Family Generator

Build software to construct a product family based on the content of a set of existing components having perceived "similarity" of purpose. Represent the family as a metaprogram with associated differential decisions whose resolution is sufficient to derive customized instances. Iteratively refine the family based on more samples to enable derivation of additional or improved instances. (Optionally, normalize hardware and software services interfaces to reduce incidental differences in sample component content.)

Build a Capabilities-Based Product Generator

Analyze a set of previously developed products in order to derive a taxonomy of similar capabilities. Derive a component family to represent each type of capability based on relevant normalized content of example products. Build software to derive a product as the composition of selected customized constituent capabilities.

Architecture Refinement for Quality Improvements

Derive alternative candidate product architectures based on analyses of alternative combinations of behavioral quality factors and alternative candidate software components corresponding to those combinations.

Enhanced Code Transformations for Optimization

Construct a correlation between hardware and software services alternatives as they affect product quality factor outcomes. Derive an analysis of dependencies among product quality factors.

Delineate Behavioral Quality Factors at the Component Level

Analyze how each component in a product architecture is expected to contribute to the various behavioral quality factor outcomes; evaluate candidate component instances in terms of fit to this contribution (i.e., how alternative solutions differ in quality effects).

Optimize Software for Hardware Mapping and Configuration

Define a computational platform family whose instances are characterized and selectable in terms of specified software capabilities to be supported; if computational capabilities are constrained, build software to fit an available computational platform configuration that is a sufficiently close fit; estimate how behavioral quality factors are affected by alternative computational platforms.

Trace Alternative Decisions to Product Quality Effects

Given an appropriate decision-based characterization of the instances of an envisioned product family, determine how alternative decision resolutions affect behavioral quality factors of the corresponding instance products. (enabling direct capability-quality tradeoffs)

Code Derivation from Hardware Behavioral and Interface Specifications

Derive alternative software emulations of hardware devices based on a software specification of its intended behavior and an analysis of these alternatives based on tradeoffs among behavioral quality factors; encapsulate hardware functionality within software that extends a device's capabilities with enhanced behavior (e.g., health monitoring for repair or replacement based on diagnostics and prognostics, data logging, data stream prediction and interpolation)

Develop Alternative Ways of Representing a Component or Product Family

{eg transformational or neural network analyses that isolate and compose behaviors}

Build Introspective Software for Explanation and Failure Management

Create the means to build efficient self-aware software that can analyze and determine rationale for its behavior to provide explanations by which developers can evaluate correct behavior and detect flaws. Extend to provide similar capabilities by which a product can instruct, monitor, and improve its effective use. Create capabilities by which software can monitor and evaluate specified hardware behavior so as to detect or predict failures and guide self-correction or diagnostic- and prognostic-based remediations.