A Value-Chain Analysis of Software Productivity Components

Barry W. Boehm, TRW Inc.
Phillip N. Papaccio, TRW Inc.

Summary

This paper summarizes a recent value-chain analysis of software productivity components at TRW. It explains the various value chain components and percentages, assesses their implications for improving software productivity, and elaborates on some further data analysis performed to address one of the major value chain components: rework costs.

1. The Software Product Value Chain

The value chain, developed by Porter and his associates at the Harvard Business School [Porter, 1980; Porter, 1985], is a useful method of understanding and controlling the costs involved in a wide variety of organizational enterprises. It identifies a canonical set of cost sources or value activities, representing the basic activities an organization can choose from to create added value for its products. Figure 1 shows a value chain for software development representative of experience at TRW. Definitions and explanations of the component value activities are given below. These are divided into what [Porter, 1985] calls primary activities (inbound logistics, outbound logistics, marketing and sales, service, and operations) and support activities (infrastructure, human resource management, technology development, and procurement).

Primary Activities

Inbound logistics covers activities associated with receiving, storing, and disseminating inputs to the products. This can be quite large for a manufacturer of, say, automobiles; for software it consumes less than 1% of the development outlay. (For software, the related support activity of procurement is also included here).

Outbound logistics covers activities concerned with collecting, storing, and physically distributing the product to buyers. Again, for software, this consumes less than 1% of the total.
Figure 1. Software Development Value Chain
Marketing and sales covers activities associated with providing a means by which buyers can purchase the product and inducing them to do so. A 5% figure is typical of government contract software organizations. Software product houses would typically have a higher figure; internal applications-programming shops would typically have a lower figure.

Service covers activities associated with providing service to enhance or maintain the value of the product. For software, this comprises the activities generally called software maintenance or evolution. For simplicity, Figure 1 avoids including a service cost component in the development value chain; a life-cycle value chain is presented and discussed as Figure 2 below.

Operations covers activities associated with transforming inputs into the final product form. For software, operations typically involves roughly four-fifths of the total development outlay.

In such a case, the value-chain analysis involves breaking up a large component into constituent activities. Figure 1 shows such a breakup into management (7%), quality assurance and configuration management (5%), and the distribution of technical effort among the various development phases. This phase breakdown also covers the cost sources due to rework. Thus, for example, of the 20% overall cost of the technical effort during the integration and test phase, 13% is devoted to activities required to rework deficiencies in or reorientations of the requirements, design, code, or documentation; the other 7% represents the amount of effort required to run tests, perform integration functions, and complete documentation even if no problems were detected in the process.

Support Activities

Infrastructure covers such activities as the organization's general management planning, finance, accounting, legal, and government affairs. The 8% figure is typical of most organizations.

Human resource management covers activities involved in recruiting, hiring, training, development, and compensation of all types of personnel. Given the labor-intensive and technology-intensive nature of software development, the 3% figure indicated here is a less-than-optimal investment.

Technology development covers activities devoted to creating or tailoring new technology to improve the organizations products or processes. The 3% investment figure here is higher than many software organizations, but still less than
Figure 2. Software Life-Cycle Value Chain
optimal as an investment to improve software productivity and quality.

Margin and Service

Margin in the value chain is the difference between the value of the resulting product and the collective cost of performing the value activities. As this difference varies widely among software products, it is not quantitatively defined in Figure 1. As discussed above, service is best quantified as a software life-cycle value chain as shown as Figure 2, with roughly 70% of the value activity devoted to service or evolution-related activity. However, since the component activities involved during evolution do not differ markedly from those which go on during software development, we will continue to focus on Figure 1 as a source of insights into understanding and controlling software costs.

Software Development Value Chain Implications

The primary implication of the software development value chain is that the "Operations" component is the key to significant improvements. Not only is it the major source of software costs, but also most of the remaining components such as "Human Resources" will scale down in a manner proportional to the scaling down of Operations cost.

Another major characteristic of the value chain is that virtually all of the components are still highly labor-intensive. Thus, there are significant opportunities in providing automated aids to make these activities more efficient and capital-intensive. Further, it implies that human-resource and management activities have much higher leverage than their 3% and 7% investment levels indicate.

The breakdown of the Operations component indicates that the leading strategies for cost savings in software development involve:

- **Making individual steps more efficient**, via such capabilities as automated aids to software requirements analysis or testing.

- **Eliminating steps**, via such capabilities as automatic programming or automatic quality assurance.

- **Eliminating rework**, via early error detection, or via such capabilities as rapid prototyping to avoid later requirements rework.
In addition, further major cost savings can be achieved by reducing the total number of elementary Operations steps, by developing products requiring the creation of fewer lines of code. This has the effect of reducing the overall size of the Value Chain itself. This source of savings breaks down into two primary options:

- **Building simpler products**, via more insightful front-end activities such as prototyping or risk management.

- **Reusing software components**, via such capabilities as fourth-generation languages or component libraries.

### 2. The Software Productivity Improvement Opportunity Tree

This breakdown of the major sources of software cost savings leads to the *Software Productivity Improvement Opportunity Tree* shown in Figure 3. This hierarchical breakdown helps us to understand how to fit the various attractive productivity options into an overall integrated software productivity improvement strategy.

Further discussions of the various productivity options are provided in [Boehm, 1986a]. As one example involving further data analysis, we studied the distribution of rework costs on a sample of 1378 problem reports on two large TRW software projects. These studies indicated that rework instances tend to follow a Pareto distribution: 80% of the rework costs typically result from 20% of the problems. Figure 4 shows some typical distributions of this nature from recent TRW software projects; similar trends have been indicated in [Rubey et al, 1975], [Formica, 1978], and [Basili-Weiss, 1981]. The major implication of this distribution is that software verification and validation activities should focus on identifying and eliminating the specific *high-risk* problems to be encountered by a software project, rather than spreading their available early-problem-elimination effort uniformly across trivial and severe problems. Even more strongly, this implies that a *risk-driven* approach to the software life-cycle such as the spiral model [Boehm, 1986] is preferable to a more *document-driven* model such as the traditional waterfall model.
Figure 3. Productivity Improvement Opportunity Tree

MAKE PEOPLE MORE EFFECTIVE
- INCENTIVES, STAFFING, TRAINING
- FACILITIES
- MANAGEMENT

MAKE STEPS MORE EFFICIENT
- SOFTWARE TOOLS, ENVIRONMENTS
- WORKSTATIONS
- OFFICE AUTOMATION

ELIMINATE STEPS
- AUTOMATED DOCUMENTATION, QUALITY ASSURANCE
- AUTOMATED PROGRAMMING

ELIMINATE REWORK
- KNOWLEDGE-BASED SOFTWARE ASSISTANT
- INFORMATION HIDING, MODERN PROGRAMMING PRACTICES
- SOFTWARE COMPUTER AIDED DESIGN
- FRONT-END LANGUAGES
- INCREMENTAL DEVELOPMENT

BUILD SIMpler PRODUCTS
- RAPID PROTOTYPING
- PROCESS MODELS

REUSE COMPONENTS
- COMPONENT LIBRARIES
- APPLICATION GENERATORS
- FOURTH-GENERATION LANGUAGES
Figure 4. Rework Costs are Concentrated in a Few High-Risk Items


