1.3 Foundational Constructs

Along with the preceding exploration into basic concepts and associated terminology, an appreciation of several foundational constructs—software development as a discipline, process and product quality, directed reviews, and the causes of product change—contribute to an understanding of the approach being described. These constructs are not unique to the described approach but the approach builds on these as foundational elements.

The Evolution of Software Development as a Discipline

Every systematic discipline of practice has started as an opaque individualized craft. As experience is gained, tools and techniques are developed that reduce routine and the more mechanical aspects of the effort. Fabricated parts are created to further reduce the detail work required to create a finished product. At this point, it is still a fundamentally "manual" effort that relies on the skills and efforts of individuals to fashion more or less custom products.

As understanding grows of a reasonably standard approach to creating a type of product, machinery is developed for automation and mass production of such products, with efforts beginning to focus on how to improve productivity and product quality. Improvements in mass production ultimately lead to mass customization for products that are individually tailored to specific needs and preferences, partially regaining some of the benefits of handcrafting but with greater consistency and reduced labor.

Software development is following this same progression. It is now at the stage of being a tool- and parts-supported craft within a somewhat standardized process. However, for software, this advance is encumbered by an outdated mentality that originated with a focus on building unique, statically-defined products meant to solve well-understood problems.

This craft-based approach for software has been augmented over time, with more disciplined practices, introduction of tools supporting more mechanical aspects of development (e.g., configuration management and builds, documentation, regression testing), and greater use of previously developed parts (i.e., commonly useful

functionality). Although many of the basic principles and practices of software engineering are well-established based on 50 years of experience, the form these take will continue to progressively change. These and other advances are a foundation for an industrial mindset that will, in supplanting systemically-flawed assumptions of traditional practice, lead to the ability to mass produce customized products.

Separation of Concerns

Separation of concerns is the principle of organizing a complex matter into a constituent set of elements (of action or information). Each such element is conceived to have singular authority over a specified scope of responsibility and can be understood and realized in terms of its unique contribution to the matter as a whole.

This principle guides the structuring of, for example, a process into activities, an enterprise into programs, human activity into roles, and hardware and software functionality into discrete components. This is the basis for the concept of information hiding that Parnas conceived as the foundation for designing software for ease of change[1].

This principle motivates distinguishing between engineering and manufacturing as distinct elements of product development. Manufacturing is seen as the realization of products whereas engineering is seen as providing the means for effective and efficient manufacturing. A third element, management, is seen as the direction and coordination of engineering and manufacturing efforts within an enterprise.

The Need for an Engineering Discipline of Software

In early days of building software, when problems to be solved were simpler, code could be written using a simple notation to just mimic the steps a person would take for a solution. Code was written in discrete sections (e.g., blocks or subroutines) that provided sufficient computational separation of concerns. Such code did not have a complex structure and could be quickly understood by others. The translation from a human description of the problem to a machine-processable solution was simple. The

[1] D.L.Parnas, "Designing software for ease of extension and contraction", ICSE'78, ACM, May 1978, p 264-277.

existing way of that people worked was affected only by the selective automating of discrete, previously manual tasks. Software was and often still is built by devising a human expression of intended behavior that is then translated into a notation that a computational device can follow to enact that behavior.

In due course, as problems have become more complex and solutions more long-lived, software has come to need a more clearly defined structure for subsequent understanding of how it should be expected to behave and what to change if different behavior is needed. With obscure relationships among different parts of the code, changes in one part can indirectly affect other parts. Behavior has become harder to predict and more likely to be different than expected. More time has to be spent trying to be sure the software works correctly. After all that, the software may still work differently than users had imagined or needed. As users' needs change over time, the software becomes still more complicated due to unplanned changes and yet more difficult to get correct and then change further. Of greater concern, the time from recognizing the need for a product and having that product available for use has grown increasingly long and even then often requires users to change how they prefer to work for the solution to work effectively.

With the growing complexity of problems and solutions, an engineering discipline is needed for building software-based products. Engineering is the systematic application of competence (knowledge, experience, and expertise) to formulate a solution to a problem, iteratively considering properties and tradeoffs among alternatives to determine a solution that best satisfies all aspects of the problem. In best cases, software is developed within an enterprise that has a specific focus and experience building particular types of software. Practitioners in such an enterprise have the competence to build such software more efficiently and effectively, leveraging past work and creating a proper foundation for building and evolving other similar software-based products, than those who lack such competence. As similar products are built, an understanding of how similar problems can differ but also lead to similar solutions becomes the basis for a streamlined approach to building such products.

Process and Product Quality

The "value" of a thing is its utility for a given purpose and its quality in serving that purpose. Quality is the degree to which the thing approximates the "ideal" form of such a thing for a given use. Quality is ultimately a subjective discernment but, for an enterprise to prosper, it benefits from having the means to continually improve the quality of its efforts. Quality improvement involves having measurable goals for its endeavors, mechanisms for determining how well those goals are being achieved, and actions to be taken when ways are found to improve both guidance for and associated practice of endeavors[2].

An enterprise is concerned with two sorts of quality: product quality and process quality. ***Product quality*** is the degree to which a result (i.e., a product or service) approximates its potentially achievable best form for satisfying its purpose. ***Process quality*** is the efficiency and effectiveness (equivalent to "productivity") of an effort intended to achieve an envisioned result. Reviews of all work should be performed to determine the degree to which both process and product quality objectives are being met.

Product Quality

Product quality represents the degree to which a product is suitable for its intended use. Product quality is dependent on the process with which the product is realized in that the process determines the criteria and attention given to achieving the various elements of product quality. Product quality can be improved either by increasing the effort spent on evaluating and achieving product quality goals or by enhancing the process with more effective means of addressing product quality. Product quality for a software-based product (elaborated in section 2.4 as product "behavioral" quality) is evaluated in terms of four aggregate properties[3]: functionality, performance, dependability, and usability.

---

2 W.E.Deming *Out of the Crisis*, The MIT Press, 1982. <https://direct.mit.edu/books/book/4192/Out-of-the-Crisis> <https://deming.org/explore/fourteen-points/>

3 "functionality" is included as a behavioral quality factor in that it may need to be traded against other factors (i.e., limited in order to achieve e.g. security, safety, or performance goals).

Expending greater effort on product quality can be viewed as reducing productivity unless the resulting improvement in product quality is recognized as requisite to product viability, whether initially or over its useful life. The challenge is to find a proper balance between acceptable quality and timely cost-effective deployment of a needed product. The process cost of improved product quality can be mitigated by improvements in the process that increase potential productivity, reducing the effort needed to achieve given product quality goals.

A less direct dependence exists between product quality and the quality of the customer's operational process: the effectiveness of the product depends on its capabilities and use being a proper fit with the customer's process and practices. Another dependence exists between development process quality and the quality of provider enterprise business practices (discussed in section 2.1) (e.g., legal/financial, human resources, technical infrastructure) upon which development practices rely.

Process Quality

A defined process specifies how actions are organized to accomplish some purpose, comprised of a partially ordered set of activities and the practices to be used for each activity. Process quality represents the degree to which a properly defined and performed process is effective for reliably achieving an intended result.

The objective of a development process is to build a product whose capabilities satisfy some need. The goal of productivity is to reduce the delay between recognizing the need for a product and the consequent effective operational use of that product. This goal persists as needs change over the useful life of the product, requiring repeated modifications. Process quality for development of a software-based product (elaborated in section 2.0 as "developmental" quality) is evaluated in terms of four aggregate properties: feasibility, sustainability, conformability, and verifiability.

Evaluating Quality

Evaluations of process and product quality are performed for the purpose of guiding management decision making regarding how an organization is working and discovering needed changes. Evaluations should follow a goal-question-metric (GQM),

or similar, discipline[4] in order to limit needed effort by obtaining only relevant information. Goals express management-specified objectives through associated process and product quality metrics for the type of work being performed.

Process quality is evaluated to determine the degree to which a process is being properly performed whereas product quality is evaluated to determine the degree to which technical objectives for consistency and completeness of all specifications, realizations, and supporting materials are being achieved. Reviewers should be experts or peers having competence in evaluating quality criteria for the type of process or product being reviewed.

Quality Perspectives on Process Variation

There are three perspectives on process variation that inform achieving an envisioned level of productivity for a targeted level of product value:

- ***Capability*** – The range in productivity achievable in following a given process, relative to the complexity and uncertainty of a problem-solution
- ***Maturity*** – The predictability with which performance of a given process can be expected to achieve that process' potential level of capability
- ***Performance*** – The productivity achieved for a resulting product value by practitioners following a given process

The objective for productivity is the high-maturity performance of an appropriate high-capability process. Process maturity is the alignment of an effective process with the competence of practitioners in performing that process and its associated practices. The capability of a process is the limit in productivity that practitioners can be expected to achieve in the performance of that process.

Process quality can be improved in different ways relative to these perspectives:

- modifying aspects of process performance to reduce rework by better avoiding product defects and anticipating planned changes

[4] V.R. Basili, G. Caldiera and H.D. Rombach, "The Goal Question Metric Approach", Encyclopedia of Software Engineering. Wiley 1994.

- providing training and mentoring of practitioners in best use of the process and associated practices to improve the maturity of process performance
- revising the process and associated practices to better align with practitioner competence
- revising the process and associated practices to increase potential capability by streamlining, automating, or eliminating activities.

Directed Reviews

A directed review is a structured evaluation[5] of the quality of in-progress or completed content. The author responsible for that content initiates timely reviews as needed with the assistance of reviewers having relevant competence. A directed review proceeds as follows:

- The review is focused on specific author-delineated content, including any relevant dependencies with other content.
- The author provides specific questions that 2-4 reviewers are to address in examining the content.
- The author specifies the particular competence needed of each reviewer to be involved in the review, emphasizing diversity of perspectives in relation to specified questions and content.
- Reviewers are assigned, with each separately examining relevant content and responding to assigned questions with competence-informed observations and any suggestions for improved content.
- The author convenes a meeting with the assigned reviewers to ensure an accurate and consistent view of their responses, supporting any needed improvements in the reviewed content.

[5] D.L. Parnas and D.M. Weiss, "Active Design Reviews: Principles and Practices", *Proceedings of the 8th international conference on Software Engineering*, August 1985, Pages 132–136.

Reviews are kept small (2-4 reviewers) to ensure that responses provide focused, timely, and substantive assistance to authors. Questions are formulated to fit each reviewer's competence and are expected to focus on areas of author uncertainties; clarity, sound rationale, and completeness of content; consistency with any related content; and implications of any potential future changes. Reviews may recur as content is revised until completion criteria for that content is satisfied.

The Causes of Product Change

A product is built to provide capabilities that support the needs of customers in a targeted market. Although it is natural to think of a product as being the fixed realization of some associated solution to a specific problem, both the problem and its solution are susceptible to change. A product is better viewed as a changing solution to a changing problem.

Three factors lead to a product being changed: problem-solution uncertainty, product deficiencies, and problem-solution changes. A realistic approach to development must account for each of these factors for a product to be sustained over its expected useful life. These three are all interrelated in that each of them can arise even as either of the other two are being addressed (e.g., a deficiency that is the result of an incorrect resolution of some uncertainty, an identified problem-solution change that changes how a problem-solution uncertainty should be resolved, or a problem-solution uncertainty that is not recognized until a problem-solution change occurs.)

Problem-solution uncertainty is a consequence of problem-solution complexity and the limits of human capability (or quoting Dijkstra: "On our inability to do much"). Product deficiencies express when the product is found to be flawed, having defects that degrade its ability to satisfy its envisioned purpose and capabilities, or when the product as released is acceptable but incomplete. Problem-solution changes are the result of changing customer needs, including changes in operational circumstances or enabling technology.

Problem-Solution Uncertainty

A product is a particular solution to an understanding of a problem. However, based on two of the precepts of software development (from section 1.1), uncertainty is a concern in two respects: is the actual problem properly understood and which of its many potential solutions is a best fit for the problem? The essential challenge, and substantial effort, of product development is resolving problem-solution uncertainty.

An effective solution requires a reasonably correct and complete understanding of the problem and the behavior that the product must exhibit to provide the customer with the capabilities needed. Certainty is an unobtainable ideal but uncertainty has many sources potentially subject to mitigation for improved confidence in the problem and a suitable solution. Uncertainty about the problem arises from unknowns (including excluded information due to security/proprietary/competitive concerns), tacit or overlooked assumptions, inherent complexity, miscommunications, ambiguity, and differing user or expert opinions about the problem or solution based on prior experience. These uncertainties can usually be satisfactorily resolved through customer-developer collaboration in exploring subject matter knowledge, customer objectives and practices, and circumstances related to operational context.

Uncertainties about the problem and its possible solution demands an iterative "learning" process of development through which understanding of the problem improves and may change and the nature and tradeoffs of alternative solutions are explored and comparatively evaluated. A premature or ill-founded resolution of uncertainties gives the illusion of problem-solution understanding but results in development of a deficient product.

Development of a product begins with a general conception of capabilities that the product should provide for its intended purpose. Such a conception lacks specificity in that many different products could in principle satisfy an insufficiently precise description of the problem or its solution. The process of creation is one of refining this conception by identifying and comparing alternatives in the problem and solution, considering issues of feasibility, practicality, constraints, convention, and preference imposed by the context of usage, to determine what form the product will take. The

effort involved in achieving a satisfactory product is dependent on the complexity of the actual problem and the viability of creating an appropriate solution to that problem.

Achieving a good solution is expedited with competence gained from solving similar problems but uncertainty can remain when alternative solutions fail to adequately resolve problem-specific conflicts among related aspects of product quality. Uncertainty about the solution concerns what combination of development choices will result in the right functionality with appropriate quality criteria, while satisfying any extrinsic constraints on the solution. Determining the best solution entails understanding the nature of the context in which the product operates and the interactions that result, how the solution addresses problem-imposed quality criteria and solution constraints, and how computational platform capabilities cause product behavior to differ with alternative solutions. Each of these aspects brings some degree of uncertainty that must be resolved to determine an acceptable solution.

A particular type of uncertainty relates to how understanding of the actual problem can change as development proceeds. An inaccurate understanding of the problem can result in resolving solution uncertainty that ends up as a deficiency in the product. Furthermore, the actual problem itself or recognition of possible solution alternatives can change even as the product is being developed, warranting changes in the solution to provide the customer with more effective capabilities.

Solution Deficiencies

A product is typically a satisfactory solution to the problem it addresses, seldom a perfect solution. The complexity of most software, and the challenges entailed in its development, means that it is going to have undiscovered defects. Furthermore, tradeoffs made among quality factors can lead to unforeseen emergent effects that could be improved with changes in the solution to achieve a closer fit to the problem. (quality factors are not absolute, every solution is an approximation to an envisioned ideal)

In addition, a sound practice of software development is to build a product that is a useful but incomplete solution to the full problem being addressed. The practice is to incrementally develop an effective solution to a partial problem, incrementally

addressing more aspects of the problem for an augmented solution in a series of product releases. The initial release provides a coherent set of capabilities but subsequent releases improve or add capabilities that the customer needed. Developing a product in this incremental way may result in the observable behavior of each release of the product purposefully differing and, in some cases, expand the market to which the product applies.

A customer's use of a product sometimes reveals that their endeavors would be improved with changes in product capabilities. Such changes may be needed for the customer to change how they perform an endeavor rather than simply to better support existing practices. This experience may be considered by the customer as being a deficiency in the product when it is actually a needed change in the problem or solution that the product has been properly built to realize.

Problem-Solution Changes

The suitability of a product can degrade over time unless the product is changed to accommodate changing user needs or to fit changing technology or changing operational circumstances. This mirrors the issues of problem-solution uncertainty in deciding how the product needs to change. Furthermore, such change can occur even as a product is being developed, with implications for reconsidering how aspects of uncertainty have already been resolved.

Recognizing the potential for change, building a product that is a fine solution to a properly understood problem is a good start but not the end. Changes can occur even as a product is being developed due to how uncertainties and tradeoffs in the problem or its solution are resolved or changes in the operational context in which the product is to be used. Over its expected useful life, changes in context will repeatedly occur in ways that will require changing the product so it continues to serve its intended purpose.

Modifying an existing product is substantially the same as dropping into the middle of initial development, recognizing that prior resolutions of some uncertainties are not easily modified. In using a product, customers often discover improvements in how their enterprise operates that would benefit from changes in the product. When a

customer's needs or operational circumstances change, motivating reconsideration of the problem-solution for the product, the same considerations of uncertainty and tradeoffs of the original development continue to apply. This will be an easier task if past uncertainties were resolved in expectation of these changes.

A particular concern in both creating and modifying a product is maintaining the ability to make further changes as customer needs and circumstances change. This requires judgement concerning the uncertainty regarding what types of changes are most likely. To be able to anticipate such changes, information needs to be provided, by problem and solution experts, as to what changes are more likely to be encountered during the expected life of the product.

Failing to identify likely potential changes is equivalent to assuming that nothing about a product will ever need to be changed. An informed but imperfect projection of potential change is generally more accurate than an (implicit) assumption of no change. Developing a product without considering likely future changes results in a product for which any change is unpredictably difficult and may be unnecessarily costly to make.

The degree of certainty concerning the nature of specific changes guides how much effort should be taken to prepare for those changes, recognizing that some potential changes may never occur or may take a form that differs from current expectations. Considering potential future change does not necessarily incur additional cost in that software developers often explore alternatives in order to resolve uncertainties and augment missing information (but then discard resulting information and insights that might save time when future changes do occur). More information about potential future changes serve to better inform developer explorations and allow them to better organize their work so as to localize likely changes.