1.1 Introduction

This text is a systemic reconception of the engineering and manufacture of software-based products. A software-based product is an article created and employed to induce or change the capabilities, properties, or behavior of a system of interest so as to improve the degree to which that system supports particular objectives of an enterprise.

This is a generalized response to Dijkstra's admonition[1] to regard every software program as a member of a *family* of related programs, "as alternative programs for the same task or as similar programs for similar tasks". He suggests that it is not enough to just build an initially acceptable product: one must be able to argue that the resulting solution is "a deliberate choice out of a possible set of things you could have done", reconsidering that choice when changed circumstances require a modified solution.

Six precepts characterize the nature of building modern software-based products:

- A product is conceived to serve a well-defined purpose. It provides capabilities that enable a customer enterprise to operate effectively toward perceived objectives. A well-conceived product lacks nothing essential and includes nothing extraneous to its intended purpose. It must conform to or induce changes in how a customer enterprise operates.
- A product operates according to its purpose to influence the behavior of an ecosystem (i.e., entities interacting within a shared environment). A product is built to observe and induce changes in an ecosystem based on an understanding of the nature and natural behavior of the environment and associated entities, the product's purpose and agency, and the mechanisms available to observe and influence behavior.
- The effective ability of an enterprise to build products relies on coherent focus, continuity of purpose, a shared discipline of practice, and effective competence.

[1] See extract of Dijkstra from *Structured Programming* on program families (1972) & EWD249.pdf (Apr 1970) in Appendix

  - Coherent focus is the alignment of enterprise objectives, focusing the purpose for which the enterprise has been established, with a well-defined market focus.
  - Continuity of purpose is the commitment to maintaining the established purpose of the enterprise as the rationale for all past, current, and future endeavors of the enterprise. This provides a basis for treating past products as a retained capital asset, recognizing and benefiting from the residual value of product elements, as an investment in future capability that can be leveraged in building similar future products, reducing repeated effort and associated risk. Elements of past products may be useful in building subsequent products or provide insights regarding how to resolve particular engineering tradeoffs.
  - A shared discipline of practice is a commitment to the maturity, suitability, and compatibility of disciplined management and technical practices that are appropriate to the type of problems and potential range of solutions to be realized in products.
  - Effective competence is the alignment of collective developer problem-solution *competence* (knowledge, experience, and expertise) with enterprise development efforts. In particular, developers with experience building similar products require less effort to identify and resolve problem-solution uncertainties and tradeoffs in building an acceptable product.

- Every well-defined problem has many potential solutions. Each such solution must satisfy the criteria and constraints that characterize the problem but will differ from others in both essential and incidental aspects. Incidental differences, which arise due to different techniques, terminology, or conventions but result in products that exhibit equivalent behavior, can be avoided through use of disciplined practices. Conversely, essential differences represent different resolutions of critical tradeoffs in functionality, quality factors, and total lifecycle costs that must be weighed. Alternative solutions, due to these differences, can differ in significant characteristics such as performance, ease of use, accuracy of

data, or ease of change. Differences in underlying hardware and software choices and interactions with the operational context (environment, people, and systems) can favor some solutions over others. Focusing prematurely on a particular solution, to be iteratively improved through trial and error leading to repeated rework, without properly evaluating among feasible alternatives and associated tradeoffs, can result in a product that has been costly to build, is an unnecessarily inferior fit to the customer's needs, and is difficult to modify over time. Still, the most viably buildable product may be an imperfect fit to those needs, only approximating what would be an ideal solution. The answer is to determine the space of potentially viable solutions that can be comparatively evaluated based on essential differences to find a best fit to problem criteria and constraints.

- A problem must be properly understood to be properly solved. The initial conception of a problem is often an incomplete, inconsistent, or poorly communicated understanding of needed capabilities. Perceptions of the problem may differ due to uncertainties, unknown or undisclosed information, differing assumptions, and conflicting opinions that should not be arbitrarily or prematurely resolved. In any case, the actual problem is not static but is prone to change over time due to changes in needs, circumstances, or enabling technology, sometimes even as a solution is being developed. Understanding of a problem and its solution tends to change due to insights gained during both development and use of a product. Changes in the problem as understood lead to changes in the solution. An effective development approach will anticipate and support the ability to change the solution as problem understanding changes, while maintaining the consistency and integrity of the product as a whole.
- Every constructible product is similar, in varying degrees and aspects, to other products, past, present, and future. In all cases, similarity describes the change that would be required to cause a product to have behavior equivalent to that of a different product. This implicitly includes the similarity that exists between different versions of a single product.

These precepts are the foundation for an industrial approach for systematically building software-based products. Recognizing that similar problems are amenable to similar solutions is the basis for leveraging effort needed to build similar products. Customers needing substantially similar capabilities may be satisfied with a single (“one-size-fits-all”) product but may need customized solutions if their essential needs differ by too much. Customized solutions may be custom (“one-of-a-kind”) products or instances of a jointly managed set of products for customers that have reasonably similar needs. As traditional approaches poorly account for uncertainty, change, and diversity and treat software as a recurring cost rather than as an investment in future capability, a different approach can better support the predictable and cost-effective building of either singular changing products or sets of similar changing products.