

# Chapter 1

## The Software Business

To run a business and be successful, there are some basic requirements. We must understand the processes and products of the business, i.e., we must know the business. We must define our business needs and the means of achieving them, i.e., we must define our product and process characteristics and qualities. We must evaluate every aspect of our business, so we can understand our successes and our failures and where to make changes. We must define closed loop processes to feed back information for project control and accumulate knowledge for corporate learning. We must learn from our experiences, i.e., each project should provide information that allows us to do business better the next time. We must build competencies in the domain of our business by packaging our successful experiences for reuse, and then we must make these successful experiences or business specific competencies part of our business.

Thus, any successful business requires a combination of technical and managerial solutions. It requires a well defined set of product needs so we may articulate and satisfy our customers and assist the developer in accomplishing this task, and create competencies for future business. It requires a well defined set of processes to provide a means of accomplishing what needs to be accomplished, controlling the development, and improving the overall business. A successful business requires any number of closed loop processes that provide project feedback for control and real-time modification of the project process, and support for organizational learning across multiple projects. Key technologies necessary for supporting these requirements include the modeling, measurement, and reuse of the various processes, products, and other forms of knowledge relevant to our business.

But almost any business today involves the development or use of software. It is either the main aspect of the business, the value added to the product, or it is on the critical path to project success. It permeates every aspect of our lives. For example, a company like AT&T is not in the telephony business but in the software telephony business. Its systems are not analog but digital and the function provided to the customer is predominantly software; although it may make its profit from selling hardware in the form of switches, etc. A company like IBM is not just in the computer hardware business but in the computer software business. It provides its customers with an integrated software system of applications that communicate across various hardware platforms. Customers purchase their systems for the functions they deliver and these functions are mostly software intensive. If an organization does not recognize that it is in the software business and is not treating software as a business, i.e., investing in building core competencies, then it is not evolving and improving as a business, will not be competitive and may not be in business in the future.

But what is software? What is the nature of this artifact? What is the nature of the processes by which it is built? This book is based upon the view that software is developed, in the creative, intellectual sense,

rather than produced in the manufacturing sense. The software processes are development not production. This aspect of the discipline is probably the most important in determining how we do business. We have learned a great deal over the years about production processes. We have many models and techniques we can use to learn about the product and processes, feedback information, build models, measure effects, reuse knowledge, and improve production, based upon the works of Demming, Juran, Ishikawa, etc. But there has been little support for development processes in the past. This may be in part because manufacturing has made its profits on the efficiency and quality of its production rather than the efficiency of its development. Companies were willing to pay a little more for development because they did not develop many products from scratch, were willing to pay for the benefits of creativity, did not want to interfere with the creative juices, etc. However, software is all development, i.e., production is negligible, so we have to worry about improving the development business.

A second premise of this book is that the software discipline is evolutionary and experimental; i.e., software engineering is a laboratory science. There is a relationship between the various processes we apply and the product characteristics that result and this relationship cannot only be understood by analysis. We need to experiment and learn from our experiments whether such relationships hold, how they vary, what the limits of various technologies are, so we can know how to configure process to develop software better.

A third premise is that the technologies of the discipline are predominantly human based. This means we will never eliminate the thinking process. There are steps that can be automated but, the most important technologies, such as design and reading, must always be done by people. This creates problems for our experiments because there will always be variation in the results based upon human differences.

To compound the current problems, the discipline is immature in the sense that there is a lack of models that allow us to reason about the process and the product. This compounds the non-visible nature of software. In most cases, we do not even have intuitive models of the software product. Consider the implications of this statement. Suppose I asked you to build a car with a fifth wheel equidistant from the other four. You do not have to be an automotive engineer to know that this request is ridiculous. Because you have an intuitive model of the chassis of a car, you know that this extra wheel would add no useful function and it would be difficult to take an existing automobile design and make the modification. But if I asked you to add the functional equivalent of a fifth wheel to a software system, even if you were able to assess the benefits or lack thereof of the new function, it is not clear that you would be able to understand the limits of the existing design relative to the requirements for the modification.

Software is difficult. This statement is often not fully appreciated. It is difficult because it is the part of the system that is new, we least understand or requires the greatest amount of change. And there is a genuine lack of understanding of implications of change. The ability to make a change varies with the type of change and its relationship to the particular system design. And so the lack of models, especially tractable models of the various aspects of the business, make it even more difficult.

Sometimes, software is made even more difficult because we view it in a fuzzy way, i.e., we have fuzzy expectations. We accept poorly-defined methods, poorly-defined requirements, and poor products. This makes a difficult business even more difficult. But this last problem is our fault, not caused by the nature of software.

Premise number five, all software is not the same, just as all hardware is not the same. If I were building a toaster or a satellite, no one would assume because they are both hardware that the processes appropriate for developing them, the form of the requirements, the goals and important characteristics of the final product are the same. So why should the processes, goals, etc. for the micro code for the toaster and the flight dynamic software for the satellite be the same? Software isn't software, any more than hardware is hardware. That is, process is a variable, goals are variable, etc. So, before we decide how to build something we need to know something about the environment and the characteristics of the thing we are about to build, even if it is software.

Developing packaged models of our experiences for future use (reuse) requires additional resources in the form of organization, processes, people, etc. Building models, taking measurements, experimenting to find the most effective technologies, feeding back information for corporate learning cost time and money. This cannot be a by-product of software development. If these activities are not explicitly supported, independent of the product development, then they will not occur and we will not make quality improvements in the development process.

The concepts of quality improvement have permeated nearly every form of business. It is clear that the nineties will be the quality era for software and there is a growing need to develop or adapt quality improvement approaches to the software business. Thus we must understand software as an artifact and software as a business.

## **Software Quality Goals**

We will treat software quality as the focal point, assuming that other characteristics, such a productivity are also qualities and that an overall improvement in quality will provide the needed improvements in productivity. There are four major components to achieving software quality.

First, we must define the required software qualities required operationally, relative to the project and the organization. This statement implies that there are a variety of qualities, that they need to be defined in measurable terms, and that their definitions may vary with the particular project and organization. Because there are a large number of potential quality goals we need to bound and prioritize them.

Second, we must find criteria for selecting the appropriate methods and tools and tailoring them to the needs of the organization and the project. Since process is a variable, we must make our process selection appropriate to the qualities desired. We might also have to modify the processes to be most effective in the environment in which it is being applied. It implies we need to study and understand that environment as well as the conditions under which our set of available approaches are most effective.

Third, we must evaluate the quality of the process and product relative to the specific project goals. Even though we may have defined the qualities of interest so they are measurable, it is not always easy to evaluate quality because of the vast number of parameters, the inadequacy of our models, the effects of human behavior on the processes, and the complexity and error-proneness of the measurement process. Evaluation is not an easy task and requires a fair amount of analysis.

Fourth, we must organize quality assurance from planning through execution through evaluation, feedback and improvement. Quality is an integral part of the software development process, it must be planned for before the project begins, monitored during the execution of the project, and analyzed after the project is complete.

The current state of the practice for each of these major components varies dramatically from company to company, and within companies as well. However, in general, it is poor. For example, for many organizations, quality is simply defined as less errors reported by the customer, process for the entire organization is defined by a company-wide software development methodology standards document (which few projects actually follow), evaluation involves counting the number of customer trouble reports generated, and the quality assurance organization is the group that provides management with the trouble report data.

What's wrong with this view of quality? First, the definition of quality is too narrow. Surely, quality means more than error reports, what about functionality, service with regard to responding to the customer's need for an evolving system, ... In fact, reliability might be more important than a count of defects. A company-wide software development methodology standards document is typically too general a process definition and not sufficiently flexible. One process model won't work because not all projects are the same. Processes not only need to be defined and tailored for the characteristics of the project at hand, but also based upon what the developer's can do.

The view of evaluation of quality as the counting of trouble reports is passive rather than active. Based upon this definition, there is no way to learn how to do better and no evaluation of the methods or tools used. If I told you that you made 1000 defects, or 2 defects per thousand lines of code, what does this tell you? Without knowing your history for similar projects, it is not clear whether you can evaluate your "quality". Without knowing something about the types of defects you are making, where you are making them, and their causes, it is not clear whether you can ever get better. It is like telling a car manufacturer that he is making 5 defects per car but not what they are or where they are and what causes them. How is this auto manufacturer ever going to improve.

Lastly, the view is not quality oriented. Quality is the result not the driver. There is not way to learn how to make a quality product from a mere count of the error reports - we need to at least understand the kinds of errors we are making and how we can prevent or eliminate them and let that be the driver of the process.

In order to gain a deeper insight into quality activities, let's define the terms quality control and quality assurance. We define **quality control** as the act of directing, influencing, verifying, and correcting for ensuring the conformance of a specific product to a design or specification. We define **quality**

**assurance** as the act of leading, teaching, auditing the process by which the product is produced to provide confidence in the conformance of a specific product to a design or specification.

Given these definitions, we see that quality control is focused upon verifying and correcting the product, while quality assurance is focused upon, teaching and auditing the process. Thus the activities of quality control involve evaluating products and providing feedback to the project. Based upon this definition, quality control should be part of the project and interactive with the project. People requirements include knowledge of the processes, knowledge of the product requirements, and an understanding of the solutions.

Quality assurance activities, on the other hand, include defining and evaluating the processes, collecting data from quality control, and providing feedback to projects, the organization, and the quality assurance organization itself. A Quality Assurance organization should have an independent chain of command from the project but still needs to be interactive with the project. This latter is because our processes are still not well defined and there are numerous project influences we still do not understand. People requirements include a high level of expertise with respect to technology and management knowledge, and a deep understanding of the process and the product.

In the software arena, what quality control activities do we actually perform? For any project, consider the various types of activities we perform. Consider the activities as broken into constructive and analytic activities? Clearly constructive activities include specifying, designing, and coding. Analytic activities include reviewing, inspecting and testing. What % of the time is spent in each of these classes of activities? Reasonable public data implies that somewhere between 30% to 50% of the project effort is spent on the analytic activities. If we were to consider the analytic activities as quality control activities, this would imply we spend 30% to 50% of the development budget on quality control activities. This might sound shocking, since manufacturing tries to keep such activities to 5% of the budget. But this points out the difference between the human thought intensive activity of software development versus production. Do we know what the cost of quality control activities should be for should be for a development business such as software?

Besides, in software development, are the analytic activities really quality control activities? That depends upon how we define and differentiate these activities, from whose perspective they are performed, and what their goals are. For example, consider unit test. Is it a quality control activity or a development activity? If the coder is using a development paradigm that translates into a `code` a little, test a little approach, is the goal of unit test in this context quality control? In fact, who owns the activity in this context? Isn't it the unit tester? For each activity we need to define the goals and the perspective from which it is performed, i.e., the customer, manager, developer or corporation?

How do we enforce quality control and assurance? What controls do they have? Can a project be stopped from moving on to the next phase? Can part of a design be rejected because it doesn't pass standards? In general the answer is no. Why? Because we usually don't have sufficient information or data to enforce the decision. What we need are baselines, typically in the form of measurement data, based upon past project performance with regard to the passage criteria. We can use this to evaluate

whether a project should pass on to the next stage. We need the courage of our convictions, supported by data. We need real control so we can make the necessary decisions in trading off alternatives, e.g., schedule vs. quality. Over time, we need to evolve to something akin to what statistical quality control does for production.

In summary, to be able to deal with software quality issues, we must characterize, understand, evaluate, assess, predict, motivate, manage, and control the business. We must characterize in order to differentiate project environments and understand the current software process and product, and provide baselines for future assessments. In other words, we need to build descriptive models and baselines of the business.

We need to evaluate and assess the achievement of our quality goals and the impact of technology on products so we can understand where technology needs to be improved and tailored. That is, we need to compare models. We need to be able to predict so we can control the relationships between and among processes and products, i.e., we need to find patterns in our environment and build predictive models. We need to motivate, manage, and control various software qualities so we can provide quantitative motivation and guidelines to support what it is we are trying to accomplish, i.e., we need to build and package prescriptive models for reuse.

## **Improvement Approaches**

If our goal is to improve software quality, let us examine some of the variety of organizational frameworks proposed to improve quality for various businesses. We will discuss and compare six of them.

Plan-Do-Check-Act is a quality improvement process based upon a feedback cycle for optimizing a single process model/production line. The Experience Factory /Quality Improvement Paradigm involves continuous improvement through the experimentation, packaging and reuse of experiences based upon a business's needs. Total Quality Management represents a management approach to long term success through customer satisfaction based on the participation of all members of an organization. The SEI Capability Maturity Model is a staged process improvement based upon assessment with regard to a set of key process areas until you reach a level 5 which represents a continuous process improvement. Lean (Software) Development represents a principle supporting the concentration of the production on "value added" activities and the elimination or reduction of "not value added" activities. In what follows, we will try to define these concepts in a little more detail to distinguish and compare them.

**Plan-Do-Check-Act Cycle (PDCA).** The approach is based upon work by W. A. Shewart [Sh31] and was made popular and applied effectively to improve Japanese manufacturing after World War II by W. E. Demming [De86]. The goal of this approach is to optimize and improve a single process model / production line. It uses such techniques as feedback loops and statistical quality control to experiment with methods for improvement and build predictive models of the product.



Simply stated, if a Process (P) produces a Product (X) then the approach yields a family of processes and a series of versions of product X, produced by a series of modifications to the processes P; (each modification is meant to be an improvement to the previous version of the product itself or to the efficiency of the process, made through experimentation), yielding

$$P_0, P_1, P_2, \dots, P_n \text{ -----} > X_0, X_1, X_2, \dots, X_n$$

where  $P_i$ , represents an improvement over  $P_{i-1}$  and/or  $X_i$  has better quality than  $X_{i-1}$ .

The approach is defined as four basic steps:

**Plan:** Develop a plan for effective improvement, e.g., quality measurement criteria are set up as targets and methods for achieving the quality criteria are established.

**Do:** The plan is carried out, preferably on a small scale, i.e., the product is produced by complying with development standards and quality guidelines.

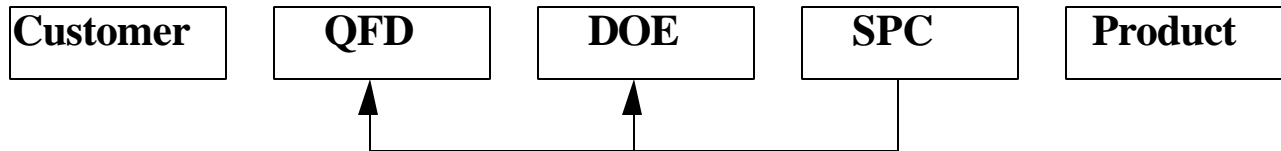
**Check:** The effects of the plan are observed; at each stage of development, the product is checked against the individual quality criteria set up in the Plan phase.

**Act:** The results are studied to determine what was learned and what can be predicted, e.g., corrective action is taken based upon problem reports.

Although the approach was developed for manufacturing, it has been used by several companies as an improvement paradigm for software development.

**Total Quality Management (TQM).** The term TQM was coined by the Naval Air Systems Command in 1985 to describe its Japanese style management approach to quality improvement. Many of the concepts were also generated by Feigenbaum [Fe90]. The goal of TQM is to generate institutional commitment to success through customer satisfaction. The approaches to achieving TQM vary greatly in practice. To provide some basis for comparison, we offer the approach being applied at Hughes [De92]. Hughes uses such techniques as Quality Function Deployment (QFD), design of experiments (DOE), and statistical process control (SPC), to improve the product through the process.

Identify needs --> ID Important items --> Make Improvements --> Hold Gains --> Provide Satisfaction



The approach, as applied by Hughes, has similar characteristics to the PDCA approach. If Process (P) --> Product (X) then the approach yields

$$P_0, P_1, P_2, \dots, P_n \text{ -----} > X_0, X_1, X_2, \dots, X_n$$

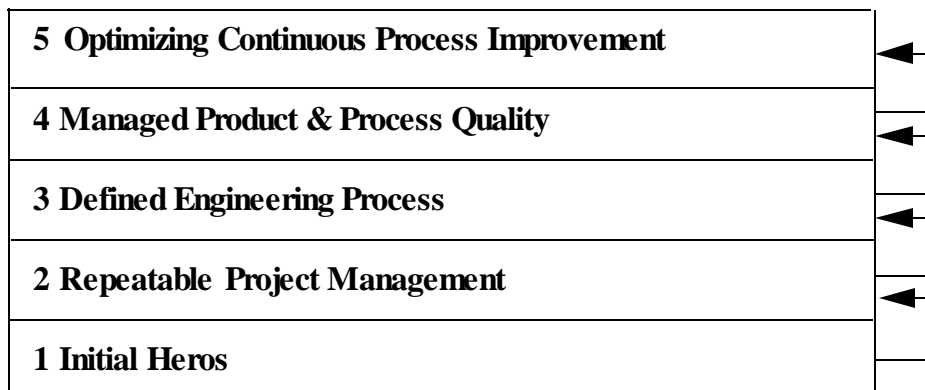
where  $P_i$ , represents an improvement over  $P_{i-1}$  because  $X_i$  provides better customer satisfaction than  $X_{i-1}$ .

1.

**SEI Capability Maturity Model.** The approach is based upon organizational and quality management maturity models developed by R Likert [Li67] and P. Crosby [Cr80], respectively. A software maturity model was developed by Ron Radice, et. al. [RaHaMuPh85] while he was at IBM. It was made popular by Watts Humphrey [Hu89] at the SEI. The goal of the approach is to achieve a level 5 maturity rating, i.e., continuous process improvement via defect prevention, technology innovation, and process change management.

As part of the approach, a 5 level process maturity model is defined. A maturity level is defined based on repeated assessment of an organization's capability in key process areas. Improvement is achieved by action plans for poorly assessed processes.

### Level Focus



Thus, if a Process (P) is level i then modify the process based upon the key processes of the model until the process model is at level i+1.

The SEI has developed a Process Improvement Cycle to support the movement through process levels. Basically is consists of the following activities:

**Initialize**

- Establish sponsorship
- Create vision and strategy
- Establish improvement structure

**For each Maturity level:**

- Characterize current practice in terms of key process areas
- Assessment recommendations
- Revise strategy (generate action plans and prioritize key process areas)

**For each key process area:**

- Establish process action teams
- Implement tactical plan, define processes, plan and execute pilot(s),  
plan and execute institutionalization
- Document and analyze lessons
- Revise organizational approach

It should be noted that this approach is fundamentally different from the previous two approaches in that it does not examine the product or any other business characterization. It assumes that there are essential or idealized processes and that adhering to these processes will generate good products. But the product is not assessed explicitly.

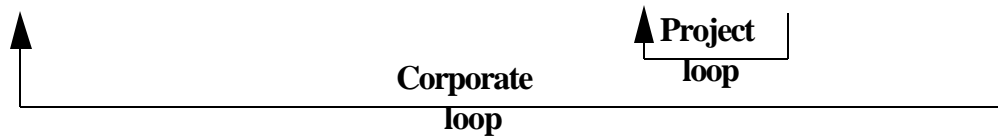
**Lean Software Development.** The approach is based upon Lean Enterprise Management, a philosophy that has been used to improve factory output. Womack, et. al. [WoJoRo90], have written a book on the application of lean enterprises in the automotive industry. The goal is to build software using the minimal set of activities needed, eliminating non essential steps, i.e., tailoring the process to the product needs. The approach uses such concepts as technology management, human centered management, decentral organization, quality management, supplier and customer integration, and internationalization/regionalization.

Given the characteristics for a product V, select the appropriate mix of sub-processes  $p_i$ ,  $q_j$ ,  $r_k$ , ... to satisfy the goals for V, yielding a minimal tailored process  $P_V$  which is composed of  $p_i$ ,  $q_j$ ,  $r_k$ , ..., yielding

Process ( $P_V$ ) -----> Product (V)

**Quality Improvement Paradigm.** This approach has evolved over 17 years based upon lessons learned in the SEL [Ba85a], [Ba89], [BaRo87], [BaRo88], [BaCaMc92]. Its goal is to build a continually improving organization based upon the evolving organizational goals and an assessment of its status relative to those goals. The approach uses internal assessment against the organizations own goals and status (rather than process areas) and such techniques as GQM, model building, and qualitative/quantitative analysis to improve the product and other aspects of the business through the process.

## Characterize - Set Goals - Choose Process - Execute - Analyze - Package



If Processes ( $P_X, Q_Y, R_Z, \dots$ )  $\rightarrow$  Products ( $X, Y, Z, \dots$ ) and we want to build  $V$ , then based upon an understanding of the relationship between  $P_X, Q_Y, R_Z, \dots$  and  $X, Y, Z, \dots$  and goals for  $V$  we select the appropriate mix of processes  $p_i, q_j, r_k, \dots$  to satisfy the goals for  $V$ , yielding a tailored Process ( $P_V$ )  $\rightarrow$  Product ( $V$ )

The Quality Improvement Paradigm consists of six steps:

**Characterize** the current project and its environment with respect to models and metrics.

**Set** the quantifiable **goals** for successful project performance and improvement.

**Choose** the appropriate **process** model and supporting methods and tools for this project.

**Execute** the processes, construct the products, collect and validate the prescribed data, and analyze it to provide real-time feedback for corrective action.

**Analyze** the data to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.

**Package** the experience in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and save it in an experience base to be reused on future projects.

The six steps of the Quality Improvement Paradigm can be combined in various ways to provide different views into the activities. First note that there are two feedback loops, a project feedback loop that takes place in the execution phase and an organizational feedback loop that takes place after a project is completed and changes the organization's understanding of the world between the packaging of what was learned from the last project and the characterization and baselining of the environment for the new project.

One high level organizational view of the paradigm is that we must **understand** (Characterize), **assess** (Set goals, Choose processes, Execute processes, Analyze data) and **package** (Package experience). Another view is to **plan** for a project (Characterize, Set goals, Choose processes), **develop** it (Execute processes), and then **learn** from the experience (Execute processes, Analyze data). There are two feedback loops explicitly specified in the paradigm, although there are numerous other loops visible at

lower levels of instantiation. These are the project feedback loop in step 4, executing the process, and the organization feedback loop from step 6, packaging new models, to step 1, characterizing the (new) environment.

Although most of this book is aimed at describing the QIP in more detail, here we discuss each of the six steps in a little more detail.

**Characterizing the Project and Environment.** Based upon a set of models of what we know about our business we need to classify the current project with respect to a variety of characteristics, distinguish the relevant project environment for the current project, and find the class of projects with similar characteristics and goals. This provides a context for goal definition, reusable experiences and objects, process selection, evaluation and comparison, and prediction. There are a large variety of project characteristics and environmental factors that need to be modeled, base lined. They include various people factors, such as the number of people, level of expertise, group organization, problem experience, process experience,...; problem factors, such as the application domain, newness to state of the art, susceptibility to change, problem constraints, ...; process factors, such as the life cycle model, methods, techniques, tools, programming language, other notations, ...; product factors, such as deliverables, system size, required qualities, e.g., reliability, portability, ..., and resource factors, such as target and development machines, calendar time, budget, existing software, ...

**Goal Setting and Measurement.** We need to establish goals for the processes and products. These goals should be measurable, driven by models of the business. There are a variety of mechanisms for defining measurable goals: Quality Function Deployment Approach (QFD), the Goal/Question/Metric Paradigm (GQM), and the Software Quality Metrics Approach (SQM).

Goals may be defined for any object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. For example, goals should be defined from a variety of points of view: user, customer, project manager, corporation, ...

**Choosing the Execution Model.** We need to be able to choose a generic process model appropriate to the specific context, environment, project characteristics, and goals established for the project at hand, as well as any goals established for the organization, e.g., experimentation with various processes or other experience objects. This implies we need to understand under what conditions various processes are effective. All processes must be defined to be measurable and defined in terms of the goals they must satisfy. The concept of defining goals for processes will be made clearer in later chapters.

Once we have chosen a particular process model, we must tailor it to the project and choose the specific integrated set of sub-processes, such as methods and techniques, appropriate for the project. In practice, the selection of processes is iterative with the redefinition of goals and even some environmental and project characteristics. It is important that the execution model resulting from these first three steps be integrated in terms of its context, goals and processes. The real goal is to have a set

of processes that will help the developer satisfy the goals set for the project in the given environment. This may sometimes require that we manipulate all three sets of variables to assure this consistency.

**Executing the Processes.** The development process must support the access and reuse packaged experience of all kinds. On the other hand it needs to be supported by various types of analysis, some done in close to real time for feedback for corrective action. To support this analysis, data needs to be collected from the project. But this data collection must be integrated into the processes, not be an add on, e.g. defect classification forms part of configuration control mechanism. Processes must be defined to be measurable to begin with, e.g., design inspections can be defined so that we keep track of the various activities, the effort expended in those activities, such as peer reading, and the effects of those activities, such as the number and types of defects found. This allows us to measure such things as domain conformance and assures that the processes are well defined and can evolve.

Support activities, such as data validation, education and training in the models and metrics and data forms are also important. Automated support necessary to support mechanical tasks and deal with the large amounts of data and information needed for analysis. It should be noted however, that most of the data cannot be automatically collected. This is because the more interesting and insightful data tends to require human response.

The kinds of data collected include: resource data such as, effort by activity, phase, type of personnel, computer time, and calendar time; change and defect data, such as changes and defects by various classification schemes, process data such as process definition, process conformance, and domain understanding; product data such as product characteristics, both logical, e.g., application domain, function, and physical, e.g. size, structure, and use and context information, e.g., who will be using the product and how will they be using it so we can build operational profiles.

**Analyzing the Data.** Based upon the goals, we interpret the data that has been collected. We can use this data to characterize and understand, so we can answer questions like "What project characteristics effect the choice of processes, methods and techniques?" and "Which phase is typically the greatest source of errors?" We can use the data to evaluate and analyze to answer questions like "What is the statement coverage of the acceptance test plan?" and "Does the Cleanroom Process reduce the rework effort?" We can use the data to predict and control to answer questions like "Given a set of project characteristics, what is the expected cost and reliability, based upon our history?" and "Given the specific characteristics of all the modules in the system, which modules are most likely to have defects so I can concentrate the reading or testing effort on them?" We can use the data to motivate and improve so we can answer questions such as "For what classes of errors is a particular technique most effective?" and "What are the best combination of approaches to use for a project with a continually evolving set of requirements based upon our organization's experience?"

**Packaging the models.** We need to define and refine models of all forms of experiences, e.g., resource models and baselines, change and defect baselines and models, product models and baselines, process definitions and models, method and technique evaluations, products and product parts, quality models, and lessons learned. These can appear in a variety of forms, e.g., we can have mathematical

models, informal relationships, histograms, algorithms and procedures, based upon our experience with their application in similar projects, so they may be reused in future projects.

How does the Quality Improvement Paradigm approach work in practice? You begin by putting the organization in place. This means collecting data to establish baselines, e.g., defects and resources, that are process and product independent and measuring your strengths and weaknesses to provide a business focus and goals for improvement, and establish product quality baselines. Using this information about your business, you select and experiment with methods and techniques to improve your processes based upon your product quality needs and evaluate your improvement based upon existing resource and defect baselines. You can define and tailor better and measurable processes, based upon the experience and knowledge gained within your own environment. You must measure for process conformance and domain understanding to make sure that your results are valid. You now begin to understand the relationship between some process characteristics and product qualities and are able to manipulate some processes to achieve those product characteristics. As you change your processes you will establish new baselines and learn where the next place for improvement might be.

### **Comparison of the frameworks**

The Quality Improvement Paradigm can be compared with the other frameworks from a variety of points of view. First, it is similar to the Plan-Do-Check-Act paradigm in that it is evolutionary, based upon feedback loops, and learns from experiments. It is different in the sense that the Plan-Do-Check-Act paradigm is based upon production, i.e., it attempts to optimize a single process model/production line. In development, we rarely replicate the same thing twice. In production, we can collect a sufficient set of data based upon continual repetition of the same process to develop quantitative models of the process that will allow us to evaluate and predict quite accurately the effects of the single process model. We can use statistical quality control approaches. This is not possible for development, i.e. we must learn from one process about another, so our models are less rigorous and more abstract. Development processes are also more human based. This again effects the building, use, and accuracy of the types of models we can build.

The QIP approach is compatible with TQM in that it can cover goals that are customer satisfaction driven and it is based upon the philosophy that quality is everyone's job. That is, everyone is part of the technology infusion process. Someone can be on the project team on one project and experimenting team on another. All the project personnel play the major role in the feedback mechanism. If they are not using the technology right it can be because they don't understand it, e.g., it wasn't taught right, it doesn't fit/interface with other project activities, it needs to be tailored, or it simply doesn't work. You need the user to tell you how to change it. This is consistent with the QIP philosophy that no method is packaged that hasn't been tried (applied, analyzed, tailored).

The QIP approach is most similar to the concepts of Lean Software Development in that it is based upon the ideas of tailoring a set of processes to meet particular problem/product under development. The goal is to generate an optimum set of processes, based upon models of the business and our experience about the relationship between process characteristics and product characteristics.

Comparing QIP to the SEI CMM approach, you pull yourself up from the top rather than pushing up from the bottom. At step 1 you start with a level 5 style organization even though you do not yet have

level 5 process capabilities. That is, you are driven by an understanding of your business, your product and process problems, your business goals, your experience with methods, etc. You learn from your business, not from an external model of process. You make process improvements based upon an understanding of the relationship between process and product in your organization. Technology infusion is motivated by the local problems, so people are more willing to try something new.

QIP is not incompatible with the SEI CMM model in that you can still use key process assessments to evaluate where you stand (along with your internal goals, needs, etc.). However, using the QIP, the chances are you will move up the maturity scale faster. You will have more experience early on operating within an improvement organization structure, and you can demonstrate product improvement benefits early.

In summary, the QIP approach provides for a separation of concerns/focus in differentiating between problem solving and experience modeling/packaging. It offers a support for learning and reuse and a means of formalizing and integrating management and development technologies. It allows for the generation of a tangible corporate asset: an experience base of software competencies. It offers a Lean Software Development approach compatible with TQM while providing a level 5 CMM organizational structure. It links focused research with development. Best of all you can start small, evolve and expand, e.g., focus on a homogeneous set of projects or a particular set of packages and build from there.

The software business requires understanding, continuous improvement, and the packaging of experience for reuse. There are certain concepts that have become understood with regard to software:

1. There are factors that create similarities and differences among projects. This means that one model for software development does not work in all situations.
2. There is a direct relationship between process and product. This means we must choose the right processes to create the desired product characteristics.
3. Measurement is necessary and must be based on the appropriate goals and models. That is, appropriate measurement provides visibility.
4. Evaluation and feedback are necessary for project control. This means we need a closed loop process for project control.
5. Software development follows an experimental paradigm. Thus, learning and feedback are natural activities for software development and maintenance.
6. Process, product, knowledge, and quality models need to be better defined and tailored. We need evolving definitions of the components of the software business.
7. Evaluation and feedback are necessary for learning. We need a closed loop for long range improvement, as well as for individual project control.
8. New technologies must be continually introduced. We need to experiment with technologies.
9. Reusing experience in the form of processes, products, and other forms of knowledge is essential for improvement. That is, reuse of knowledge is the basis of improvement.
10. Experience needs to be packaged. We must build competencies in software
11. Experiences must be evaluated for reuse potential. An analysis processes is required.
12. Software development and maintenance processes must support reuse of experience. We must say how and when to reuse.
13. A variety of experiences can be packaged. We can build process, product, resource, defect and quality models.
14. Experiences can be packaged in a variety of ways. We can use equations, histograms, algorithms as mechanisms for packaging experience.

15. Packaged experiences need to be integrated. We need an experience base of integrated information.

To address the business needs of software, we offer an evolutionary improvement paradigm tailored for the software business, the **Quality Improvement Paradigm**; a paradigm for establishing project and corporate goals and a mechanism for measuring against those goals, the **Goal/Question/Metric Paradigm**, and an organizational approach for building software competencies and supplying them to projects, the **Experience Factory**

The Quality Improvement Paradigm involves the following steps:

- (1) Set the quantifiable goals for successful project performance and improvement.
- (2) Characterize the current project and its environment.
- (3) Choose the appropriate process model and supporting methods and tools for this project.
- (4) Execute the processes, construct the products, collect and validate the prescribed data, and analyze it to provide real-time feedback for corrective action.
- (5) Analyze the data to evaluate current practices, determine problems, record findings, and make recommendations for future project improvements.
- (6) Package the experience in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and save it in an experience base so it is available for future projects.

The Goal/Question/Metric Paradigm is the mechanism used in step 2 of the Quality Improvement Paradigm for defining and evaluating a set of operational goals, using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organization.

The goals are defined in an operational, tractable way by refining them into a set of quantifiable questions that are used to extract the appropriate information from the models. The questions and models, in turn, define a specific set of metrics and data for collection and provide a framework for interpretation.

The Quality Improvement Paradigm is based upon the notion that improving the software process and product requires the continual accumulation of evaluated experiences (**learning**) in a form that can be effectively understood and modified (**experience models**) into a repository of integrated experience models (**experience base**) that can be accessed and modified to meet the needs of the current project (**reuse**). The paradigm implies the separation of project development (performed by the Project



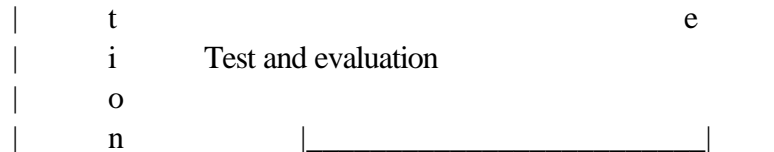


Figure 1. **The Standard Software Life Cycle**

With regard to quality issues, there are several problems associated with this superficial diagram. First, the process are not well defined or understood. What are the basic technologies? What education and training exists? How are they combined and managed? How are they adapted to the environment?

Second, major pieces are missing. How is planning represented? How is the user represented? How is quality assurance and control represented? How is improvement achieved?

Third, the process and product qualities are poorly articulated. What are the goals of the processes and products? How are the goals tracked?

Part of the problem lies in the fact that the basic terminology has not been well defined. For example, what do we mean by the terms process, technique, method, etc. and how are they related? These terms do not have standard definitions and various authors will use them in different ways. For the sake of consistency, we will adopt a set of definitions that we will try to keep consistent throughout these notes. We define a **technique** to be a basic algorithm, or set of steps to be followed in constructing or assessing the software. For example, code reading by stepwise abstraction is a technique for assessing the code. We define a **method** to be an organized approach based upon applying some technique. A method has associated with it a technique, as well as a set of guidelines about how and when to apply the technique, when to stop applying it, when the technique is appropriate and how we can evaluate it. For example, a method will have associated with it a set of entry and exit criteria and a set of management supports. For example, code inspection is a method, based upon some reading technique, which has a well defined set of entry and exit criteria as well as a set of management functions defined for how to use the method.

We define a **life cycle model** as an integrated set of methods that covers the entire life cycle of a software product. For example, an incremental development model using structured design, design inspections, etc.

By definition, the terms technique, method, and life cycle model are all abstractions. They do not take into account the environment in which they are to be applied. In that sense they are logical concepts We define **engineering** as the application and tailoring of techniques, methods and life cycle models to a specific problem, project and organization.

For example, there are a variety of software process models, each of which is useful under different circumstances. The Waterfall model (Figure 2) is basically a sequential process model where each of the major documents are developed in sequence, starting with the most abstract, i.e. the requirements document.

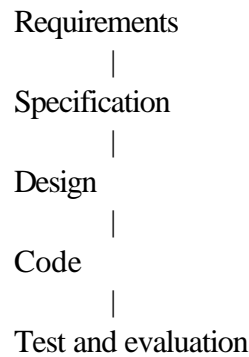


Figure 2. **Waterfall Model**

In practice, however, the process is typically iterative, cycling down, i.e., we may return to prior documents to make changes as we learn from the development of more detailed documents. The model results in a single version of the full system. The waterfall model is most efficient when the problem is well defined and the solution is well understood and, that is when there are not a lot of iterations through the cycle of documents. If the problem or the solution are not well-defined, other process models may be more effective.

The iterative enhancement model (Figure 3) is an incremental model that builds several versions of the system, each with more functionality. It starts with a simple initial implementation of a subset of the problem and iteratively enhances the existing version until the full system is implemented. At each step of the process, not only extensions but design modifications are made, based upon what we have learned about the problem and the solution. The process model results in several versions of the system, e.g., PO, P1, ..., PN. Iterative enhancement is effective when the problem or solution are not well understood, schedule for full function a risk, or the requirements changing. It allows the developer to learn through each cycle of development, improving each version until we reach the final version of the system.

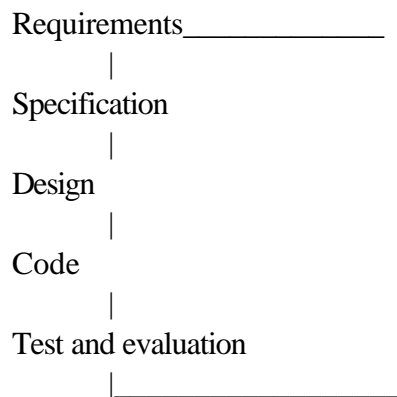


Figure 3. **Iterative enhancement Model**

Another effective process model is prototyping (Figure 4). It involves the development of an experimental version of some aspect of the system. The prototype is typically built in very high level language or using some modeling or simulation tools. It provides a better specification of the problem requirements and is effective when the user is unsure of the system needs, some aspect of the system unclear, or an experimental version is needed for analysis.

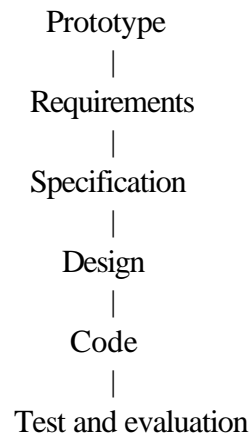


Figure 4. **Prototyping Model**

A life cycle model defines the order of application of the various methods and the development of the accompanying documents, but it does not define the particular methods or techniques that need to be applied. These must be determined based upon the needs of the problem, the environment and the ability of the methods to be integrated in the form required by the life cycle process model. At this level, there are a variety of methods and techniques available. We can differentiate them by various characteristics. The characteristics we will use here are: **input**: a definition of the objects that the method operates on and the state the environment must be in to be able to be applied; **output**: the objects produced by the method and a description of any results or changes in the environment from applying the method; **technique**: the basic technology performed on the input to produce the output, i.e., the algorithm or procedure followed to make the transformation from the input state to the output state; **formality**: the degree of rigor to which the technique is applied; **emphasis**: the set of focuses of the method and technique; **method definition**: the process model associated with applying the technique, including such issues as management and organizational structures; **perspective**: the viewpoint from which the technique and method are applied; **product quality**: the set of qualities of the output of interest; **process quality**: the set of qualities of the process of most interest.

By way of example, consider the following two approaches to software analysis: reading and testing. With regard to the above characteristics, we can define various reading approaches in terms of the characteristics given above. The input can be any document, e.g., requirements, specification, design, code, test plan. One of the outputs is a set of anomalies discovered by the reader. Techniques for reading include sequential, path analysis, stepwise abstraction, etc. The level of formality can be defined as reading, correctness demonstrations, proofs of correctness, etc. The emphasis could be fault detection, traceability, performance, etc. The method definition could be Walk-throughs, inspections,

reviews, etc. The perspective might include, reading from the point of view of the user, designer, tester, maintainer, etc. Product qualities include correctness, reliability, efficiency, portability, etc. Process qualities include adherence to the method, integration into the process model, etc. Reading can be done from the quality view of assurance or control.

We can also use the characterization scheme to define various approaches to testing. The input can be the system, subsystem, feature, module, etc. One of the outputs is the test results. The techniques available are any of the classes of structural testing, functional testing, error-based testing, statistical testing, etc.. The level of formality can be defined as full adherence or partial adherence to the rules defined for the test technique, e.g. 90% statement coverage. The emphasis can be fault detection, new features, reliability, performance, etc. The method definition is typically what is specified in the test plan, e.g., mapping of technique onto class of requirement. The perspective might be defined as testing the system from the point of view of various classes of customers and hardware configurations, or from the viewpoint of the next test phase. Product qualities include reliability, efficiency, etc. Process qualities again include adherence to method, integration into process model, etc. Testing can also be done from the quality view of assurance or control.

As an example of a constructive approach to software development, consider stepwise refinement. Here the input could be the requirements or specification document, higher level design (possibly mix of old and new), etc. An output is some level of design document. Available techniques include function/state machine decomposition, the axiomatic approach, the Jackson approach, etc. The level of formality can be represented via. the notation, e.g., tree charts, PDL, functions, state machines, predicate calculus, etc. The emphasis can be information hiding, strength and coupling, low complexity, etc. The method definition might involve top down design, with various builds, various review, milestone, entry and exit criteria, etc. The perspective could be to satisfy any of the consumers of the output document, e.g., the coder, tester, maintainer. Product qualities include correctness, efficiency, portability, reusability. Again, process qualities include adherence to method, integration into process model, etc.

It is clear from this discussion that process is a variable that must be tailored to fit the individual situation. This is part of the engineering process. The more we know about the problem and the environment, the better we are able to tailor the various processes to the specific project. For example, if we know that the problem definition and solution domain are not well understood, then we are better off choosing a life cycle model like iterative enhancement. If we know that 40% of our errors are errors of omission, i.e., that we tend to leave things out of the solution, then we might wish to emphasize reading for traceability. In each case we must choose and tailor an appropriate generic process model, choose and tailor an integrated set of methods in the context of the process model, choose and tailor the appropriate techniques for each method. It is important to recognize that choosing and tailoring is always done in the context of the environment, project characteristics, and goals established for the products and processes.

In order to aid in the selection of the appropriate processes, we need to establish goals for the software development process from a variety of perspectives. These goals need to be articulated and evaluated.

There are a large number of goals, for example, from the project management perspective, we want to develop a set of documents that all represent the same system; from the user's perspective, we want to develop a system that satisfies the user's needs with respect to functionality, quality, cost, etc., from the corporate perspective, we want to improve the organization's ability to develop quality systems productively and profitably.

Goals help define the process model, methods and techniques. For example, what are the goals of the test activity? Is it to assess quality or to find failures? Which test activities we select upon the answer to that question. If our goal is to assess quality, then we want tests based upon user operational scenarios, a statistically-based testing technique, and we can use reliability modeling for assessment. If on the other hand, our goal is to find failures, then we should probably test a function at a time, going from the general to the specific, and in that case, reliability models are not appropriate.

Process goals are dependent upon the environment. For example, what are the goals of the requirements activity? We might respond simply that it is to characterize the user's needs in order to specify a system that satisfies them. However, the selection of activities depends upon the environment, e.g., contract vs. general product. What does this say about user input? For example, how do we create customer/user scenarios/models in a contract versus a general product environment? In each case how do we get the user involved in requirements definition (e.g., prototype screens), and test plans?

It is clear that there is a great deal of complexity and flexibility involved in the selection, tailoring and application of process to a particular project environment. We need flexible definitions and goals for processes. We need experimentation with and analysis of the various technologies to understand their strengths and weaknesses, when and where they are appropriate to apply, and how to tailor them to a specific project. Most times this experimentation can be done on live projects. Occasionally, we may wish to experiment off-line, when there is a real risk in applying the technology or we need a more controlled environment to evaluate the effects of the technology. We need extensive education, training and skill development in the application, methods, techniques, tools and software development environment.

Improving the software process and product requires that we characterize, understand, evaluate, predict, motivate, manage and control the process and product qualities. Characterizing and understanding permit us to differentiate project environments, understand the current software process and product, and provide baselines for future assessment. It requires that we build descriptive models that allow us to recognize what we are doing. Evaluation permits us to assess the achievement of our quality goals, assess the impact of technology on products, and understand where technology needs to be improved, tailored. Predictive models allow us to control and understand the relationships between and among processes and products. We can define prescriptive models that allow us to motivate, manage, and control the process by providing quantitative motivation and guidelines that support what it is we are trying to accomplish.

Applying the Quality Improvement Paradigm to the improvement of process can be accomplished by the following steps:

1. Characterizing the current project environment. This involves finding candidate process models from similar projects by searching the experience base for projects with similar characteristics.
2. Setting up the goals/questions/metrics. This must be based upon establishing goals that characterize the effects of the process and evaluating whether it provides improvement over executions of previous processes..
3. Choosing the appropriate process model, methods and tools. This involves combining, tailoring or redefining candidate process models based on the goals and characteristics of this project.
4. Executing the current project according to the process model. This involves recording data and qualitative experiences based upon the goals for learning and improvement, and providing feedback in real-time for corrective action by modifying the definition of the process where appropriate.
5. Analyzing the results. This involves the evaluating the process in terms of the goals and the context of the current project, determining problems, and writing lessons learned with regard to successes, failures and new insights into applying the process model.
6. Formalizing the results. This involves modifying the process model and any baselines, making the experience available for the next project in the form of a new process model that can be incorporated into the experience base.

The Quality Improvement Paradigm provides a high level view of the software process. It is an organizational process model and as such requires a top level organizational commitment. The first step is revolutionary. It requires us to think differently. We need to think of software development as an experimental activity from which we must learn and package our knowledge in the form of models. Fortunately, this can be achieved in small steps, i.e. we can start with one or two projects and add projects as we go along. Later steps are evolutionary, they are achieved by applying the steps of this high level process.