

1. Software Quality Models and Philosophies

1.1. Introduction

The purpose of this chapter is to provide an overview to different quality models. It will also discuss what quality is by presenting a number of high-profile quality gurus together with their thoughts on quality (which in some cases actually results in a more or less formal quality model). The chapter is structured as follows: To be able to discuss the topic of quality and quality models, we as many others, must first embark on trying to define the concept of quality. Section 1.2 provides some initial definitions and scope on how to approach this elusive and subjective word. Section 1.3 provides a wider perspective on quality by presenting a more philosophical management view on what quality can mean. Section 1.4 continues to discuss quality through a model specific overview of several of the most popular quality models and quality structures of today. The chapter is concluded in Section 1.5 with a discussion about presented structures of quality, as well as some concluding personal reflections.

1.2. What is Quality

To understand the landscape of software quality it is central to answer the so often asked question: *what is quality?* Once the concept of quality is understood it is easier to understand the different structures of quality available on the market. As follows, and before we embark into the quality quagmire, we will spend some time to sort out **the** question: *what is quality*. As many prominent authors and researchers have provided an answer to that question, we do not have the ambition of introducing yet another answer but we will rather answer the question by studying the answers that some of the more prominent gurus of the quality management community have provided. By learning from those gone down this path before us we can identify that there are two major camps when discussing the meaning and definition of (software) quality [1]:

- 1) **Conformance to specification:** Quality that is defined as a matter of products and services whose measurable characteristics satisfy a fixed specification – that is, conformance to an in beforehand defined specification.
- 2) **Meeting customer needs:** Quality that is identified independent of any measurable characteristics. That is, quality is defined as the products or services capability to meet customer expectations – explicit or not.

1.3. Quality Management Philosophies

One of the two perspectives chosen to survey the area of quality structures within this technical paper is by means of quality management gurus. This perspective provides a qualitative and flexible [2] alternative on how to view quality structures. As will be discussed in Section 1.5, quality management philosophies can sometimes be a good alternative to the more formalized quality models discussed in Section 1.4.

1.3.1. Quality according to Crosby

In the book “Quality is free: the art of making quality certain” [3], Philip B. Crosby writes:

The first erroneous assumption is that quality means goodness, or luxury or shininess. The word “quality” is often used to signify the relative worth of something in such phrases as “good quality”, “bad quality” and “quality of life” - which means different things to each and every person. As follows quality must be defined as “conformance to requirements” if we are to manage it. Consequently, the nonconformance detected is the absence of quality, quality problems become nonconformance problems, and quality becomes definable.

Crosby is a clear “conformance to specification” quality definition adherer. However, he also focuses on trying to understand the full array of expectations that a customer has on quality by expanding the, of today’s measure, somewhat narrow production perspective on quality with a supplementary external perspective. Crosby also emphasizes that it is important to clearly define quality to be able to measure and manage the concept. Crosby summarizes his perspective on quality in fourteen steps but is built around four fundamental “absolutes” of quality management:

- 1) Quality is defined as conformance to requirements, not as “goodness” or “elegance”
- 2) The system for causing quality is prevention, not appraisal. That is, the quality system for suppliers attempting to meet customers' requirements is to do it right the first time. As follows, Crosby is a strong advocate of prevention, not inspection. In a Crosby oriented quality organization everyone has the responsibility for his or her own work. There is no one else to catch errors.
- 3) The performance standard must be Zero Defects, not "that's close enough". Crosby has advocated the notion that zero errors can and should be a target.
- 4) The measurement of quality is the cost of quality. Costs of imperfection, if corrected, have an immediate beneficial effect on bottom-line performance as well as on customer relations. To that extent, investments should be made in training and other supporting activities to eliminate errors and recover the costs of waste.

1.3.2. Quality according to Deming

Walter Edwards Deming's “Out of the crisis: quality, productivity and competitive position” [4], states:

The problem inherent in attempts to define the quality of a product, almost any product, where stated by the master Walter A. Shewhart. The difficulty in defining quality is to translate future needs of the user into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price that the user will pay. This is not easy, and as soon as one feels fairly successful in the endeavor, he finds that the needs of the consumer have changed, competitors have moved in etc.

One of Deming's strongest points is that quality must be defined in terms of customer satisfaction – which is a much wider concept than the “conformance to specification” definition of quality (i.e. “meeting customer needs” perspective). Deming means that quality should be defined only in terms of the agent – the judge of quality.

Deming's philosophy of quality stresses that meeting and exceeding the customers' requirements is the task that everyone within an organization needs to accomplish. Furthermore, the management system has to enable everyone to be responsible for the quality of his output to his internal customers. To implement his perspective on quality Deming introduced his 14 Points for Management in order to help people understand and implement the necessary transformation:

- 1) **Create constancy of purpose for improvement of product and service:** A better way to make money is to stay in business and provide jobs through innovation, research, constant improvement and maintenance.
- 2) **Adopt the new philosophy:** For the new economic age, management needs to take leadership for change into a *learning organization*. Furthermore, we need a new belief in which mistakes and negativism are unacceptable.
- 3) **Cease dependence on mass inspection:** Eliminate the need for mass inspection by building quality into the product.
- 4) **End awarding business on price:** Instead, aim at minimum total cost and move towards single suppliers.
- 5) **Improve constantly and forever the system of production and service:** Improvement is not a one-time effort. Management is obligated to continually look for ways to reduce waste and improve quality.
- 6) **Institute training:** Too often, workers have learned their job from other workers who have never been trained properly. They are forced to follow unintelligible instructions. They can't do their jobs well because no one tells them how to do so.
- 7) **Institute leadership:** The job of a supervisor is not to tell people what to do nor to punish them, but to lead. Leading consists of helping people to do a better job and to learn by objective methods.
- 8) **Drive out fear:** Many employees are afraid to ask questions or to take a position, even when they do not understand what their job is or what is right or wrong. To assure better quality and productivity, it is necessary that people feel secure. "The only stupid question is the one that is not asked."
- 9) **Break down barriers between departments:** Often a company's departments or units are competing with each other or have goals that conflict. They do not work as a team; therefore they cannot solve or foresee problems. Even worse, one department's goal may cause trouble for another.
- 10) **Eliminate slogans, exhortations and numerical targets:** These never help anybody do a good job. Let workers formulate their own slogans. Then they will be committed to the contents.
- 11) **Eliminate numerical quotas or work standards:** Quotas take into account only numbers, not quality or methods. They are usually a guarantee of inefficiency and high cost. A person, in order to hold a job, will try to meet a quota at any cost, including doing damage to his company.
- 12) **Remove barriers to taking pride in workmanship:** People are eager to do a good job and distressed when they cannot.
- 13) **Institute a vigorous programme of education:** Both management and the work force will have to be educated in the new knowledge and understanding, including teamwork and statistical techniques.
- 14) **Take action to accomplish the transformation:** It will require a special top management team with a plan of action to carry out the quality mission. A critical mass of people in the company must understand the 14 points.

1.3.3. Quality according to Feigenbaum

The name Feigenbaum and the term total quality control are virtually synonymous due to his profound influence on the concept of total quality control (but also due to being the originator of the concept). In “Total quality control” [5] Armand Vallin Feigenbaum explains his perspective on quality through the following text:

Quality is a customer determination, not an engineer's determination, not a marketing determination, nor a general management determination. It is based on upon the customer's actual experience with the product or service, measured against his or her requirements – stated or unstated, conscious or merely sensed, technically operational or entirely subjective – and always representing a moving target in a competitive market.

Product and service quality can be defined as: The total composite product and service characteristics of marketing, engineering, manufacture and maintenance though witch the product and service in use will meet the expectations of the customer.

Feigenbaum's definition of quality is unmistakable a “meeting customer needs” definition of quality. In fact, he goes very wide in his quality definition by emphasizing the importance of satisfying the customer in both actual and expected needs. Feigenbaum essentially points out that quality must be defined in terms of customer satisfaction, that quality is multidimensional (it must be comprehensively defined), and as the needs are changing quality is a dynamic concept in constant change as well. It is clear that Feigenbaum's definition of quality not only encompasses the management of product and services but also of the customer and the customer's expectations.

1.3.4. Quality according to Ishikawa

Kaoru Ishikawa writes the following in his book “What is quality control? The Japanese Way” [6]:

We engage in quality control in order to manufacture products with the quality which can satisfy the requirements of consumers. The mere fact of meeting national standards or specifications is not the answer, it is simply insufficient. International standards established by the International Organization for Standardization (ISO) or the International Electrotechnical Commission (IEC) are not perfect. They contain many shortcomings. Consumers may not be satisfied with a product which meets these standards. We must also keep in mind that consumer requirements change from year to year and even frequently updated standards cannot keep the pace with consumer requirements. How one interprets the term “quality” is important. Narrowly interpreted, quality means quality of products. Broadly interpreted, quality means quality of product, service, information, processes, people, systems etc. etc.

Ishikawa's perspective on quality is a “meeting customer needs” definition as he strongly couples the level of quality to every changing customer expectations. He further means that quality is a dynamic concept as the needs, the requirements and the expectations of a customer continuously change. As follows, quality must be defined comprehensively and dynamically. Ishikawa also includes that price as an attribute on quality – that is, an overpriced product can neither gain customer satisfaction and as follows not high quality.

1.3.5. Quality according to Juran

In “Jurans's Quality Control Handbook” [7] Joseph M. Juran provides two meanings to quality:

The word quality has multiple meanings. Two of those meanings dominate the use of the word: 1) Quality consists of those product features which meet the need of customers and thereby provide product satisfaction. 2) Quality consists of freedom from deficiencies. Nevertheless, in a handbook such as this it is most convenient to standardize on a short definition of the word quality as “fitness for use”

Juran takes a somewhat different road to defining quality than the other gurus previously mentioned. His point is that we cannot use the word quality in terms of satisfying customer expectations or specifications as it is very hard to achieve this. Instead he defines quality as “fitness for use” – which indicates references to requirements and products characteristics. As follows Juran's definition could be interpreted as a “conformance to specification” definition more than a “meeting customer needs” definition. Juran proposes three fundamental managerial processes for the task of managing quality. The three elements of the Juran Trilogy are:

- Quality planning: A process that identifies the customers, their requirements, the product and service features that customers expect, and the processes that will deliver those products and services with the correct attributes and then facilitates the transfer of this knowledge to the producing arm of the organization.
- Quality control: A process in which the product is examined and evaluated against the original requirements expressed by the customer. Problems detected are then corrected.
- Quality improvement: A process in which the sustaining mechanisms are put in place so that quality can be achieved on a continuous basis. This includes allocating resources, assigning people to pursue quality projects, training those involved in pursuing projects, and in general establishing a permanent structure to pursue quality and maintain the gains secured.

1.3.6. Quality according to Shewhart

As referred to by W.E. Deming, “the master”, Walter A. Shewhart defines quality in “Economic control of quality of manufactured product” [8] as follows:

There are two common aspects of quality: One of them has to do with the consideration of the quality of a thing as an objective reality independent of the existence of man. The other has to do with what we think, feel or sense as a result of the objective reality. In other word, there is a subjective side of quality.

Although Shewhart’s definition of quality is from 1920s, it is still considered by many to be the best and most superior. Shewhart talks about both an objective and subjective side of quality which nicely fits into both “conformance to specification” and “meeting customer needs” definitions.

1.4. Quality Models

In the previous section we presented some quality management gurus as well as their ideas and views on quality – primarily because this is a used and appreciated approach for dealing with quality issues in software developing organizations. Whereas the quality management philosophies presented represent a more flexible and qualitative view on quality, this section will present a more fixed and quantitative [2] quality structure view.

1.4.1. McCall’s Quality Model (1977)

One of the more renown predecessors of today’s quality models is the quality model presented by Jim McCall et al. [9-11] (also known as the General Electrics Model of 1977). This model, as well as other contemporary models, originates from the US military (it was developed for the US Air Force, promoted within DoD) and is primarily aimed towards the system developers and the system development process. In his quality model McCall attempts to bridge the gap between users and developers by focusing on a number of software quality factor that reflect both the users’ views and the developers’ priorities.

The McCall quality model has, as shown in Figure 1, three major perspectives for defining and identifying the quality of a software product: product revision (ability to undergo changes), product transition (adaptability to new environments) and product operations (its operation characteristics).

Product revision includes maintainability (the effort required to locate and fix a fault in the program within its operating environment), flexibility (the ease of making changes required by changes in the operating environment) and testability (the ease of testing the program, to ensure that it is error-free and meets its specification).

Product transition is all about portability (the effort required to transfer a program from one environment to another), reusability (the ease of reusing software in a different context) and interoperability (the effort required to couple the system to another system).

Quality of product operations depends on correctness (the extent to which a program fulfils its specification), reliability (the systems ability not to fail), efficiency (further categorized into execution efficiency and storage efficiency and generally meaning the use of resources, e.g. processor time, storage), integrity (the protection of the program from unauthorized access) and usability (the ease of the software).

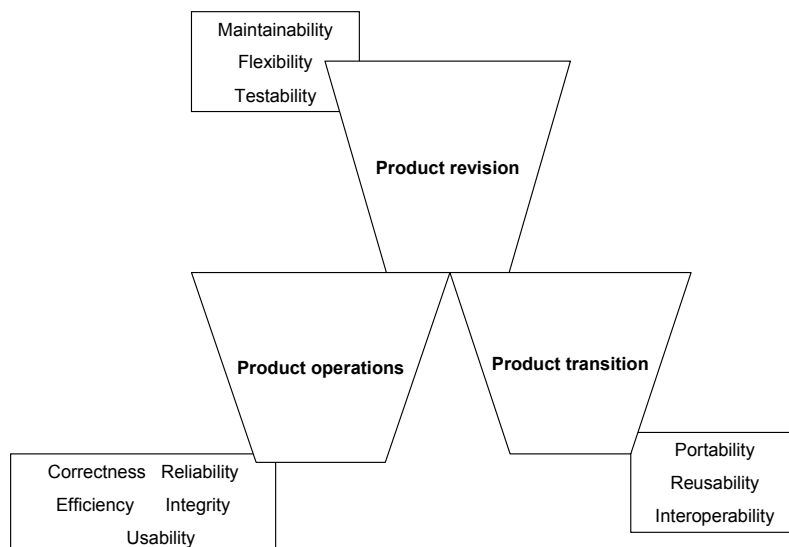


Figure 1: The McCall quality model (a.k.a. McCall’s Triangle of Quality) organized around three types of quality characteristics.

The model furthermore details the three types of quality characteristics (major perspectives) in a hierarchy of factors, criteria and metrics:

- 11 Factors (To specify): They describe the external view of the software, as viewed by the users.
- 23 quality criteria (To build): They describe the internal view of the software, as seen by the developer.
- Metrics (To control): They are defined and used to provide a scale and method for measurement.

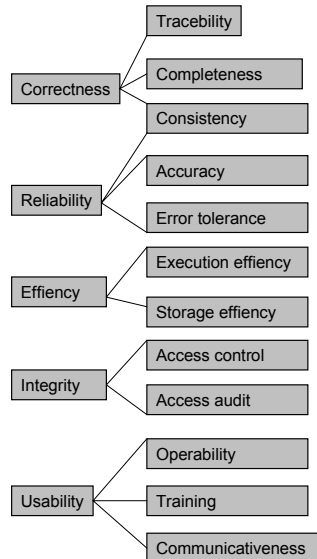


Figure 2: McCall's Quality Model illustrated through a hierarchy of 11 quality factors (on the left hand side of the figure) related to 23 quality criteria (on the right hand side of the figure).

The quality factors describe different types of system behavioral characteristics, and the quality criterions are attributes to one or more of the quality factors. The quality metric, in turn, aims to capture some of the aspects of a quality criterion.

The idea behind McCall's Quality Model is that the quality factors synthesized should provide a complete software quality picture [11]. The actual quality metric is achieved by answering yes and no questions that then are put in relation to each other. That is, if answering equally amount of "yes" and "no" on the questions measuring a quality criteria you will achieve 50% on that quality criteria¹. The metrics can then be synthesized per quality criteria, per quality factor, or if relevant per product or service.

¹ The critique of this approach is that the quality judgment is subjectively measured based on the judgment on the person(s) answering the questions.

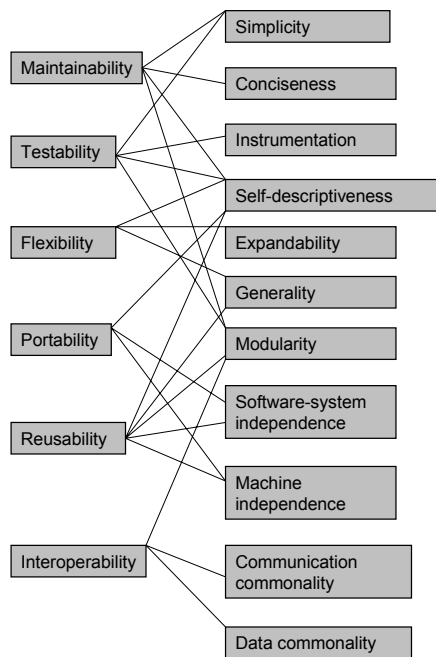


Figure 3: McCall's Quality Model (cont.) illustrated through a hierarchy of 11 quality factors (on the left hand side of the figure) related to 23 quality criteria (on the right hand side of the figure).

1.4.2. Boehm's Quality Model (1978)

The second of the basic and founding predecessors of today's quality models is the quality model presented by Barry W. Boehm [12;13]. Boehm addresses the contemporary shortcomings of models that automatically and quantitatively evaluate the quality of software. In essence his models attempts to qualitatively define software quality by a given set of attributes and metrics. Boehm's model is similar to the McCall Quality Model in that it also presents a hierarchical quality model structured around high-level characteristics, intermediate level characteristics, primitive characteristics - each of which contributes to the overall quality level.

The high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put – the general utility of software. The high-level characteristics address three main questions that a buyer of software has:

- As-is utility: How well (easily, reliably, efficiently) can I use it as-is?
- Maintainability: How easy is it to understand, modify and retest?
- Portability: Can I still use it if I change my environment?

The intermediate level characteristic represents Boehm's 7 quality factors that together represent the qualities expected from a software system:

- Portability (General utility characteristics): Code possesses the characteristic portability to the extent that it can be operated easily and well on computer configurations other than its current one.
- Reliability (As-is utility characteristics): Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily.
- Efficiency (As-is utility characteristics): Code possesses the characteristic efficiency to the extent that it fulfills its purpose without waste of resources.
- Usability (As-is utility characteristics, Human Engineering): Code possesses the characteristic usability to the extent that it is reliable, efficient and human-engineered.
- Testability (Maintainability characteristics): Code possesses the characteristic testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance.
- Understandability (Maintainability characteristics): Code possesses the characteristic understandability to the extent that its purpose is clear to the inspector.
- Flexibility (Maintainability characteristics, Modifiability): Code possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined. (Note the higher level of abstractness of this characteristic as compared with augmentability).

The lowest level structure of the characteristics hierarchy in Boehm's model is the primitive characteristics metrics hierarchy. The primitive characteristics provide the foundation for defining qualities metrics – which was one of the

goals when Boehm constructed his quality model. Consequently, the model presents one ore more metrics² supposedly measuring a given primitive characteristic.

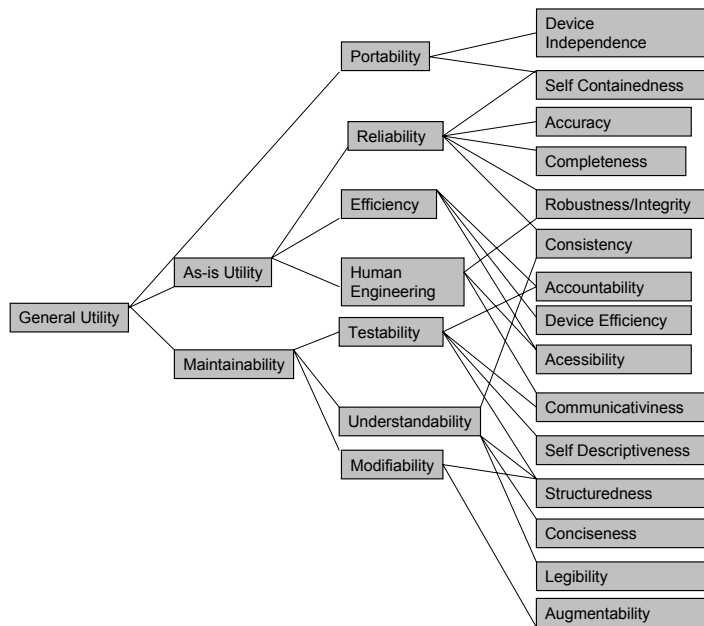


Figure 4: Boehm's Software Quality Characteristics Tree [13]. As-is Utility, Maintainability, and Portability are necessary (but not sufficient) conditions for General Utility. As-is Utility requires a program to be Reliable and adequately Efficient and Human-Engineered. Maintainability requires that the user be able to understand, modify, and test the program, and is aided by good Human-engineering

Though Boehm's and McCall's models might appear very similar, the difference is that McCall's model primarily focuses on the precise measurement of the high-level characteristics "As-is utility" (see Figure 4 above), whereas Boehm's quality mode model is based on a wider range of characteristics with an extended and detailed focus on primarily maintainability. Figure 5 compares the two quality models, quality factor by quality factor.

<i>Criteria/goals</i>	<i>McCall, 1977</i>	<i>Boehm, 1978</i>
Correctness	*	*
Reliability	*	*
Integrity	*	*
Usability	*	*
Efficiency	*	*
Maintainability	*	*
Testability	*	
Interoperability	*	
Flexibility	*	*
Reusability	*	*
Portability	*	*
Clarity		*
Modifiability		*
Documentation		*
Resilience		*
Understandability		*
Validity		*
Functionality		
Generality		*
Economy		*

² Defined by Boehm as: "a measure of extent or degree to which a product possesses and exhibits a certain (quality) characteristic".

Figure 5: Comparison between criteria/goals of the McCall and Boehm quality models [14].

As indicated in Figure 5 above Boehm focuses a lot on the models effort on software maintenance cost-effectiveness – which, he states, is the primary payoff of an increased capability with software quality considerations.

1.4.3. FURPS/FURPS+

A later, and perhaps somewhat less renown, model that is structured in basically the same manner as the previous two quality models (but still worth at least to be mentioned in this context) is the FURPS model originally presented by Robert Grady [15] (and extended by Rational Software [16-18] - now IBM Rational Software - into FURPS+³). FURPS stands for:

- Functionality – which may include feature sets, capabilities and security
- Usability - which may include human factors, aesthetics, consistency in the user interface, online and context-sensitive help, wizards and agents, user documentation, and training materials
- Reliability - which may include frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failure (MTBF)
- Performance - imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage
- Supportability - which may include testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, localizability (internationalization)

The FURPS-categories are of two different types: Functional (F) and Non-functional (URPS). These categories can be used as both product requirements as well as in the assessment of product quality.

1.4.4. Dromey's Quality Model

An even more recent model similar to the McCall's, Boehm's and the FURPS(+) quality model, is the quality model presented by R. Geoff Dromey [19;20]. Dromey proposes a product based quality model that recognizes that quality evaluation differs for each product and that a more dynamic idea for modeling the process is needed to be wide enough to apply for different systems. Dromey is focusing on the relationship between the quality attributes and the sub-attributes, as well as attempting to connect software product properties with software quality attributes.

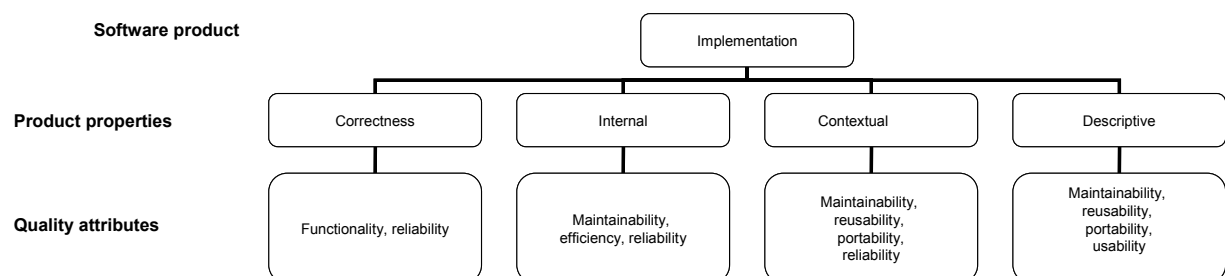


Figure 6: Principles of Dromey's Quality Model

As Figure 6 illustrates, there are three principal elements to Dromey's generic quality model

³ The "+" in FURPS+ includes such requirements as design constraints, implementation requirements, interface requirements and physical requirements.

- 1) Product properties that influence quality
- 2) High level quality attributes
- 3) Means of linking the product properties with the quality attributes.

Dromey's Quality Model is further structured around a 5 step process:

- 1) Chose a set of high-level quality attributes necessary for the evaluation.
- 2) List components/modules in your system.
- 3) Identify quality-carrying properties for the components/modules (qualities of the component that have the most impact on the product properties from the list above).
- 4) Determine how each property effects the quality attributes.
- 5) Evaluate the model and identify weaknesses.

1.4.5. ISO

1.4.5.1 ISO 9000

The renowned ISO acronym stands for International Organization for Standardization⁴. The ISO organization is responsible for a whole battery of standards of which the ISO 9000 [21-25] (depicted in Figure 7 below) family probably is the most well known, spread and used.

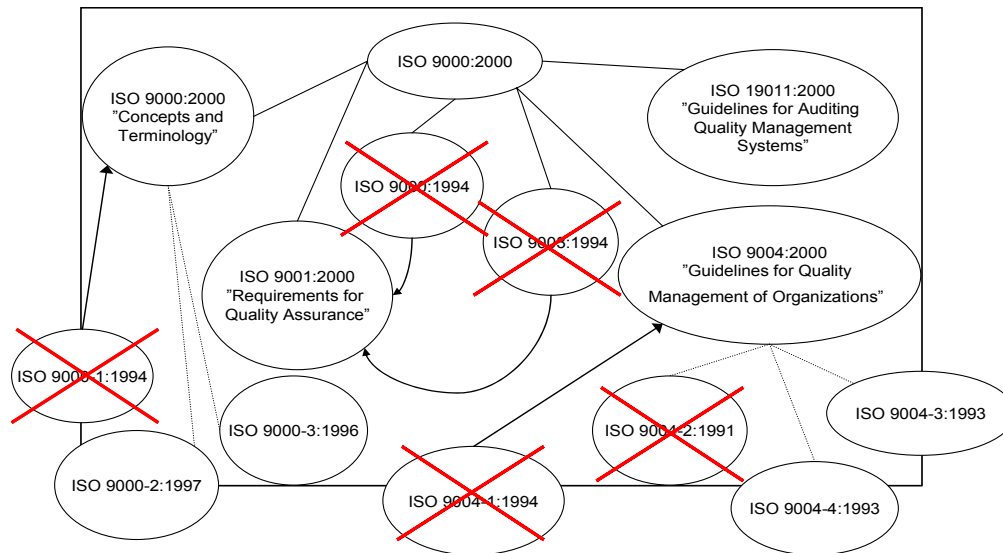


Figure 7: The ISO 9000:2000 standards. The crosses and arrows indicate changes made from the older ISO 9000 standard to the new ISO 9000:2000 standard.

ISO 9001 is an international quality management system standard applicable to organizations within all type of businesses. ISO 9001 internally addresses an organization's processes and methods and externally at managing (controlling, assuring etc.) the quality of delivered products and services. ISO 9001 is a process oriented approach towards quality management. That is, it proposes designing, documenting, implementing, supporting, monitoring, controlling and improving (more or less) each of the following processes:

- Quality Management Process
- Resource Management Process
- Regulatory Research Process
- Market Research Process
- Product Design Process
- Purchasing Process
- Production Process
- Service Provision Process
- Product Protection Process
- Customer Needs Assessment Process

⁴ ISO was chosen instead of IOS, because iso in Greek means equal, and ISO wanted to convey the idea of equality - the idea that they develop standards to place organizations on an equal footing.

- Customer Communications Process
- Internal Communications Process
- Document Control Process
- Record Keeping Process
- Planning Process
- Training Process
- Internal Audit Process
- Management Review Process
- Monitoring and Measuring Process
- Nonconformance Management Process
- Continual Improvement Process

1.4.5.2 ISO 9126

Besides the famous ISO 9000, ISO has also release the ISO 9126: Software Product Evaluation: Quality Characteristics and Guidelines for their Use-standard⁵ [26] (among other standards).

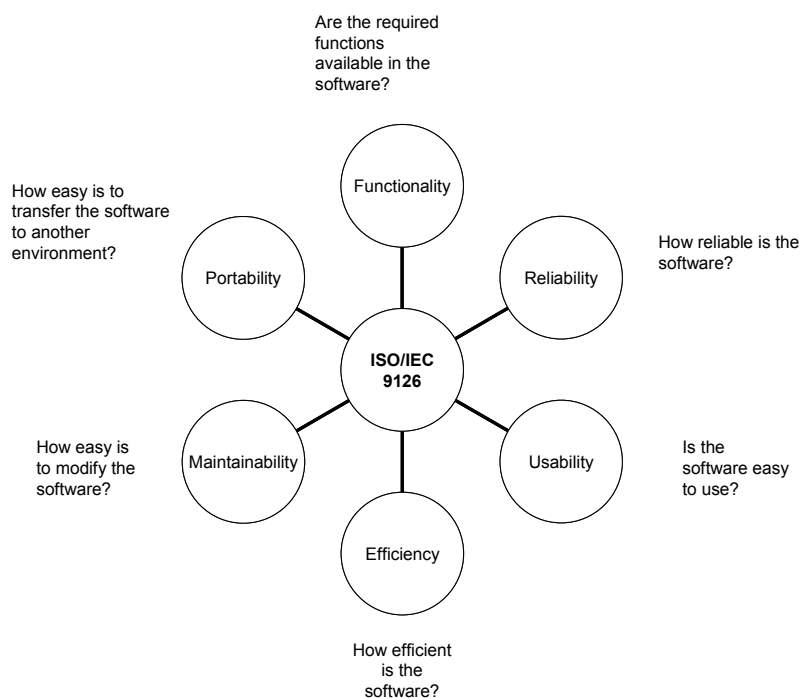


Figure 8: The ISO 9126 quality model

This standard was based on the McCall and Boehm models. Besides being structured in basically the same manner as these models (see Figure 10), ISO 9126 also includes functionality as a parameter, as well as identifying both internal and external quality characteristics of software products.

⁵ ISO/IEC 9126:2001 contains 4 parts

- Part 1: Quality Model
- Part 2: External Metrics
- Part 3: Internal Metrics
- Part 4: Quality in use metrics

<i>Criteria/goals</i>	<i>McCall, 1977</i>	<i>Boehm, 1978</i>	<i>ISO 9126, 1993</i>
Correctness	*	*	maintainability
Reliability	*	*	*
Integrity	*	*	
Usability	*	*	*
Efficiency	*	*	*
Maintainability	*	*	*
Testability	*		maintainability
Interoperability	*		
Flexibility	*	*	
Reusability	*	*	
Portability	*	*	*
Clarity		*	
Modifiability		*	maintainability
Documentation		*	
Resilience		*	
Understandability		*	
Validity		*	maintainability
Functionality			*
Generality		*	
Economy		*	

Figure 9: Comparison between criteria/goals of the McCall, Boehm and ISO 9126 quality models [14].

ISO 9126 proposes a standard which species six areas of importance, i.e. quality factors, for software evaluation.

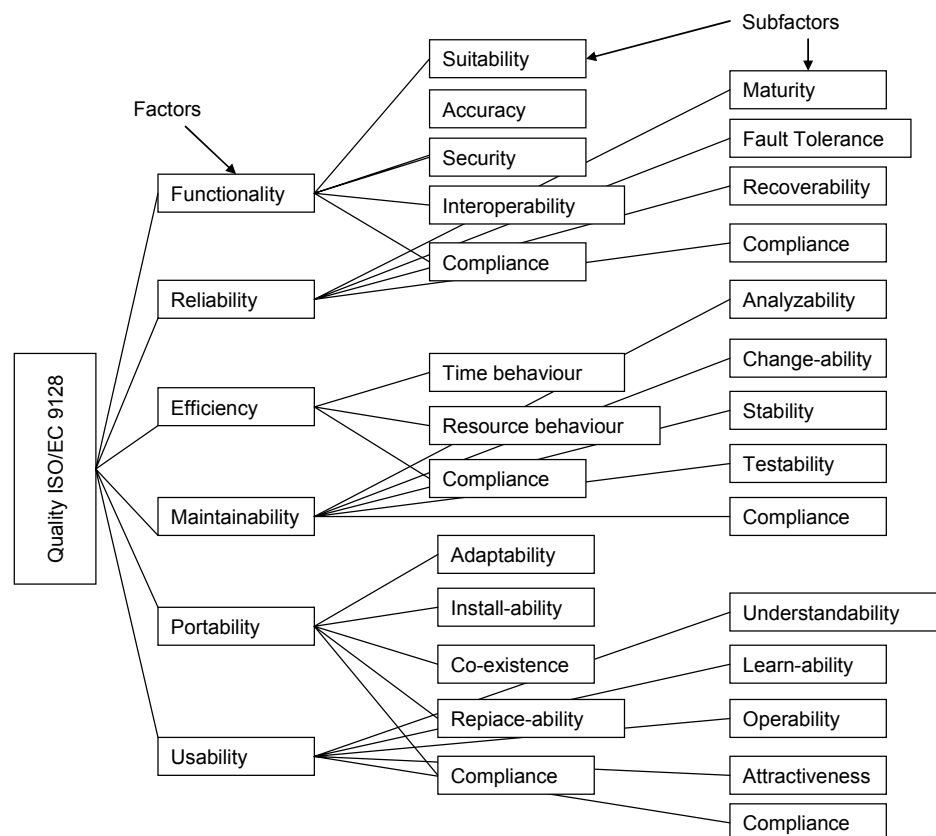


Figure 10: ISO 9126: Software Product Evaluation: Quality Characteristics and Guidelines for their Use

Each quality factors and its corresponding sub-factors are defined as follows:

- **Functionality:** A set of attributes that relate to the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
 - **Suitability:** Attribute of software that relates to the presence and appropriateness of a set of functions for specified tasks.
 - **Accuracy:** Attributes of software that bare on the provision of right or agreed results or effects.
 - **Security:** Attributes of software that relate to its ability to prevent unauthorized access, whether accidental or deliberate, to programs and data.
 - **Interoperability:** Attributes of software that relate to its ability to interact with specified systems.
 - **Compliance:** Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions.
- **Reliability:** A set of attributes that relate to the capability of software to maintain its level of performance under stated conditions for a stated period of time.
 - **Maturity:** Attributes of software that relate to the frequency of failure by faults in the software.
 - **Fault tolerance:** Attributes of software that relate to its ability to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
 - **Recoverability:** Attributes of software that relate to the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it.
 - **Compliance:** See above.
- **Usability:** A set of attributes that relate to the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
 - **Understandability:** Attributes of software that relate to the users' effort for recognizing the logical concept and its applicability.
 - **Learnability:** Attributes of software that relate to the users' effort for learning its application (for example, operation control, input, output).
 - **Operability:** Attributes of software that relate to the users' effort for operation and operation control.
 - **Attractiveness:** -
 - **Compliance:** Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions.
- **Efficiency:** A set of attributes that relate to the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
 - **Time behavior:** Attributes of software that relate to response and processing times and on throughput rates in performing its function.
 - **Resource behavior:** Attributes of software that relate to the amount of resources used and the duration of such use in performing its function.
 - **Compliance:** See above.
- **Maintainability:** A set of attributes that relate to the effort needed to make specified modifications.
 - **Analyzability:** Attributes of software that relate to the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.
 - **Changeability:** Attributes of software that relate to the effort needed for modification, fault removal or for environmental change.
 - **Stability:** Attributes of software that relate to the risk of unexpected effect of modifications.
 - **Testability:** Attributes of software that relate to the effort needed for validating the modified software.
 - **Compliance:** See above.
- **Portability:** A set of attributes that relate to the ability of software to be transferred from one environment to another.
 - **Adaptability:** Attributes of software that relate to on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered.
 - **Installability:** Attributes of software that relate to the effort needed to install the software in a specified environment.
 - **Conformance:** Attributes of software that make the software adhere to standards or conventions relating to portability.
 - **Replaceability:** Attributes of software that relate to the opportunity and effort of using it in the place of specified other software in the environment of that software.

1.4.5.3 ISO/IEC 15504 (SPICE⁶)

The ISO/IEC 15504: Information Technology - Software Process Assessment is a large international standard framework for process assessment that intends to address all processes involved in:

- Software acquisition
- Development
- Operation
- Supply
- Maintenance
- Support

ISO/IEC 15504 consists of 9 component parts covering concepts, process reference model and improvement guide, assessment model and guides, qualifications of assessors, and guide for determining supplier process capability:

- 1) ISO/IEC 15504-1 Part 1: Concepts and Introductory Guide.
- 2) ISO/IEC 15504-2 Part 2: A Reference Model for Processes and Process Capability.
- 3) ISO/IEC 15504-3 Part 3: Performing an Assessment.
- 4) ISO/IEC 15504-4 Part 4: Guide to Performing Assessments.
- 5) ISO/IEC 15504-5 Part 5: An Assessment Model and Indicator Guidance.
- 6) ISO/IEC 15504-6 Part 6: Guide to Competency of Assessors.
- 7) ISO/IEC 15504-7 Part 7: Guide for Use in Process Improvement.
- 8) ISO/IEC 15504-8 Part 8: Guide for Use in Determining Supplier Process Capability.
- 9) ISO/IEC 15504-9 Part 9: Vocabulary.

Given the structure and contents of the ISO/IEC 15504 documentation it is more closely related to ISO 9000, ISO/IEC 12207 and CMM, rather than the initially discussed quality models (McCall, Boehm and ISO 9126).

1.4.6. IEEE

IEEE has also release several standards, more or less related to the topic covered within this technical paper. To name a few:

- IEEE Std. 1220-1998: IEEE Standard for Application and Management of the Systems Engineering Process
- IEEE Std 730-1998: IEEE Standard for Software Quality Assurance Plans
- IEEE Std 828-1998: IEEE Standard for Software Configuration Management Plans – Description
- IEEE Std 829-1998: IEEE Standard For Software Test Documentation
- IEEE Std 830-1998: IEEE recommended practice for software requirements specifications
- IEEE Std 1012-1998: IEEE standard for software verification and validation plans
- IEEE Std 1016-1998: IEEE recommended practice for software design descriptions
- IEEE Std 1028-1997: IEEE Standard for Software Reviews
- IEEE Std 1058-1998: IEEE standard for software project management plans
- IEEE Std 1061-1998: IEEE standard for a software quality metrics methodology
- IEEE Std 1063-2001: IEEE standard for software user documentation
- IEEE Std 1074-1997: IEEE standard for developing software life cycle processes
- IEEE/EIA 12207.0-1996: Standard Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology Software Life Cycle Processes

Of the above mentioned standards it is probably the implementation of ISO/IEC 12207: 1995 that most resembles previously discussed models in that it describes the processes for the following life-cycle:

- Primary Processes: Acquisition, Supply, Development, Operation, and Maintenance.
- Supporting Processes: Documentation, Configuration Management, Quality Assurance, Verification, Validation, Joint Review, Audit, and Problem Resolution.
- Organization Processes: Management, Infrastructure, Improvement, and Training

In fact, IEEE/EIA 12207.0-1996 is so similar to the ISO 9000 standard that it could actually be seen as a potential replacement for ISO within software engineering organizations.

The IEEE Std 1061-1998 is another standard that is relevant from the perspective of this technical paper as the standard provides a methodology for establishing quality requirements and identifying, implementing, analyzing and validating the process and product of software quality metrics.

⁶ SPICE is an acronym for “Software Process Improvement and Capability dEtermination”

1.4.7. Capability Maturity Model(s)

The Carnegie Mellon Software Engineering Institute (SEI), non-profit group sponsored by the DoD work at getting US software more reliable. Examples of relevant material produced from SEI is the PSP [27;28] and TSPi [29]. While PSP and TSPi briefly brushes the topic of this technical paper, SEI has also produced a number of more extensive Capability Maturity Models that in a very IEEE and ISO 9000 similar manner addresses the topic of software quality:

- CMM / SW-CMM [28;30;31]
- P-CMM [32]
- CMMI [33]
 - PDD-CMM
 - SE-CMM
 - SA-CMM

The CMM/SW-CMM depicted in Figure 11 below addresses the issue of software quality from a process perspective.

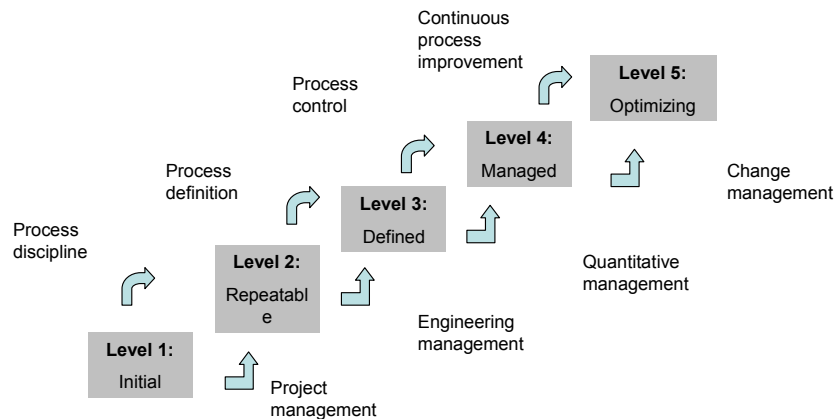


Figure 11: Maturity Levels of (SW-)CMM

Table 1: Maturity levels with corresponding focus and key process areas for CMM.

Level	Focus	Key Process Area
Level 5 – Optimizing level	Continuous improvement	Process Change Management Technology Change Management Defect Prevention
Level 4 – Managed level	Product and process quality	Software Quality Management Quantitative Process Management
Level 3 – Defined level	Engineering process	Organization Process Focus Organization Process Definition Peer Reviews Training Program Intergroup Coordination Software Product Engineering Integrated Software Management
Level 2 – Repeatable level	Project Management	Requirements Management Software Project Planning Software Project Tracking and Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management
Level 1 – Initial level	Heroes	No KPAs at this time

The SW-CMM is superseded by the CMMI model which also incorporates some other CMM models into a wider scope. CMMI Integrates systems and software disciplines into one process improvement framework and is structured around the following process areas:

- Process management
- Project management
- Engineering
- Support

...and similarly to the SW-CMM the following maturity levels:

- Maturity level 5: Optimizing - Focus on process improvement
- Maturity level 4: Quantitatively managed - Process measured and controlled.
- Maturity level 3: Defined - Process characterized for the organization and is proactive.
- Maturity level 2: Managed - Process characterized for projects and is often reactive.
- Maturity level 1: Initial - Process unpredictable, poorly controlled and reactive.
- Maturity level 0: Incomplete

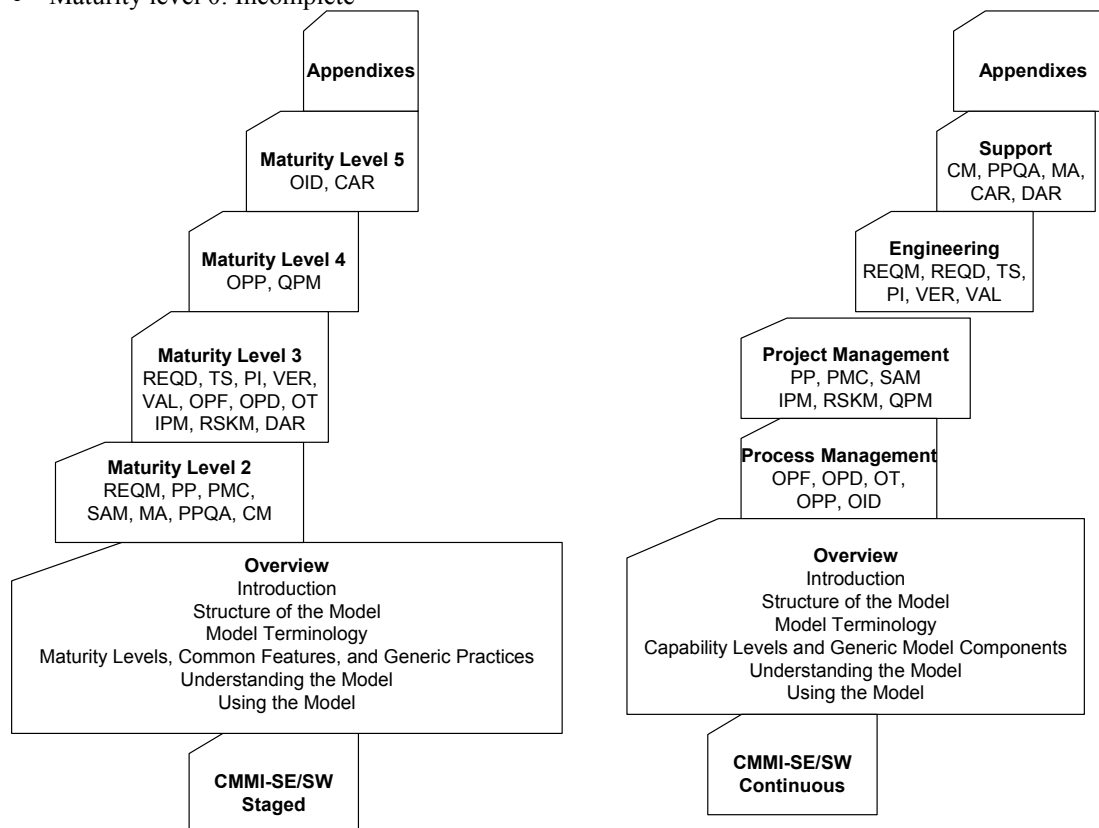


Figure 12: The two representations of the CMMI model.

1.4.8. Six Sigma

Given that we are trying to provide a somewhat all covering picture of the more known quality models and philosophies we also need to at least mention Six Sigma. Six Sigma can be viewed as a management philosophy that uses customer-focused measurement and goal-setting to create bottom-line results. It strongly advocates listening to the voice of the customer and converting customer needs into measurable requirements.

1.5. Conclusion and discussions

Throughout this chapter the ambition has been to briefly survey some different structures of quality – without any deepening drilldowns in a particular model or philosophy. The idea was to nuance and provide an overview of the landscape of what sometimes briefly (and mostly thoughtlessly) simply is labeled quality. The paper has shown that quality can be a very elusive concept that can be approached from a number of perspective dependent on once take and interest. Garvin [11;34] has made a cited attempt to sort out the different views on quality. He the following organization of the views:

- Transcendental view, where quality is recognized but not defined. The transcendental view is a subjective and non quantifiable of defining software quality. It often results in software that transcends customer expectations.
- User view on quality or “fitness for purpose” takes the starting point in software that meets the users’ needs. Reliability (failure rate, MTBF), Performance/Efficiency (time to perform a task), Maintainability and Usability are issues within this view.
- Manufacturing view on quality focuses on conformance to specification and the organizations capacity to produce software according to the software process. Here product quality is achieved through process quality. Waste reduction, Zero defect, Right the first time (defect count and fault rates, staff effort rework costs) are concepts usually found within this view.
- Product view on quality usually specifies that the characteristics of product are defined by the characteristics of its subparts, e.g. size, complexity, and test coverage. Module complexity measures, Design & code measures etc.
- Value based view on quality measures and produces value for money by balancing requirements, budget and time, cost & price, deliver dates (lead time, calendar time), productivity etc.

Most of the quality models presented within this technical paper probably could be fitted within the user view, manufacturing view or product view – though this is a futile exercise with little meaning. The models presented herein are focused around either processes or capability level (ISO, CMM etc.) where quality is measured in terms of adherence to the process or capability level, or a set of attributed/metrics used to distinctively assess quality (McCall, Boehm etc.) by making quality a quantifiable concept. Though having some advantages (in terms of objective measurability), quality models actually reduce the notion of quality to a few relatively simple and static attributes. This structure of quality is in great contrast to the dynamic, moving target, fulfilling the customers’ ever changing expectations perspective presented by some of the quality management gurus. It is easy to see that the quality models represent leaner and narrower perspectives on quality than the management philosophies presented by the quality gurus. The benefit of quality models is that they are simpler to use. The benefit of the quality management philosophies is that they probably more to the point capture the idea of quality.

1.6. References

- [1] Hoyer, R. W. and Hoyer, B. B. Y., "What is quality?", *Quality Progress*, no. 7, pp. 52-62, 2001.
- [2] Robson, C., *Real world research: a resource for social scientists and practitioner-researchers*, Blackwell Publisher Ltd., 2002.
- [3] Crosby, P. B., *Quality is free : the art of making quality certain*, New York : McGraw-Hill, 1979.
- [4] Deming, W. E., *Out of the crisis : quality, productivity and competitive position*, Cambridge Univ. Press, 1988.
- [5] Feigenbaum, A. V., *Total quality control*, McGraw-Hill, 1983.
- [6] Ishikawa, K., *What is total quality control? : the Japanese way*, Prentice-Hall, 1985.
- [7] Juran, J. M., *Juran's Quality Control Handbook*, McGraw-Hill, 1988.
- [8] Shewhart, W. A., *Economic control of quality of manufactured product*, Van Nostrand, 1931.
- [9] McCall, J. A., Richards, P. K., and Walters, G. F., "Factors in Software Quality", *Nat'l Tech.Information Service*, no. Vol. 1, 2 and 3, 1977.
- [10] Marciniak, J. J., *Encyclopedia of software engineering, 2vol, 2nd ed.*, Chichester : Wiley, 2002.
- [11] Kitchenham, B. and Pfleeger, S. L., "Software quality: the elusive target [special issues section]", *IEEE Software*, no. 1, pp. 12-21, 1996.
- [12] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M., *Characteristics of Software Quality*, North Holland, 1978.
- [13] Boehm, Barry W., Brown, J. R., and Lipow, M.: *Quantitative evaluation of software quality*, International Conference on Software Engineering, Proceedings of the 2nd international conference on Software engineering, 1976.
- [14] Hyatt, Lawrence E. and Rosenberg, Linda H.: *A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality*, European Space Agency Software Assurance Symposium and the 8th Annual Software Technology Conference, 1996.
- [15] Grady, R. B., *Practical software metrics for project management and process improvement*, Prentice Hall, 1992.
- [16] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*, Addison Wesley Longman, Inc., 1999.
- [17] Kruchten, P., *The Rational Unified Process An Introduction - Second Edition*, Addison Wesley Longman, Inc., 2000.
- [18] Rational Software Inc., *RUP - Rational Unified Process*, www.rational.com, 2003.
- [19] Dromey, R. G., "Concerning the Chimera [software quality]", *IEEE Software*, no. 1, pp. 33-43, 1996.

- [20] Dromey, R. G., "A model for software product quality", *IEEE Transactions on Software Engineering*, no. 2, pp. 146-163, 1995.
- [21] ISO, International Organization for Standardization, "ISO 9000:2000, Quality management systems - Fundamentals and vocabulary", 2000.
- [22] ISO, International Organization for Standardization, "ISO 9000-2:1997, Quality management and quality assurance standards — Part 2: Generic guidelines for the application of ISO 9001, ISO 9002 and ISO 9003", 1997.
- [23] ISO, International Organization for Standardization, "ISO 9000-3:1998 -- Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001_1994 to the development, supply, installation and maintenance of computer software (ISO 9000-3:1997)", 1998.
- [24] ISO, International Organization for Standardization, "ISO 9001:2000, Quality management systems – Requirements", 2000.
- [25] ISO, International Organization for Standardization, "ISO 9004:2000, Quality management systems - Guidelines for performance improvements", 2000.
- [26] ISO, International Organization for Standardization, "ISO 9126-1:2001, Software engineering - Product quality, Part 1: Quality model", 2001.
- [27] Humphrey, W. S., *Introduction to the Personal Software Process*, Addison-Wesley Pub Co; 1st edition (December 20, 1996), 1996.
- [28] Humphrey, W. S., *Managing the software process*, Addison-Wesley, 1989.
- [29] Humphrey, W. S., *Introduction to the team software process*, Addison-Wesley, 2000.
- [30] Paulk, Mark C., Weber, Charles V., Garcia, Suzanne M., Chrissis, Mary Beth, and Bush, Marilyn, "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, Carnegie Mellon University, 1993.
- [31] Paulk, Mark C., Weber, Charles V., Garcia, Suzanne M., Chrissis, Mary Beth, and Bush, Marilyn, "Key practices of the Capability Maturity Model, version 1.1", 1993.
- [32] Curtis, Bill, Hefley, Bill, and Miller, Sally, "People Capability Maturity Model® (P-CMM®), Version 2.0", Software Engineering Institute, Carnegie Mellon University, 2001.
- [33] Carnegie Mellon, Software Engineering Institute, *Welcome to the CMMI® Web Site*, Carnegie Mellon, Software Engineering Institute, <http://www.sei.cmu.edu/cmmi/cmmi.html>, 2004.
- [34] Garvin, D. A., "What does 'Product Quality' really mean?", *Sloan Management Review*, no. 1, pp. 25-43, 1984.

ISHIKAWA'S 7 TOOLS

1. Check sheet - a paper form with printed items to be checked.
 - Its main purpose is to facilitate gathering data and to arrange data while collecting it so the data can be easily used later.
 - Another type of check sheet is the check-up confirmation sheet. It is concerned mainly with the quality characteristics of a process or a product.
 - To distinguish a confirmation check sheet from the ordinary data-gathering check sheet, we use the term checklist.
 - A check sheet is a tool used for data collection.

2. Pareto diagram - a frequency chart of bars in descending order; the frequency bars are usually associated with types of problems.
 - Named after a nineteenth-century Italian economist named Vilfredo Pareto who expounded his principle in terms of the distribution of wealth that a large share of the wealth is owned by a small percentage of the population.
 - In 1950 Juran applied the principle to the identification of quality problems - that most of the quality problems are due to a small percentage of the possible causes.
 - In software development, the X-axis for a Pareto diagram is usually the defect cause and the Y-axis the defect count.
 - By arranging the causes based on defect frequency, a Pareto diagram can identify the few causes that account for the majority of defects.
 - It indicates which problems should be solved first in eliminating defects and improving the operation.
 - Pareto analysis is commonly referred to as the 80-20 principle (20% of the causes account for 80% of the defects), although the cause-defect relationship is not always in an 80-20 distribution.
 - A bar graph used to make visible the vital few vs. the trivial many.

3. Histogram - a graphic representation of frequency counts of a sample or a population.
 - The X-axis lists the unit intervals of a parameter (e.g., severity level of software defects) ranked in ascending order from left to right, and the Y-axis contains the frequency counts.
 - In a histogram, the frequency bars are shown by the order of the X variable, whereas in a Pareto diagram the frequency bars are shown by order of the frequency counts.

- The purpose of the histogram is to show the distribution characteristics of a parameter such as overall shape, central tendency, dispersion, and skewness.
 - It enhances understanding of the parameter of interest.
4. Run chart - tracks the performance of the parameter of interest over time.
- The X-axis is time and the Y axis is the value of the parameter.
 - A run chart is best used for trend analysis, especially if historical data are available for comparisons with the current trend.
 - Ishikawa (1989) includes various graphs such as the pie chart, bar graph, compound bar graph, and circle graph under the section that discusses run charts.
 - An example of a run chart in software is the weekly number of open problems in the backlog; it shows the development team's workload of software fixes.
 - A Run Chart is a basic graph that displays data values in a time sequence (the order in which the data were generated). A Run Chart can be useful for identifying trends or shifts in process.
5. Scatter diagram - it vividly portrays the relationship of two interval variables.
- In a cause-effect relationship, the X-axis is for the independent variable and the Y-axis for the dependent variable.
 - Each point in a scatter diagram represents an observation of both the dependent and independent variables.
 - Scatter diagrams aid data-based decision making (e.g., if action is planned on the X variable and some effect is expected on the Y variable).
 - One should always look for a scatter diagram when the correlation coefficient of two variables is presented. This is because the method for calculating the correlation coefficient is highly sensitive to outliers, and a scatter diagram can clearly expose any outliers in the relationship.
 - Second, the most common correlation coefficient is Pearson's product moment correlation coefficient, which assumes a linear relationship.
 - If the relationship is nonlinear, the Pearson correlation coefficient may show no relationship; therefore, it may convey incorrect or false information.

- When using a scatter diagram there are two types of variables – a dependent variable and an independent variable.
 - The independent variable is usually plotted along the horizontal axis.
 - The dependent variable is usually plotted along the vertical axis.
 - If no dependent variable exists, either type of variable can be plotted on either axis.
- If the clustering of intersecting dots in the paired comparisons shows a pattern that extends from lower-left to upper-right, the scatter diagram shows evidence of a positive correlation. If the pattern of dots tends to go from the upper-left to the bottom-right, there is evidence of a negative correlation.

6. Control chart - can be regarded as an advanced form of a run chart for situations where the process capability can be defined.

- It consists of a central line, a pair of control limits (and sometimes a pair of warning limits within the control limits), and values of the parameter of interest plotted on the chart, which represent the state of a process.
- The X-axis is real time.
- If all values of the parameter are within the control limits and show no particular tendency, the process is regarded as being in a controlled state.
- If they fall outside the control limits or indicate a trend, the process is considered out of control.
- Such cases call for causal analysis and corrective actions are to be taken.
- The control chart is meant to separate common cause variation from assignable-cause variation. A control chart is useful in knowing when to act, and when to leave the process alone.

7. Cause-and-effect diagram - also known as the fishbone diagram

- Was developed by Ishikawa and associates in the early 1950s in Japan.
- It was first used to explain factors that affect the production of steel. It is included in the Japanese Industrial Standards terminology for quality control (Kume, 1989).

- It shows the relationship between a quality characteristic and factors that affect that characteristic.
- Its layout resembles a fishbone, with the quality characteristic of interest labelled at the fish head, and factors affecting the characteristics placed where the bones are located.
- While the scatter diagram describes a specific bivariate relationship in detail, the cause-and-effect diagram identifies all causal factors of a quality characteristic in one chart.
- The cause/effect diagram is drawn on a large whiteboard or a large flipchart. The *effect* is written at the 3 o'clock position. A horizontal line divides the whiteboard into two equal parts. Many times the 6M's (i.e., Man (in the generic sense), Machine, Material, Method, Measurement, and Mother Nature) are used as branches off of the horizontal line that lead to the effect on the right-hand side of the whiteboard.

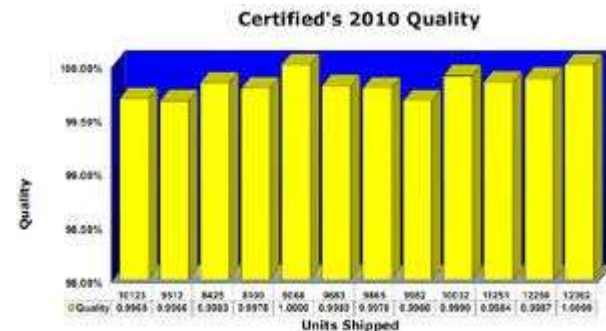
Metrics

***Express in
Numbers***

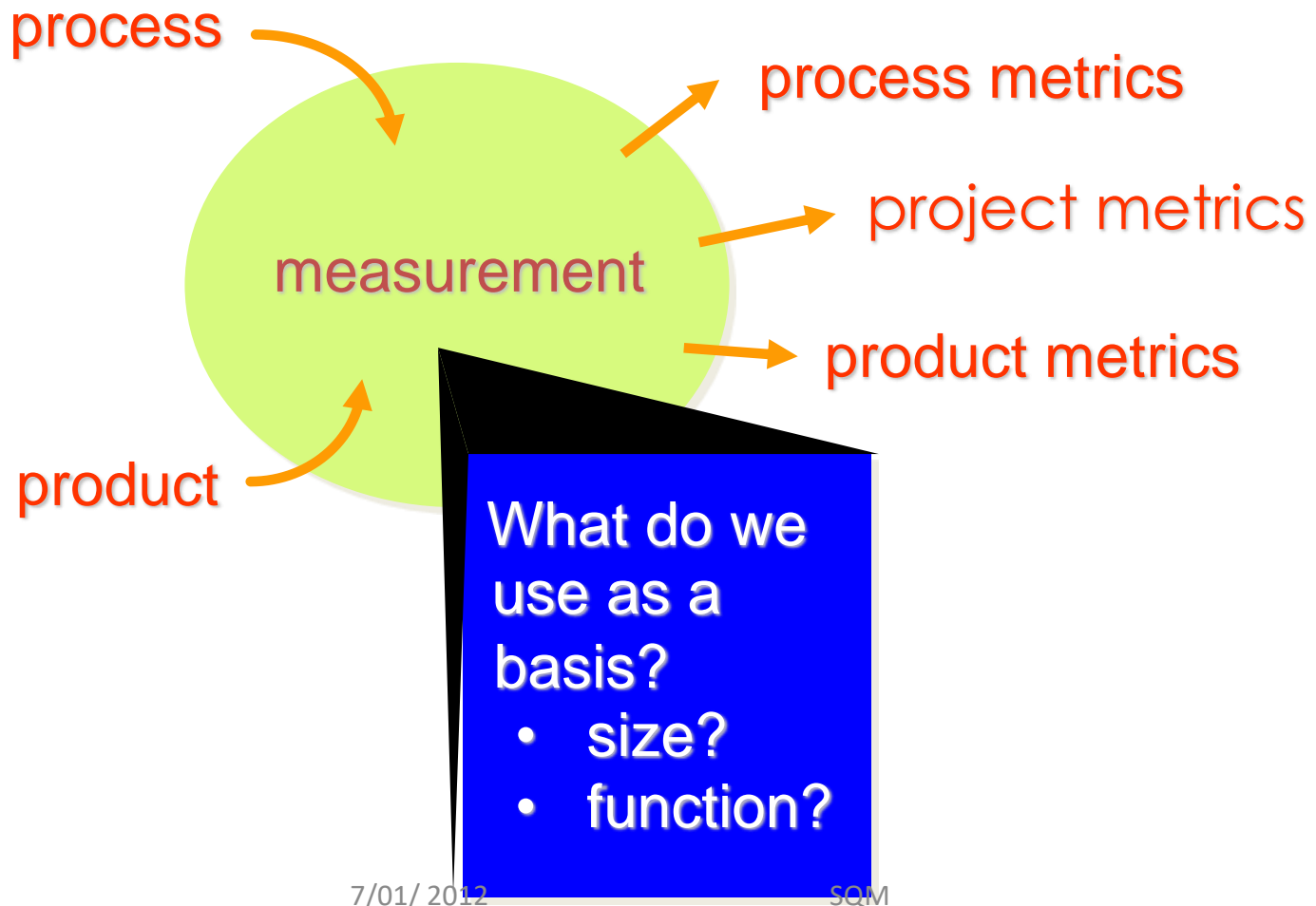
**Measurement provides a mechanism
for objective evaluation**

Measure, Metrics, Indicators

- Measure.
 - provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attributes of a product or process.
- Metrics.
 - relates the individual measures in some way.
- Indicator.
 - a combination of metrics that provide insight into the software process or project or product itself.



What Should Be Measured?



Metrics of Project Management



- Budget
- Schedule/ReResource Management
- Risk Management
- Project goals met or exceeded
- Customer satisfaction

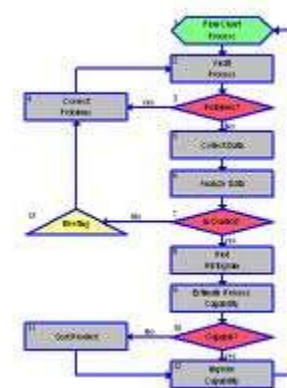
Metrics of the Software Product



- Focus on Deliverable Quality
- Analysis Products
- Design Product Complexity — algorithmic, architectural, data flow
- Code Products
- Production System

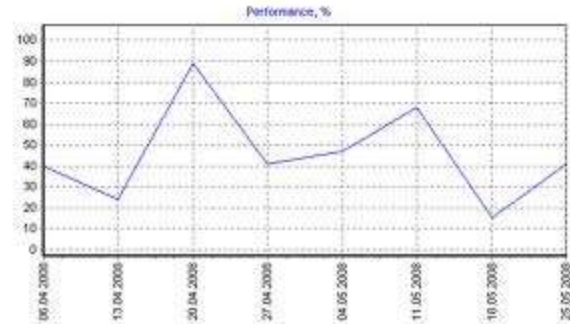
Lines of Code (LOC)

- A representation of program size
- Can be measured in different ways
 - HLL Source Statements
 - Assembler Statements
 - Executable lines
 - Executable lines plus data definitions



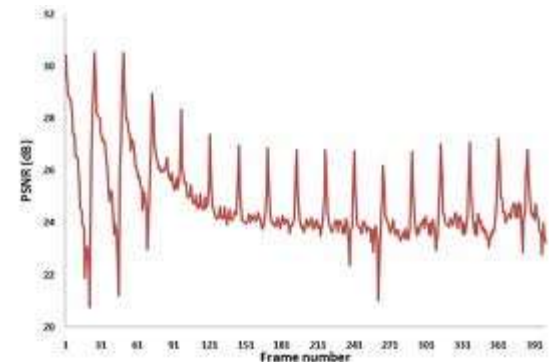
Lines of Code (LOC)

- General assumption:
 - The more lines of code, the more defects are expected, and a higher defect density (defects per KLOC) is expected
- Research has shown that the general assumption may not be true in all cases.
 - A optimum code size may exist in which expected defect rates would be contained within an acceptable upper limit.
 - Such an optimum may depend on language, project, product, and environment



Function Oriented Metric - Function Points

- Function Points are a measure of “how big” is the program, independently from the actual physical size of it
- It is a weighted count of several features of the program
- Dislikers claim FP make no sense wrt the representational theory of measurement
- There are firms and institutions taking them very seriously



Function Points comp.

- STEP1
- Count the number of **external inputs, external outputs, external inquiries, internal logic files, and external interface files required**. The IFPUG provides the following definitions for these:
 - *External Input (EI)*: An EI is an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input screen or another application. The data may be used to maintain one or more internal logical files. The data can be either control information or business information. If the data is control information it does not have to update an internal logical file.
 - *External Output (EO)*: An EO is an elementary process in which derived data passes across the boundary from inside to outside. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications. These reports and files are created from one or more internal logical files and external interface file.
 - *External Inquiry (EQ)*: An EQ is an elementary process with both input and output components that result in data retrieval from one or more internal logical files and external interface files. The input process does not update any Internal Logical Files, and the output side does not contain derived data.
 - *Internal Logic File (ILF)*: AN ILF is a user identifiable group of logically related data that resides entirely within the applications boundary and is maintained through external inputs.
 - *External Interface File (EIF)*: An EIF is a user identifiable group of logically related data that is used for reference purposes only. The data resides entirely outside the application and is maintained by another application. The external interface file is an internal logical file for another application.

Function Points comp....

- STEP2
- Rate each component (EI,EO,EQ,ILF,EIF) as low, average, or high.
- For transactions (EI's, EO's, EQ's) the ranking is based upon the number of files updated or referenced (FTR's) and the number of data element types (DET's).
- For both ILF's and EIF's files the ranking is based upon Record Element Types (RET's) and Data Element Types (DET's).
 - A record element type (RET) is a user recognizable subgroup of data elements within an ILF or EIF.
 - A data element type (DAT) is a unique user recognizable, non-recursive, field.
 - File Type Referenced (FTR): A FTR is a file type referenced by a transaction. An FTR must also be an internal logical file or external interface file.

Component	RET's	FTR's	DET's
External Inputs (EI)		✓	✓
External Outputs (EO)		✓	✓
External Inquiries (EQ)		✓	✓
External Interface Files (EIF)	✓		✓
Internal Logical Files (ILF)	✓		✓

Function Points comp.

- STEP3
- Multiply each count by the numerical rating shown for (low, average,high) to determine the rated value.
- Sum the rated values in each row (EI,EO,EQ,ILF,EIF) giving a total value for each type of component .
- Sum the totals in the rightmost column for each component to give Total Number of Unadjusted Function Points (UAF).

Type of Component	Complexity of Components			
	Low	Average	High	Total
External Inputs	x 3 =	x 4 =	x 6 =	
External Outputs	x 4 =	x 5 =	x 7 =	
External Inquiries	x 3 =	x 4 =	x 6 =	
Internal Logical Files	x 7 =	x 10 =	x 15 =	
External Interface Files	x 5 =	x 7 =	x 10 =	
Total Number of Unadjusted Function Points				
Multiplied Value Adjustment Factor				
Total Adjusted Function Points				

Function Points comp.

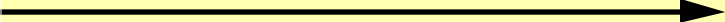
- STEP4
- Next we calculate a value adjustment factor (VAF) based on 14 general system characteristics (GSC's) that rate the general functionality of the application being counted.
- Each characteristic has associated descriptions that help determine the degrees of influence of the characteristics. The degrees of influence range on a scale of zero to five, from no influence to strong influence.

General System Characteristic		Brief Description
1.	Data communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
2.	Distributed data processing	How are distributed data and processing functions handled?
3.	Performance	Did the user require response time or throughput?
4.	Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?
5.	Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?
6.	On-Line data entry	What percentage of the information is entered On-Line?
7.	End-user efficiency	Was the application designed for end-user efficiency?
8.	On-Line update	How many ILF's are updated by On-Line transaction?
9.	Complex processing	Does the application have extensive logical or mathematical processing?
10.	Reusability	Was the application developed to meet one or many user's needs?
11.	Installation ease	How difficult is conversion and installation?
12.	Operational ease	How effective and/or automated are start-up, back up, and recovery procedures?
13.	Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
14.	Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?

Function Points comp...

- STEP5
 - Calculate the Value Adjusted Function point (VAF) using the IFPUG value adjustment equation with the 14 weightings allocated to each General System Characteristic (GSC):
 - $VAF = 0.65 + 0.01 * \text{Sum (GSC}_i)$
 - for all i in $[1..14]$, where 0.65 and 0.01 are empirically derived constants.
 - That is, VAF is calculated by summing all of the answers to the 14 questions (weighted 0-5), multiplying this sum by the factor 0.01 and then adding 0.65.
- STEP 6
 - The final Function Point Count is obtained by multiplying the VAF times the Unadjusted Function Point (UAF).
 - $FP = UAF * VAF$ (function points)

Analyzing the Information Domain

<u>measurement parameter</u>	<u>count</u>	<u>weighting factor</u>					
		<u>simple</u>	<u>avg.</u>	<u>complex</u>			
number of user inputs	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of user outputs	<input type="text"/>	X	4	5	7	=	<input type="text"/>
number of user inquiries	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of files	<input type="text"/>	X	7	10	15	=	<input type="text"/>
number of ext.interfaces	<input type="text"/>	X	5	7	10	=	<input type="text"/>
Unadjusted Function Points: 							<input type="text"/>

Assuming all inputs with the same weight, all output with the same weight, ...

Complete Formula for the Unadjusted Function Points:



$$\sum_{Inputs} W_i + \sum_{Output} W_o + \sum_{Inquiry} W_{in} + \sum_{InternalFiles} W_{if} + \sum_{ExternalInterfaces} W_{ei}$$

Taking Complexity into Account

Factors are rated on a scale of 0 (not important) to 5 (very important):

data communications
distributed functions
heavily used configuration
transaction rate
on-line data entry
end user efficiency

on-line update
complex processing
installation ease
operational ease
multiple sites
facilitate change

Formula:


$$CM = \sum_{ComplexityMultiplier} F_{ComplexityMultiplier}$$

Typical Function-Oriented Metrics

- errors per FP (thousand lines of code)
- defects per FP
- \$ per FP
- pages of documentation per FP
- FP per person-month

LOC vs. FP

- Relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design
- Empirical studies show an approximate relationship between LOC and FP

LOC/FP (average)

Assembly language	320	
C		128
COBOL, FORTRAN	106	
C++		64
Visual Basic		32
Smalltalk		22
SQL		12
Graphical languages (icons)	4	

Halstead's Software Science

- Halstead (1977) stated that Software Science is different from computer Science and that Software Science consists of programming tasks.
- A programming task is the act of selecting and arranging a finite number of program 'tokens' which are basic syntactic units distinguishable by a compiler.

Halstead's Software Science

- Tokens can be classified as either operators or operands.
- Primary measures for Halstead's Software Science are:

n_1 = the number of distinct operators that appear in a program

n_2 = the number of distinct operands that appear in a program

N_1 = the total number of operator occurrences

N_2 = the total number of operand occurrences

Halstead's Software Science

- Based on the four primitives a system of equations describing the program can be applied:

- Total Vocabulary

$$\text{Vocabulary } (n) = n_1 + n_2$$

- Overall Program Length

$$\text{Length } (N) = N_1 + N_2$$

- Potential Minimum Volume (in bits) for algorithm

$$\text{Volume } (V) = N \log_2(n_1 + n_2)$$

Halstead's Software Science

- Program Level (a complexity measure)

$$\text{Level (L)} = n_1 + n_2$$

- Program Difficulty

$$\text{Difficulty (D)} = (n_1/2) \times (N_2/n_2)$$

- Development Effort

$$\text{Effort} = V / L$$

- Projected number of faults

$$\text{Faults (B)} = V / S$$

where S equals a mean number of decisions between errors (3000 used as default)

Halstead's Software Science

Halstead Example #1

$A = B + C$

Operators (n1)	2
Operands (n2)	3
Operator Occurences (N1)	2
Operand Occurences (N2)	3
Vocabulary (n)	5.00
Length (N)	5.00
Volume (V)	11.61
Level (L)	1.00
Difficulty (D)	1.00
Effort (E)	11.61
Faults (B)	0.00

Halstead Example #2

$B = (X * Y) + (3 * Z)$
 $C = (J + 7) / (M + N)$
 $A = B + C$

Operators (n1)	4
Operands (n2)	9
Operator Occurences (N1)	10
Operand Occurences (N2)	11
Vocabulary (n)	13.00
Length (N)	21.00
Volume (V)	77.71
Level (L)	0.41
Difficulty (D)	2.44
Effort (E)	189.96
Faults (B)	0.03

Halstead's Software Science

- Problems with Halstead's Software Science:
- Faults equation is oversimplified. It simply states that the number of faults in a program is a function of its volume.
- Equations do not provide relevant information, only good for comparison
- Data for predictions must be available to perform equations, which means code must already be written.

Cyclomatic Complexity

- McCabe (1976) designed a system to indicate a programs testability and understandability (maintainability).
- A.K.A. McCabes Complexity Index or CPX
- Based on graph theory of cyclomatic number of regions in a (flow) graph
- Represents the number of linearly independent paths comprising a program
- Gives an upper bound to the number of test-cases that would be required for path testing

Cyclomatic Complexity

- General formula:

$$V(G) = e - n + 2p$$

where:

e = number of edges

n = number of nodes

p = number of unconnected parts to the graph

or

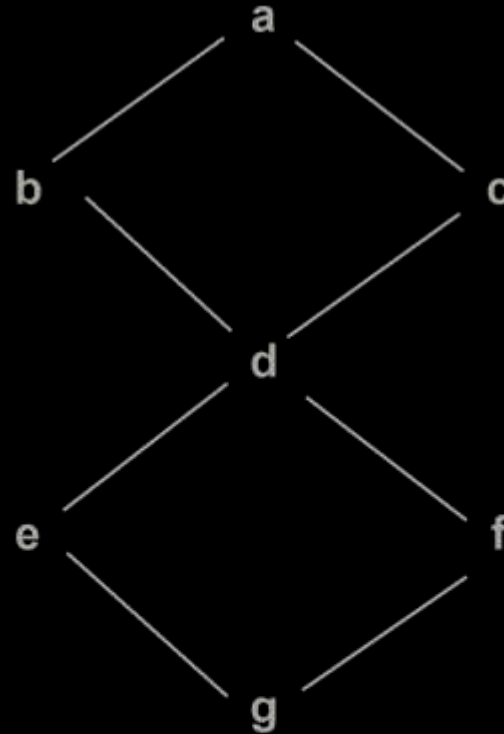
$$V(G) = e - n + 2$$

If all parts are connected

Cyclomatic Complexity

- **Example:**

```
set of processing a;  
If cond1 Then  
    set of processing b  
else  
    set of processing c;  
set of processing d;  
If cond2 then  
    set of processing e;  
else  
    set of processing f;  
set of processing g;
```



edges (e) = 8

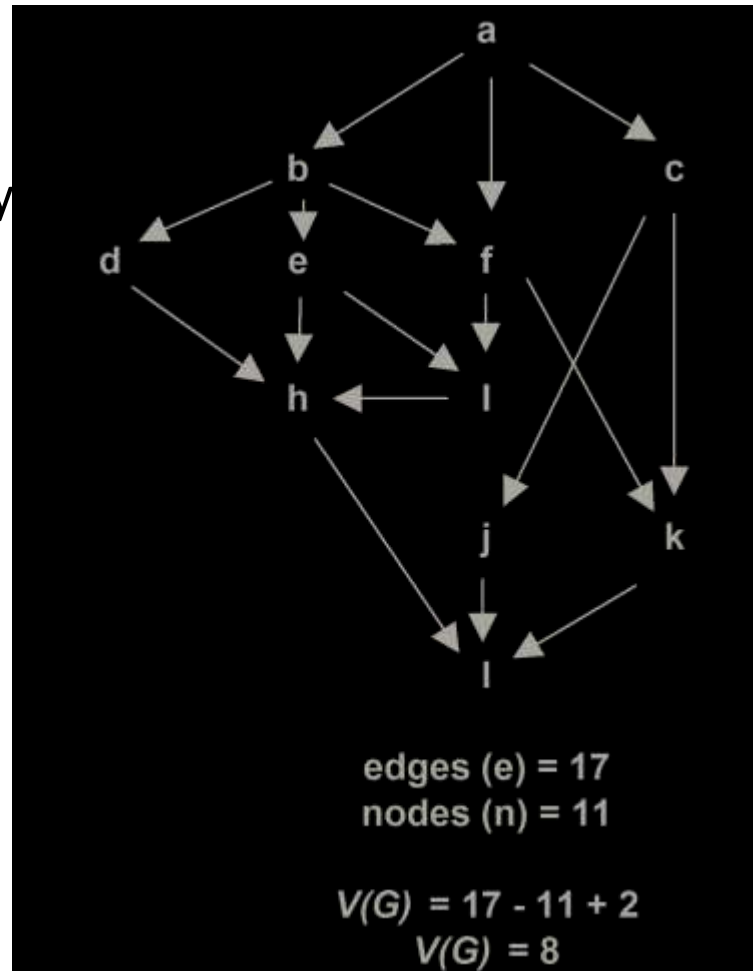
nodes (n) = 7

$$V(G) = 8 - 7 + 2$$

$$V(G) = 3$$

Cyclomatic Complexity

- Example:
More complexity



Cyclomatic Complexity

- Cyclomatic Complexity also can be computed based on the decision in a program:
 - n binary decisions: $V(g) = n + 1$
 - three-way decision: counts as $2n + 1$ binary decisions
 - n -way decision: counts as $n - 1$ binary decisions
 - loops: count as 1 binary decision
 - note: does not distinguish between different type of control flow (eg. loops vs. IF-THEN-ELSE)
- Cyclomatic complexities are additive
 - The Cyclomatic Complexity of one large graph is the sum of the individual graph's complexities.

Cyclomatic Complexity

- McCabes recommendation:
 - To achieve a good testability and maintainability, no program module should have a Cyclomatic Complexity greater than 10.
- Cyclomatic complexity correlates strongly with program size (LOC).
- There also tends to be a positive correlation between Cyclomatic Complexity and defects.
- Cyclomatic Complexity appeals to many software developers because it is tied to decisions and branches (logic).

Syntactic Constructs

- Studies that look at syntactic makeup of a program.
 - Shen (1985)
 - Found a correlation between the number of unique operands and the presence of defects
 - Lo (1992)
 - Found correlation between field defects in modules, and LOC, IF-THEN-ELSE's, DO-WHILE's, unique operands, and number of calls
 - DO-WHILE turned out to be the greatest factor, and a positive correlation between DO-WHILE use and defects was discovered (programmers needed training)

Structure Metrics

- Previous metrics all assume that each module is a independent entity.
- Structure metrics take into account that (significant) interactions between modules exist.
- Yourdon/Constantine (1979) & Myers (1978) both proposed fan-in and fan-out metrics based on the idea of coupling.

Structure Metrics

- Fan-out = number of modules called
- Fan-in = number of modules called by
- Module Coupling = “connectedness”
- Other Factors:
 - Inputs, Outputs, and Global Variables
- Low Module Coupling = relatively low number of inputs, outputs, and calls
- High Coupling = relatively high number of inputs, outputs, and calls

Structure Metrics

- Small/Simple modules are expected to have high fan-in. These modules are usually located at the lowest levels of the system structure.
- Large/Complex modules are expected to have low fan-in.
- Modules should generally not have both high fan-in and high fan-out. If so - then module is good candidate for redesign (functional decomposition).

Structure Metrics

- Modules with high fan-in are expected to have low defect levels.
 - Fan-in is expected to have negative or insignificant correlation with defects
- Modules with high fan-out are expected to have higher defect levels.
 - Fan-out is expected to have positive correlation with defects

Structure Metrics

- Card and Glass (1990) System Complexity Model:
- System Complexity is equal to the sum of structural complexity plus data complexity
 - where:
 - structural complexity is equal to mean of squared values of fan-out (fan-in is insignificant)
 - data complexity is average I/O variables

Complexity Metrics and Models

Criteria for Evaluation

- Explanatory Powers - the metrics/model ability to explain the interrelationships among complexity, quality, and other programming and design parameters.
- Applicability - the degree to which the metrics/models can be applied by software engineers to improve the quality of design, code and test.

The Goal Question Metrics (GQM) Approach

- The GQM approach is based on the idea that in order to measure in a meaningful way we must measure that which will help us to assess and meet our organizational goals.
 - A Goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. The objects of measurement are products, processes and resources.
 - Questions are used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model. Questions try to characterize the object of measurement with respect to a selected quality issue and to determine its quality from the selected viewpoint. Questions must be answerable in a quantitative manner.
 - Metrics are associated with every question in order to answer it in a quantitative way.

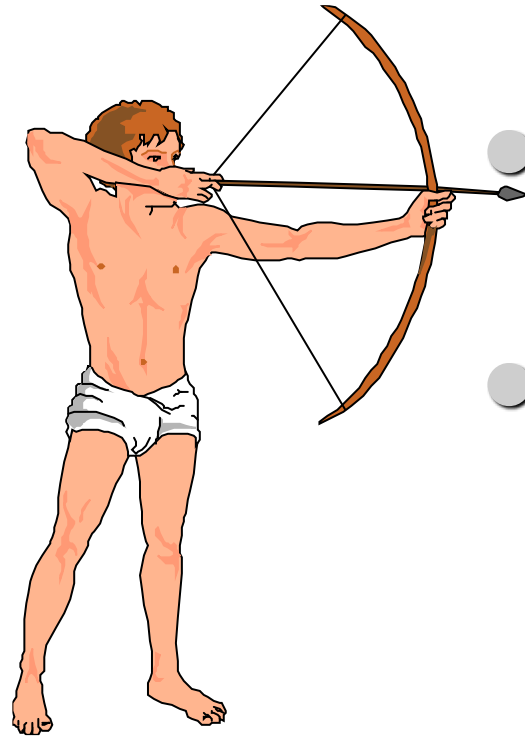
The Goal Question Metrics (GQM) Approach

- **Example:**
 - Goal: To improve the outcome of design inspections from the quality managers point of view.
 - Question: What is the current yield of design inspections?
 - Metric: $\text{inspection yield} = \text{nr defects found} / \text{est. total defects}$
 - Question: What is the current inspection rate?
 - Metric: individual inspection rate, inspection period, overall inspection rate
 - Question: Is design inspection yield improving?
 - Metric: $\text{current design inspection yield} / \text{baseline design inspection yield} * 100$
 - ...

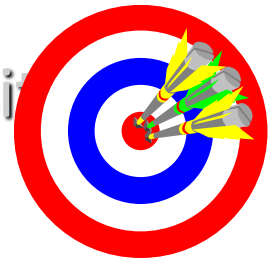
II. SOFTWARE QUALITY ASSURANCE

- Quality tasks – SQA plan – Teams – Characteristics – Implementation – Documentation
- – Reviews and Audits

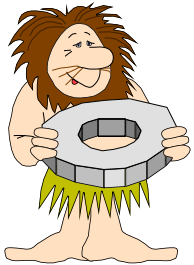
Quality Assurance



- Controlling the process has a positive effect on the product.
- Once one quality outcome has been achieved using a particular series of steps, by repeating those steps and controlling variation, the outcome will also be repeated



How was this principle established and validated?



Ever since humans began producing things, particularly for other people, Quality has been important.

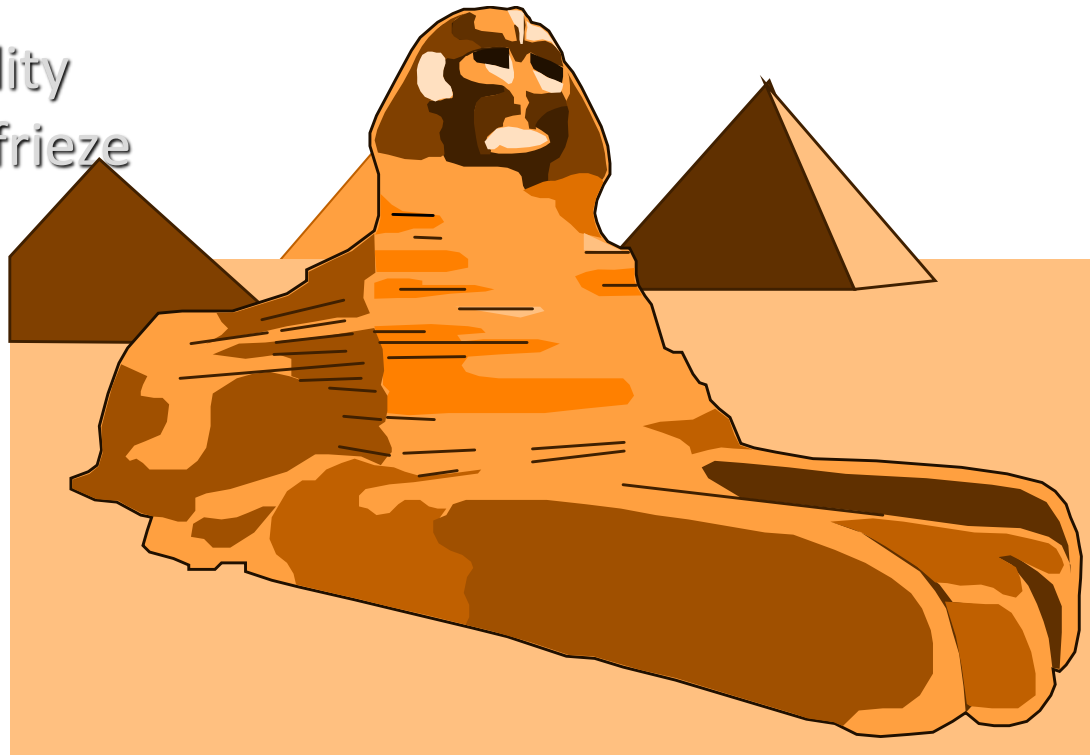
The principle was established through trial and error.



It appears to make logical sense

History of Quality

The earliest example of quality inspection is inscribed on a frieze at Thebes dated 1450 B.C.



Quality Control v Quality Assurance

CONTROL

- Product focused
- Method is Inspection and Test against a Spec.
- Quality Achieved by rework and rejection

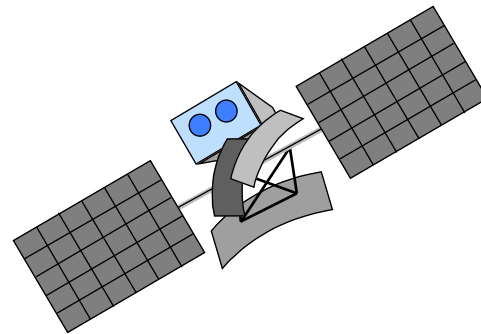
ASSURANCE

- Process focused
- Method is Audit against Procedures
- Quality Achieved by doing things the right way

In 1979, the first NASA probe to Venus missed the planet and plunged into the Sun because a full stop was used instead of a comma in a Fortran routine.

Result = hundreds of millions of dollars and
countless manhours lost.

Wallmüller 1994



Peer inspections

- Peer reviews are over hyped as key aspect of quality system. They are valuable secondary mechanisms
- Inspections are good vehicle for holding authors accountable for quality products
- A critical component deserves to be inspected by several people who have stake in quality, performance and feature
- Only 20% of artifacts(use cases, design models, source code,test cases) deserve detailed scrutiny
- Quality assurance is everyone's responsibility and should be integral to almost all process activities

Software Quality Assurance

Although there are many definitions of quality but for our purposes, software quality is:

- Conformance to ...
 - the explicitly stated functional & performance requirements,
 - explicitly documented development standards &
 - implicit characteristics that are expected of all professionally developed software
- This definition emphasizes on 3 important points
 - S/W requirements – a foundation from which quality is measured
 - Standards – define development criteria against which S/W is engineered
 - Implicit requirements – often go unmentioned but if not met, can cause suspicion in quality

Who does it?

- Prior to 20th Century
 - SQA was responsibility of the craftsperson
- During 1950s and 1960s
 - Responsibility of programmer
- Today responsible ones are ...
 - S/W Engrs. (*Apply technical methods & measures, Conduct FTRs & perform planned testing*)
 - Project managers
 - Customers
 - Sales Person
 - SQA group (*Serves as customer's in-house representative, Looks at S/W from customer's point of view, Assists the S/W Engrs team to achieve quality*)

SQA

- SQA is an activity that is applied throughout the software process and not after the software has been developed
- SQA covers the following
 - Quality management approach
 - Effective s/w engineering technology (methods & tools)
 - Formal technical reviews (applied throughout the process)
 - A multi-tiered testing strategy
 - Control of software documentation & changes made to it
 - A procedure to assure compliance with software development standards
 - Measurement & reporting mechanism

SQA Group & SQA Activities

- SQA group is responsible for QA planning, oversight, record keeping, analysis and reporting
- SEI recommends the following set of SQA group activities:
 - Prepares an SQA plan for project
 - Participates in the development of project's software process description
 - Reviews s/w engg activities to verify compliance with defines s/w process
 - Audits designated s/w work products to verify compliance
 - Ensures that deviations in s/w work & work products are documented
 - Records any non compliance & reports to senior management
- SQA groups also participates in change management & help to collect & analyze s/w metrics

Inspection

- One method of detecting errors is inspection.
- There are two major types of inspection.
 - SAMPLING INSPECTION.
 - 100% INSPECTION.

Sampling Inspection

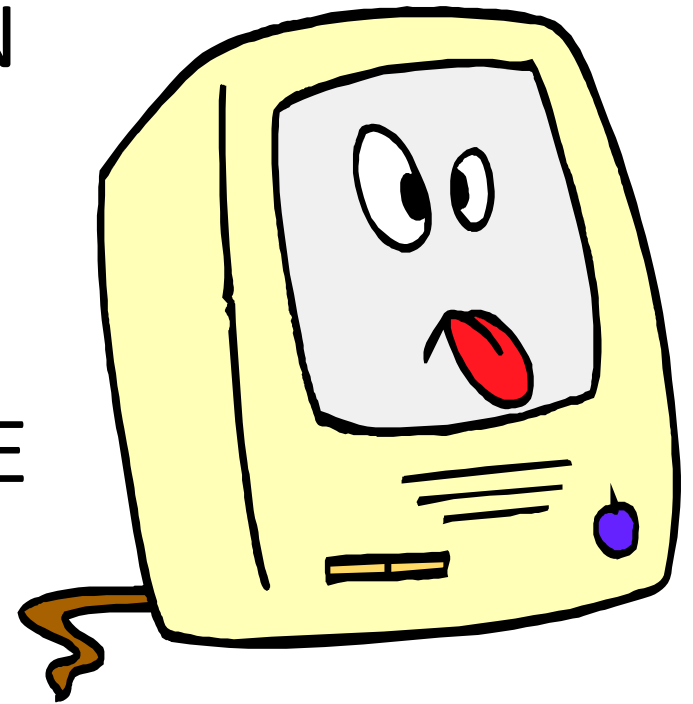
- In some factories, the attitude is:
- “It may take all day to inspect all product”.
- “There may be a few defects, but sampling is the most practical way to check”.

Sampling Inspection - Reflection

- *Think about sampling.*
- With sampling, there is a chance that an error may go undetected.
- Even if 1 part in 1000 is defective, the customer who buys that part has a 100% defective product!

Sampling Inspection - Reflection

SAMPLING INSPECTION
MAKES SENSE ONLY
FROM THE
MANUFACTURER'S
VIEW, NOT FROM THE
CUSTOMER'S!



2. 100% Inspection

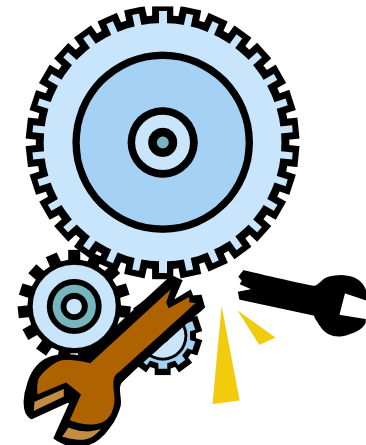
- In the best factories, the attitude is:
- “We won’t tolerate a single defect!”
- “We will organize production so that 100% of the product can be easily inspected”.
- “That makes the most sense”.

100% Inspection - Reflection

- *Think about 100% inspection.*
- Even one defective product is enough to destroy a customer's confidence in a company.
- To stay competitive a company must supply good product in thousands.
- The best way to achieve this is to organize production to inspect 100% of the products.

The User Is The Best Inspector

- No one intends to make mistakes.
- While we are working, errors can show up without us noticing.
- How can we catch these errors before they turn into defective products?



Finding Defects in the Subsequent Process

- We don't expect to find defects, but if a product we use doesn't do what it is supposed to do, then we know it is defective.
- Users are the best at discovering defects!

Finding Defects in the Subsequent Process

- Since subsequent processes are also 'users' of the product being manufactured, they are also expert at finding defects.
- If products are produced in a flow, each product is sent to the next process as soon as it is finished and defects are therefore found immediately.

UNIT III QUALITY CONTROL AND RELIABILITY 9

- Tools for Quality – Ishikawa's basic tools – CASE tools – Defect prevention and removal
- – Reliability models – Rayleigh model – Reliability growth models for quality assessment

What are the Basic Seven Tools of Quality?

- Fishbone Diagrams
- Histograms
- Pareto Analysis
- Flowcharts
- Scatter Plots
- Run Charts
- Control Charts

Where did the Basic Seven come from?

Kaoru Ishikawa

- Known for “Democratizing Statistics”
- The Basic Seven Tools made statistical analysis less complicated for the average person
- Good Visual Aids make statistical and quality control more comprehensible.

FISHBONE DIAGRAM

A diagram showing the main causes and sub-causes of a quality problem or defect.

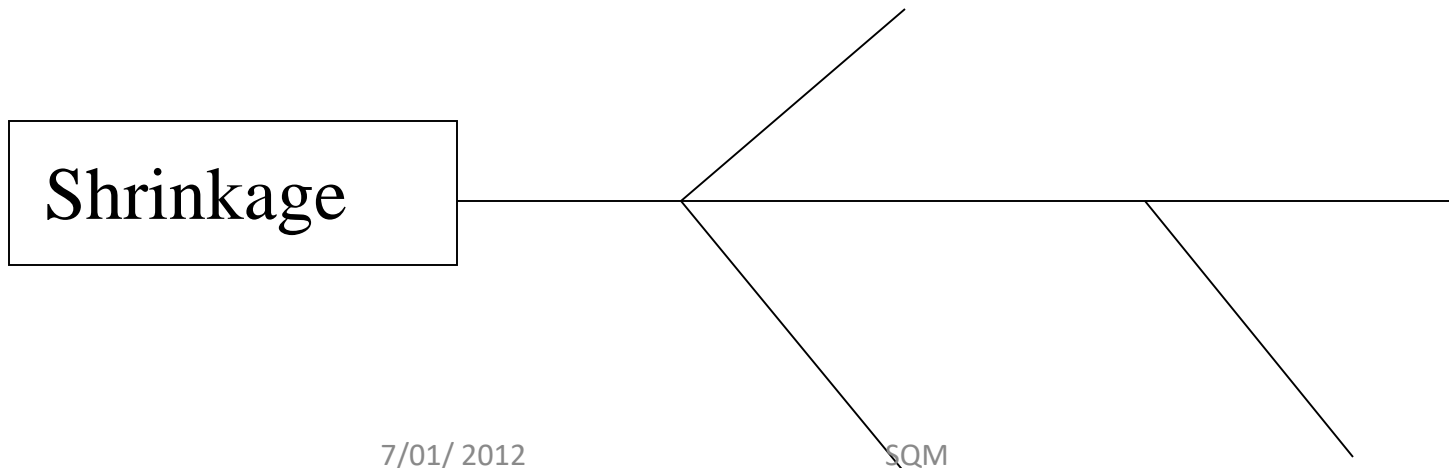
Attributes:

- No statistics involved
- Maps out a process/problem
- Makes improvement easier
- Looks like a “Fish Skeleton”

Constructing a Fishbone Diagram

- Step 1 - Identify the Problem
- Step 2 - Draw “spine” and “bones”

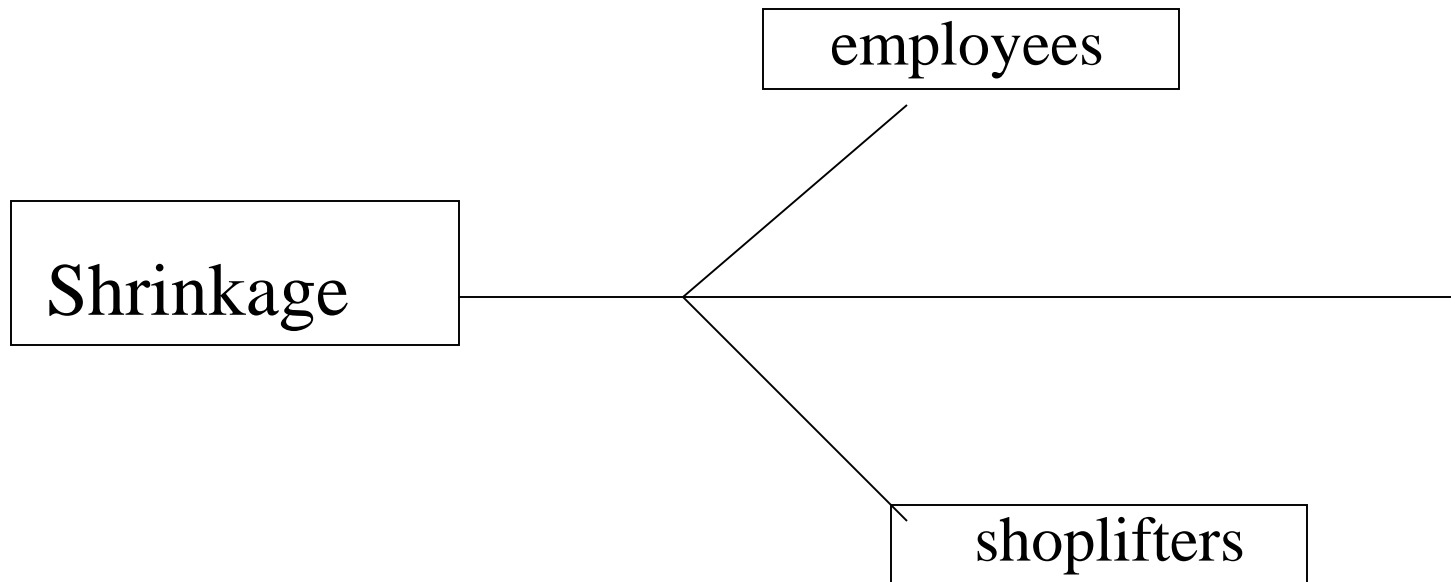
Example: High Inventory Shrinkage at local Drug Store



Constructing a Fishbone Diagram

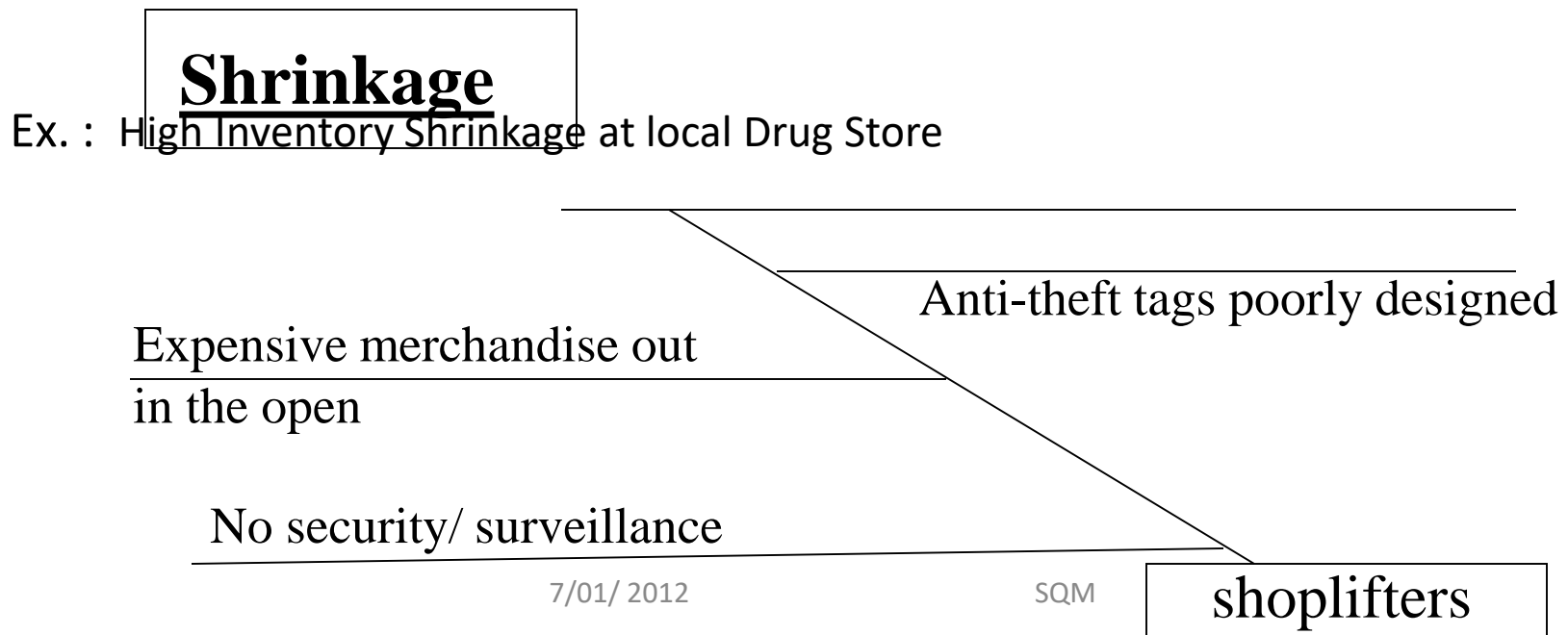
- Step 3 - Identify different areas where problems may arise from.

Ex. : High Inventory Shrinkage at local Drug Store



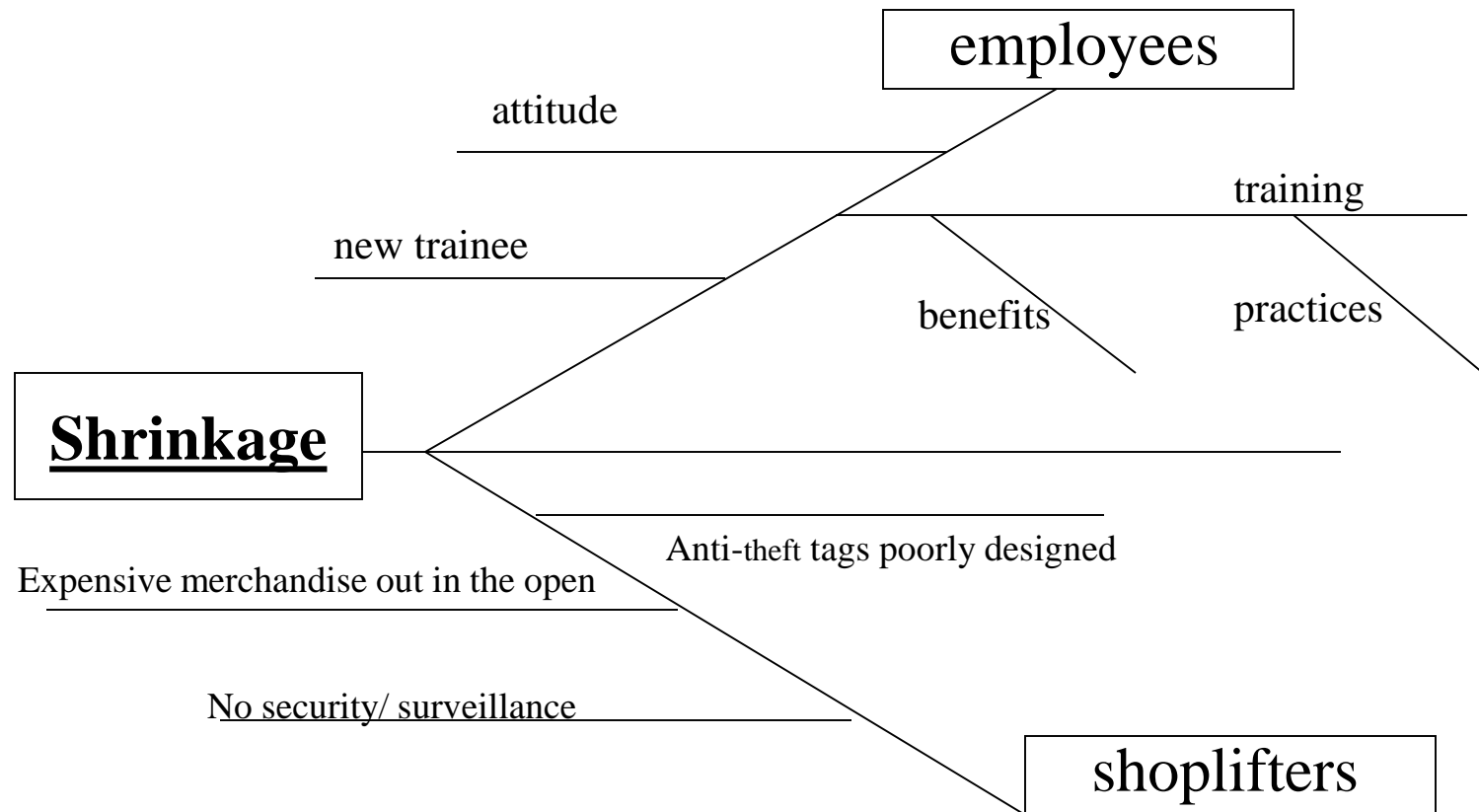
Constructing a Fishbone Diagram

- Step 4 - Identify what these specific causes could be
- Step 5 – Use the finished diagram to brainstorm solutions to the main problems.



Constructing a Fishbone Diagram

- Ex. : High Inventory Shrinkage at local Drug Store



HISTOGRAM

A Bar chart showing the distribution of variable quantities or characteristics.

- Use range to estimate beginning and end
- Calculate the width of each column by dividing the range by the number of columns

$$\frac{\text{Range}}{\text{\# of Columns}} = \text{Width}$$

Acme Pizza Example

- Let's say the owner wants a distribution of Acme's Thursday Night Sales

Data Set from last Thursday(slices)

```
0 2 1 2 2 4 1 3 1 2 1 2 2 4 3 4 1 4 3 2 2 3 2 1 2 2 1 2 2 1 4 2 2 1 2 1 2 2 1 2 1 2 1 2 1 2
1 2 1 2 2 2 1 2 1 2 1 1 2 2 2 3 1 4 2 2 3 2 2 2 1 2 3 2 2 4 2 2 4 4 1 2 2 2 3 2 2 1 2 2 4 2
1 2 4 2 1 7 2 1 2 2 3 1 2 1 1 2 1 2 2 2 1 2 2 1 2 1 2 2 2 4 2 4
```

Acme Pizza Example

Mean = 2.032258

Max = 7

Min = 0

Range = 7

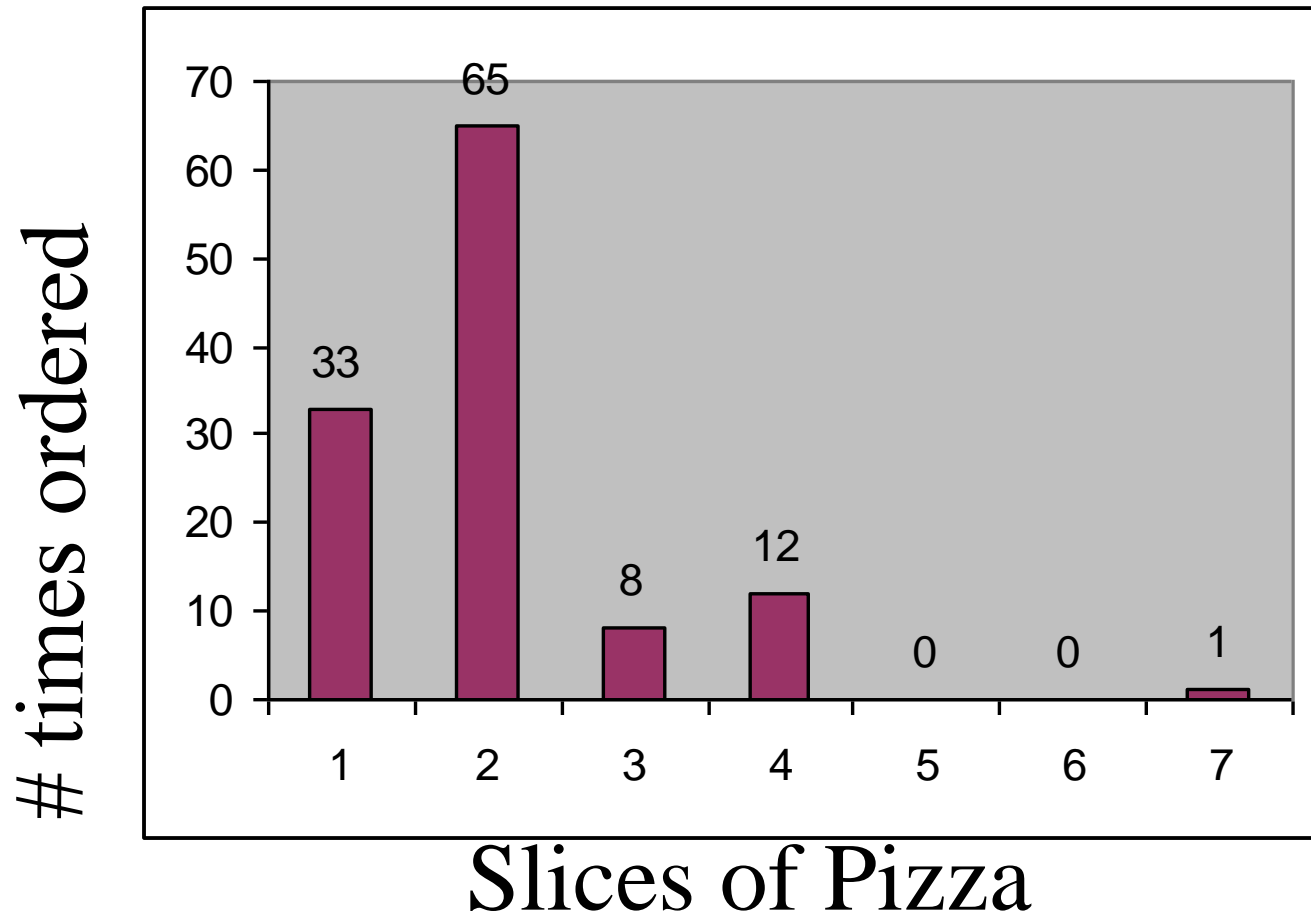
Question

For 7 columns what would the width be?

$\text{Range/Columns} = 7/7 = 1$ slice

Acme Pizza Example

Histogram



PARETO CHART

A chart showing the distribution of effects attributable to various causes or factors arranged from the most frequent to the least frequent.

- Very similar to Histograms
- Use of the 80/20 rule

Use of percentages to show importance

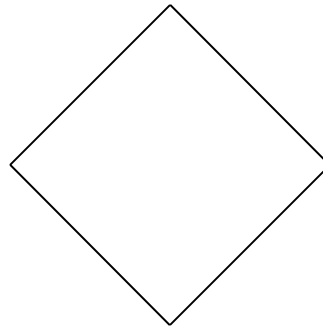
FLOW CHART

A device/method showing the order of activities in a process or project and their interdependency.

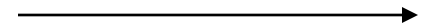
Process



Decision



The process flow

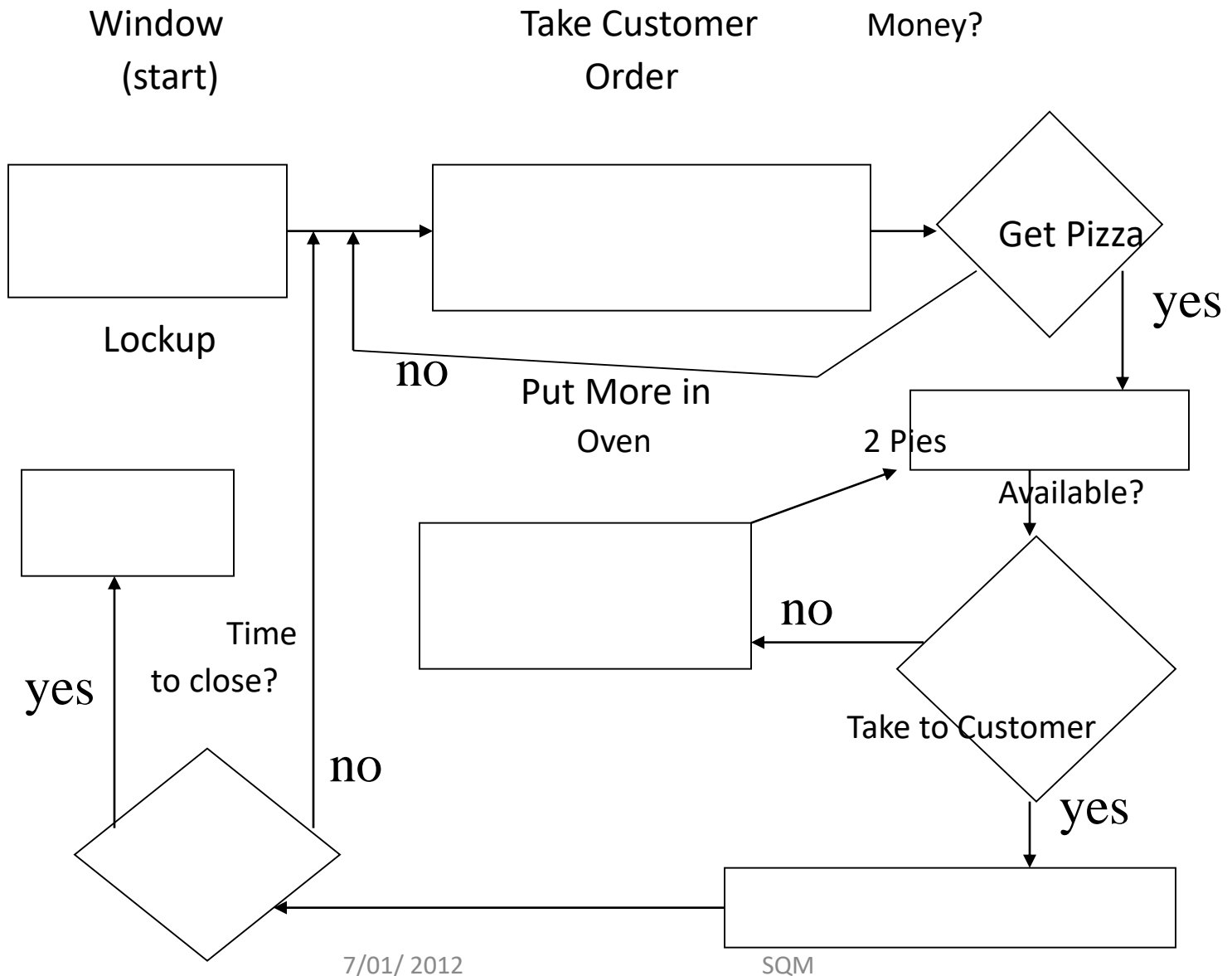


Flowcharts

Don't Forget to:

- Define symbols before beginning
- Stay consistent
- Check that process is accurate

Archie Pizza Example (Flowchart)



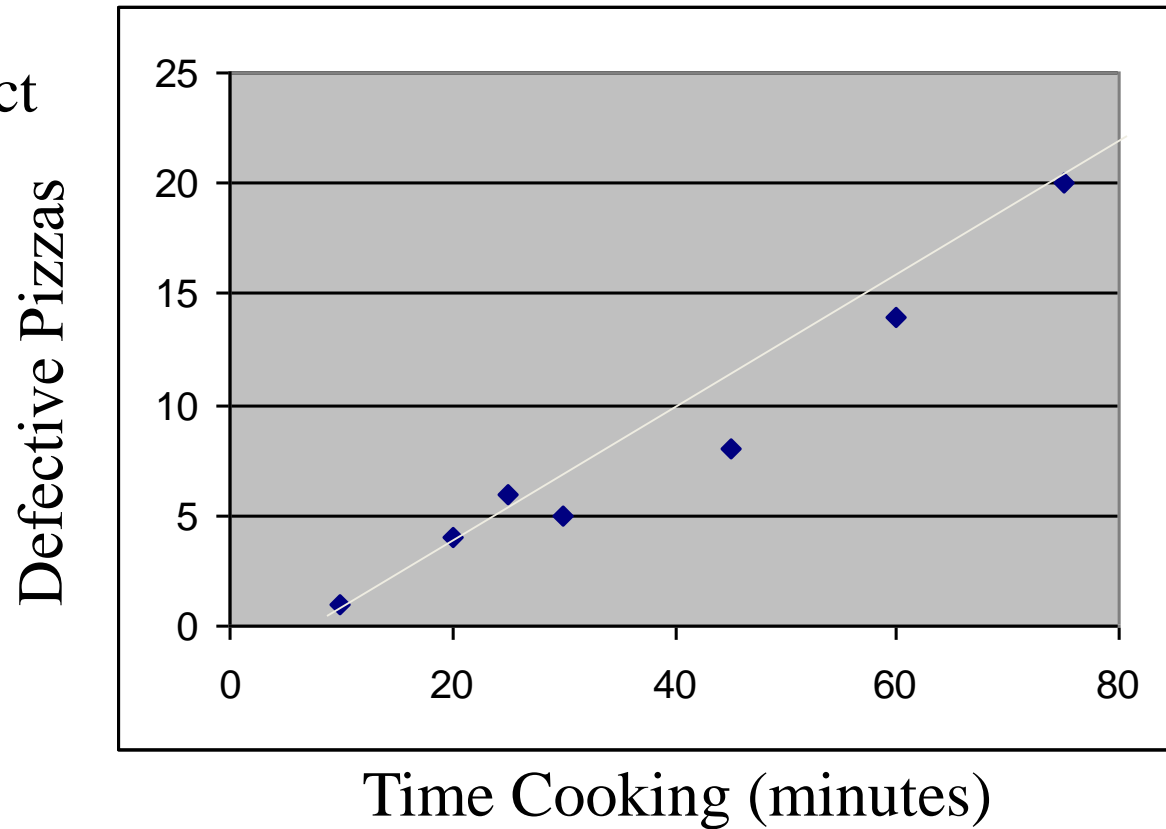
SCATTER DIAGRAM

A diagram showing the pattern that exists in the relationship between two variables 'x' and 'y'.

Scatter diagrams may be used to develop informal models to predict the future based on past correlations.

Scatter Diagrams

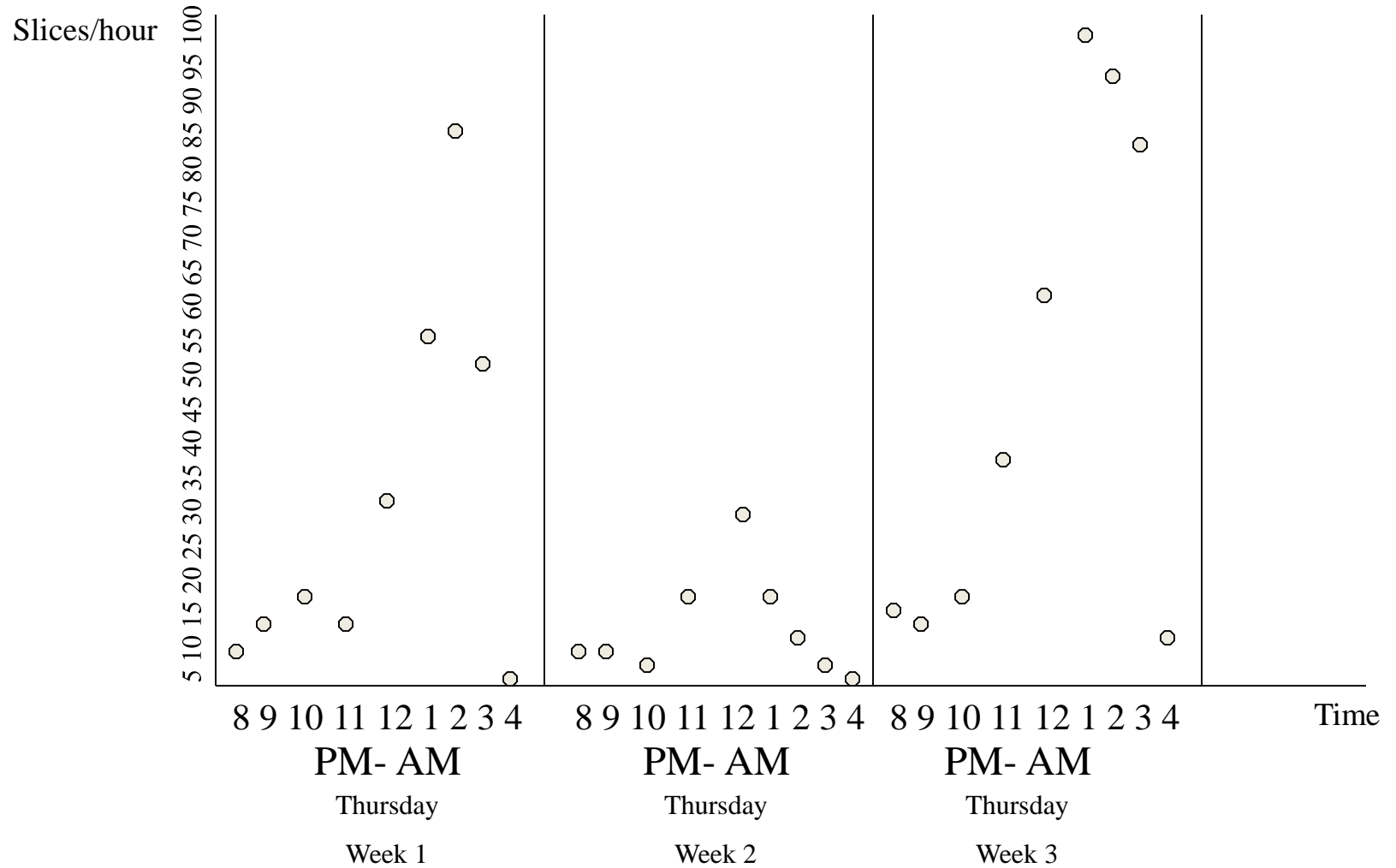
- Easier to see direct relationship



RUN CHART

- A chart showing the history and pattern or variations in time sequence.
- It is plot of data points in time sequence , connected by a line.
- This tool is used at the beginning of the change process to see what the problems are & at the end to see whether the change made has resulted in permanent process improvement.

Run Charts

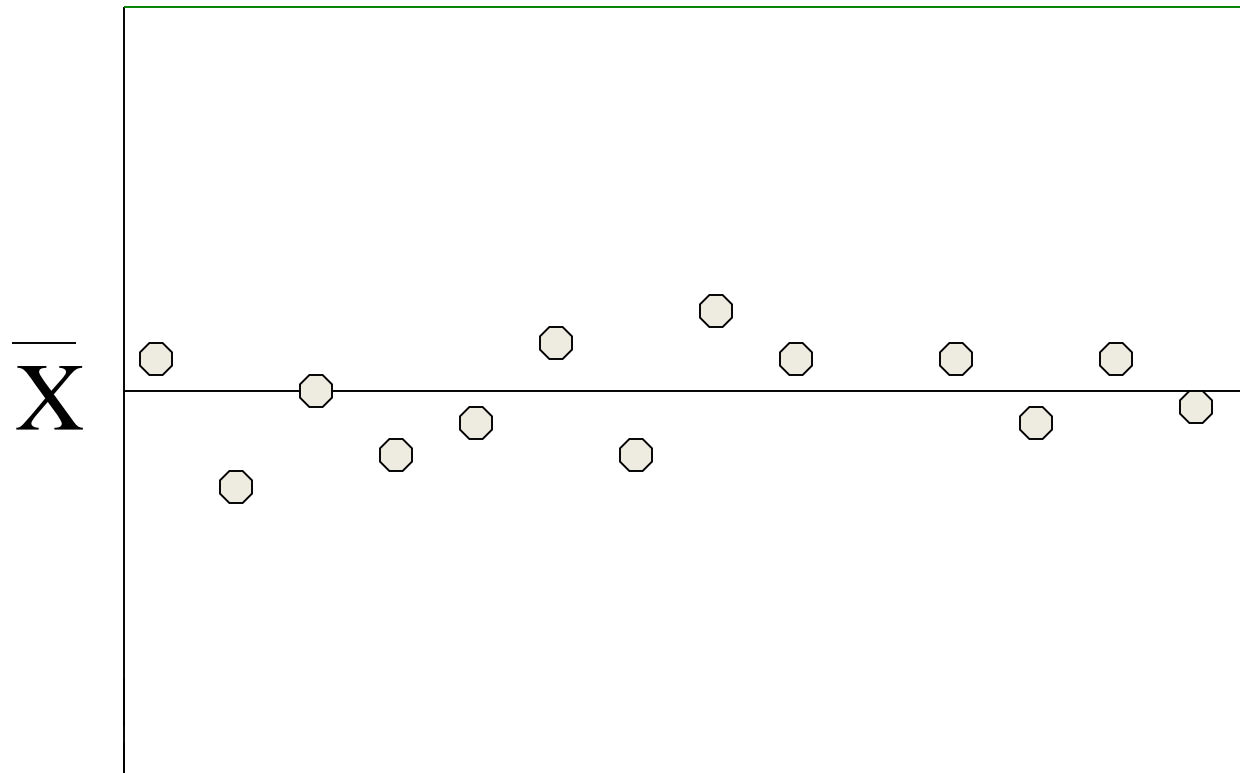


CONTROL CHART

A Line chart showing the control limits at three standard deviations above and below the average quality level. It constitutes the core of SPC.

Control Chart

Upper Limit



Lower Limit

Unacceptable
deviation

SQM

Control Charts

Acme Pizza Management wants to get in on the control chart action

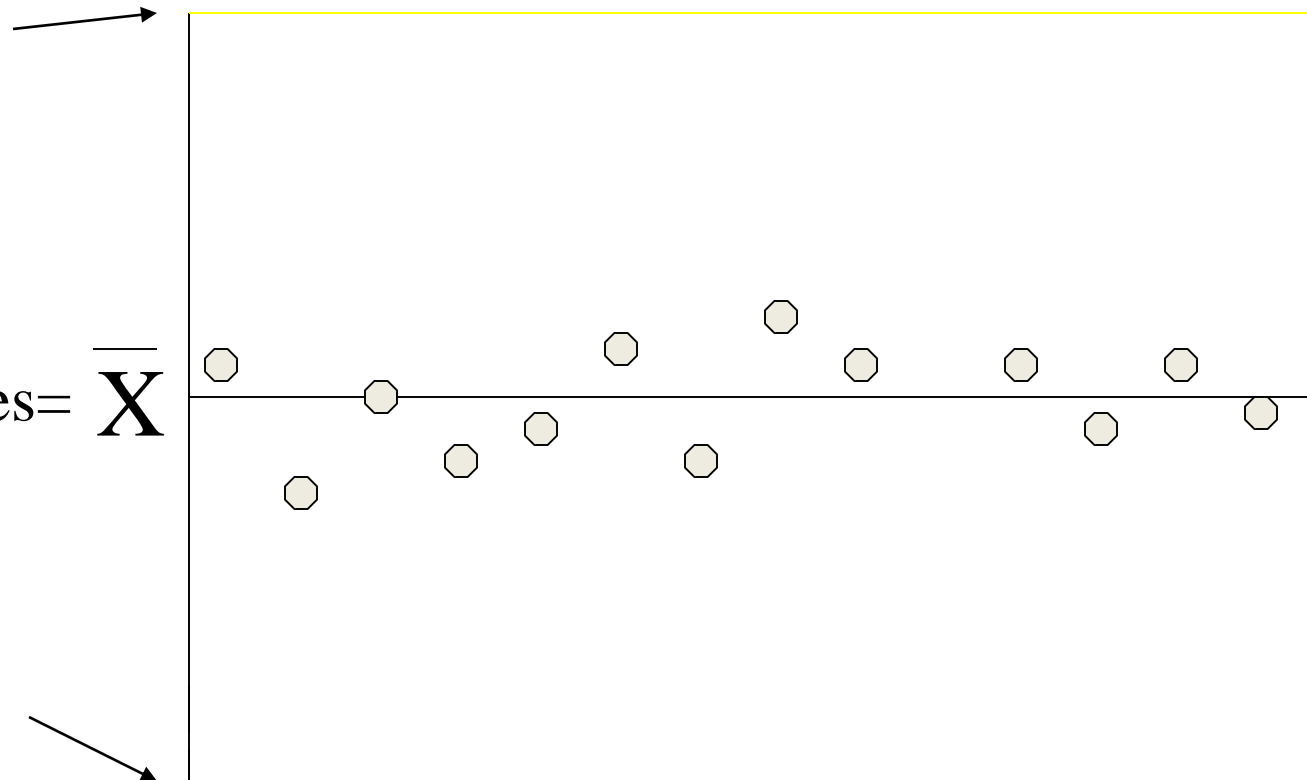
- Average Diameter = 16 inches
- Upper Limit = 17 inches
- Lower Limit = 15 inches

Acme example Control Charts

Upper Limit
17 inches

16 inches = \bar{X}

Lower Limit
15 Inches



Small Pie



Acme example #50

Control Charts

- Pies within specifications were acceptable
- One abnormally small pie is “uncommon”
- Should be examined for quality control

GENICHI TAGUCHI'S METHOD

- Born in Japan, 1924
- Electrical Engineer
- Worked during 1950's to improve Japan's post-WWII telephone communication system
- Father of the “Taguchi Method” and “Robust Engineering”

Don't run away!



- Not a mathematician?
- You can still successfully apply Taguchi Method concepts to your service business.
- Basic concepts are simple.
- Just keep reading.

Competitive Edge 101:

- “In the next century, the capability of developing robust technology will be essential to the competitiveness of any manufacturing enterprise.”
(Tsai) (Taguchi: p xi)
- Substitute “robust services” and “service enterprise.”
You need this.

Seven Quality Tools

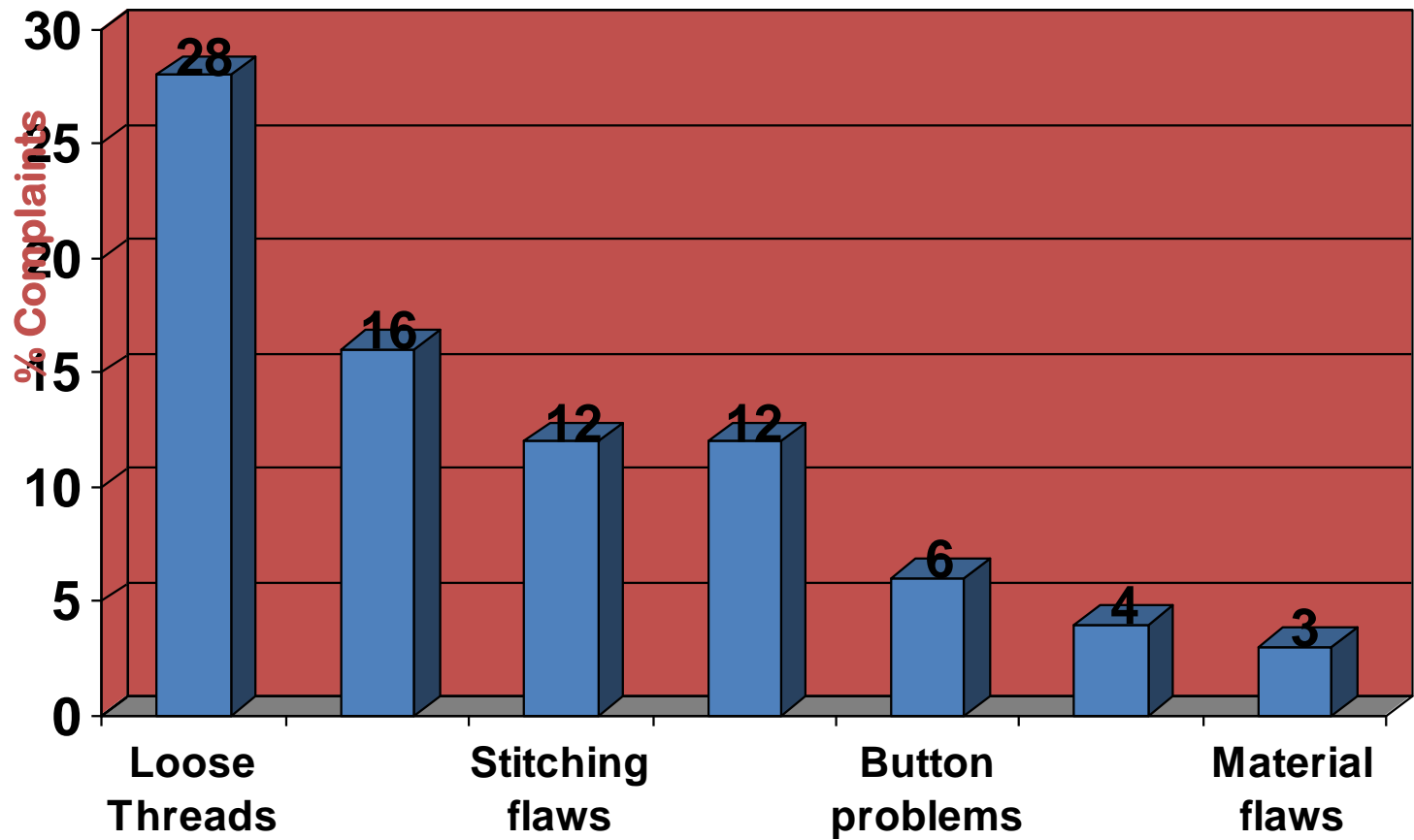
Seven Quality Control Tools

- Pareto Chart
 - Histogram
- Process flow diagram
- Check sheet
- Scatter diagram
- Control chart
- Run Chart
- Cause and Effect Diagram

Pareto Principle

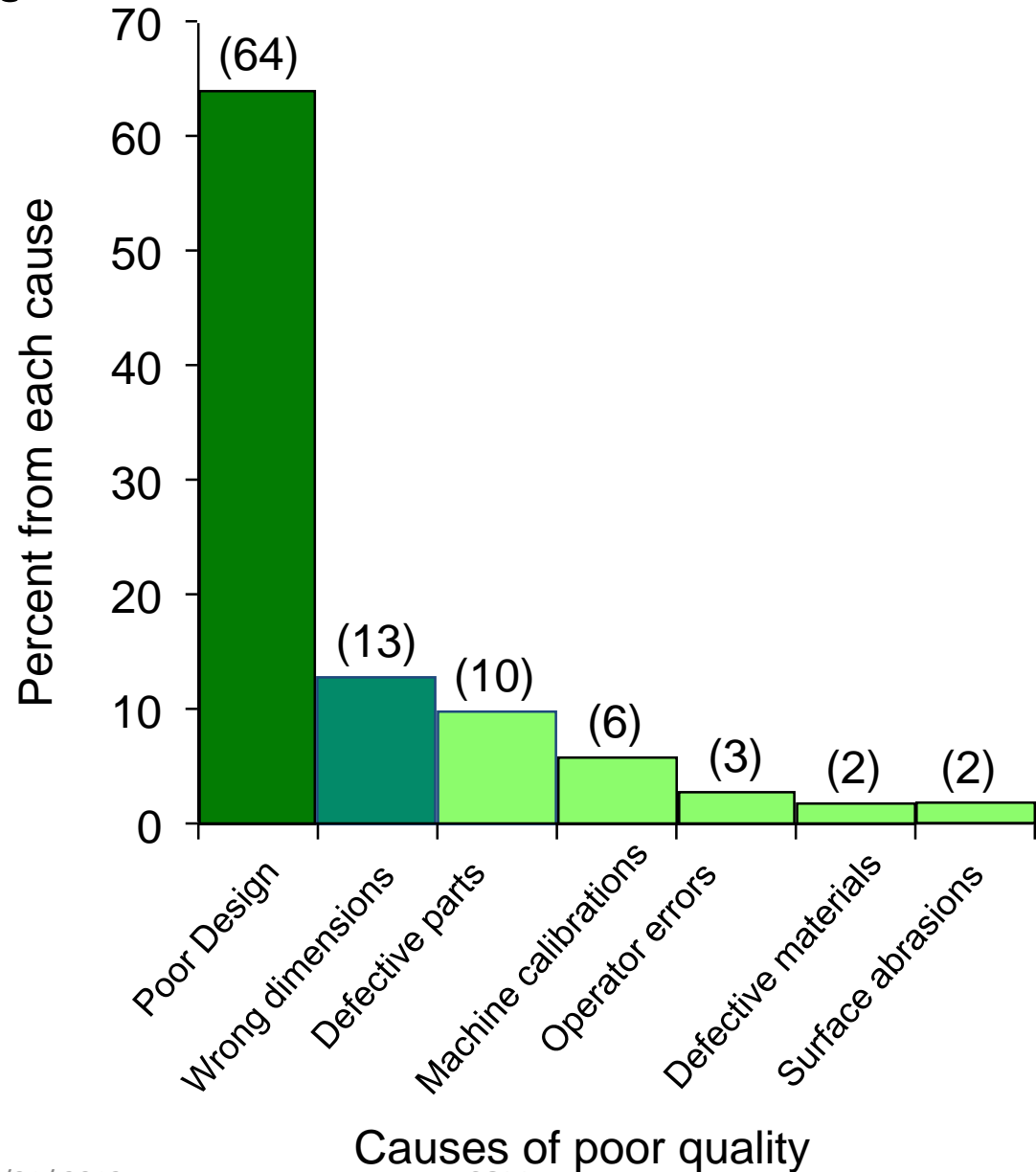
- Vilfredo Pareto (1848-1923) Italian economist
 - 20% of the population has 80% of the wealth
- Juran used the term “vital few, trivial many”. He noted that 20% of the quality problems caused 80% of the dollar loss.

Pareto chart

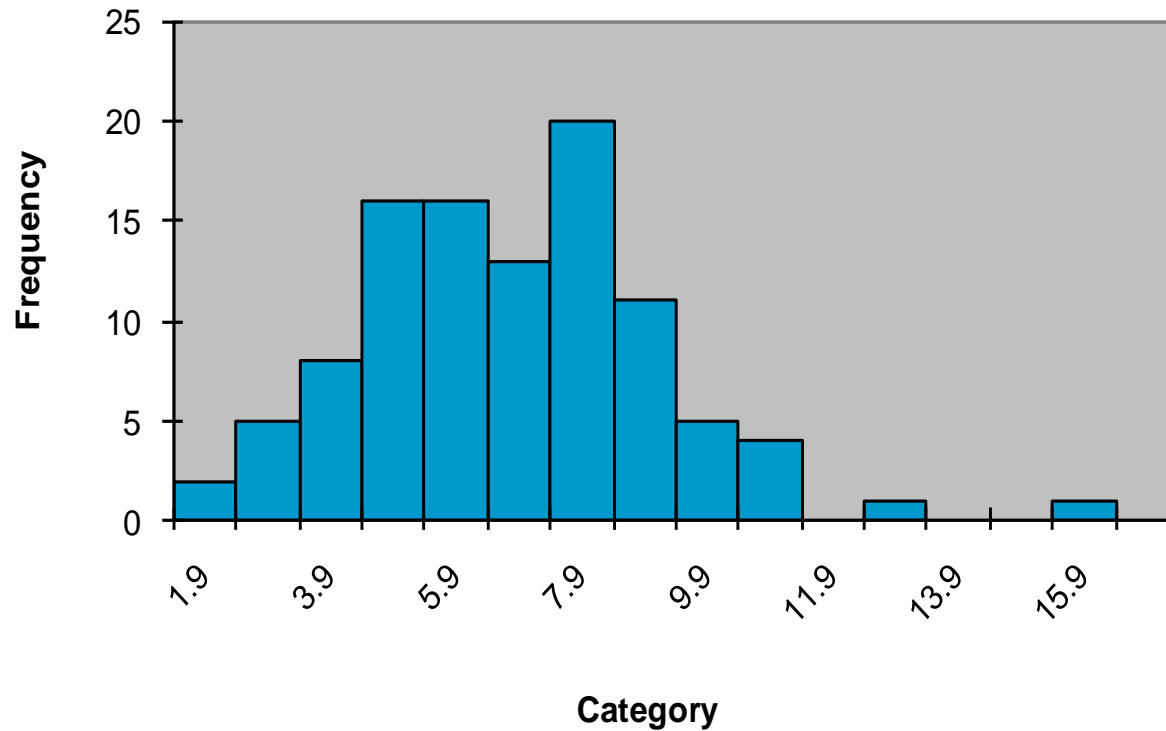


7 Quality Tools

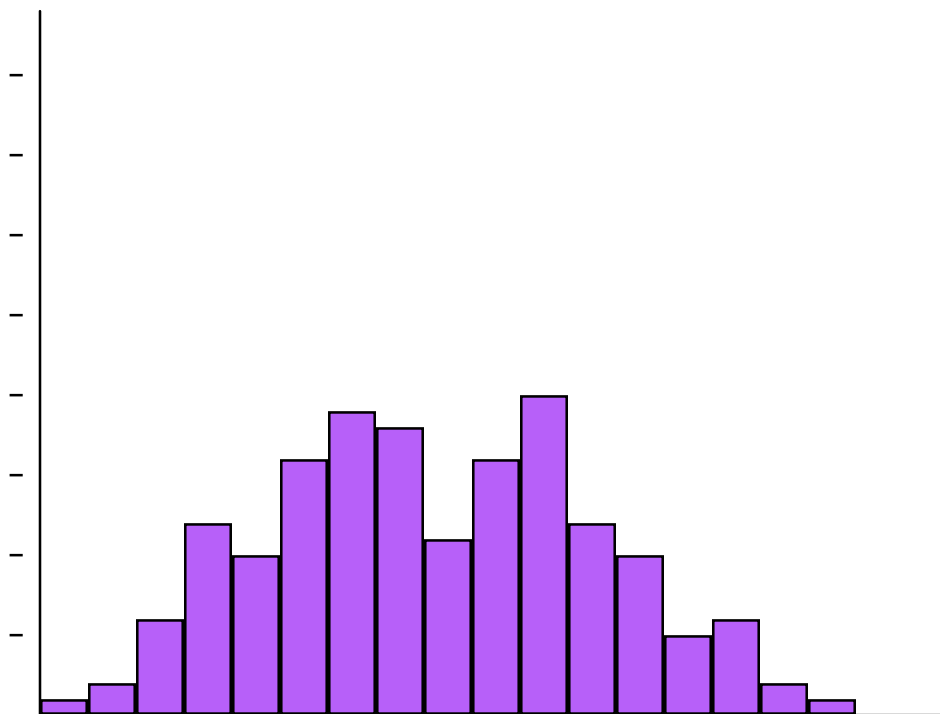
Pareto Chart



Histogram



Histogram



Flowcharts

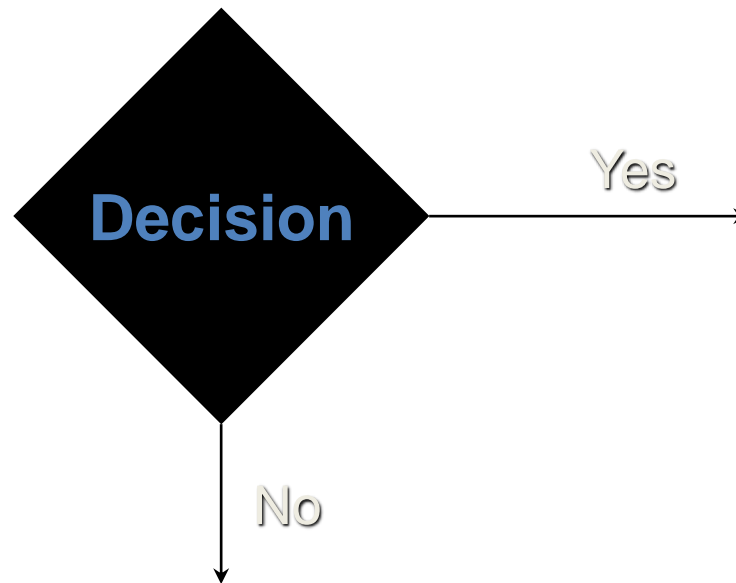
- Flowcharts
 - Graphical description of how work is done.
 - Used to describe processes that are to be improved.

Flow Diagrams

" Draw a flowchart for whatever you do. Until you do, you do not know what you are doing, you just have a job."

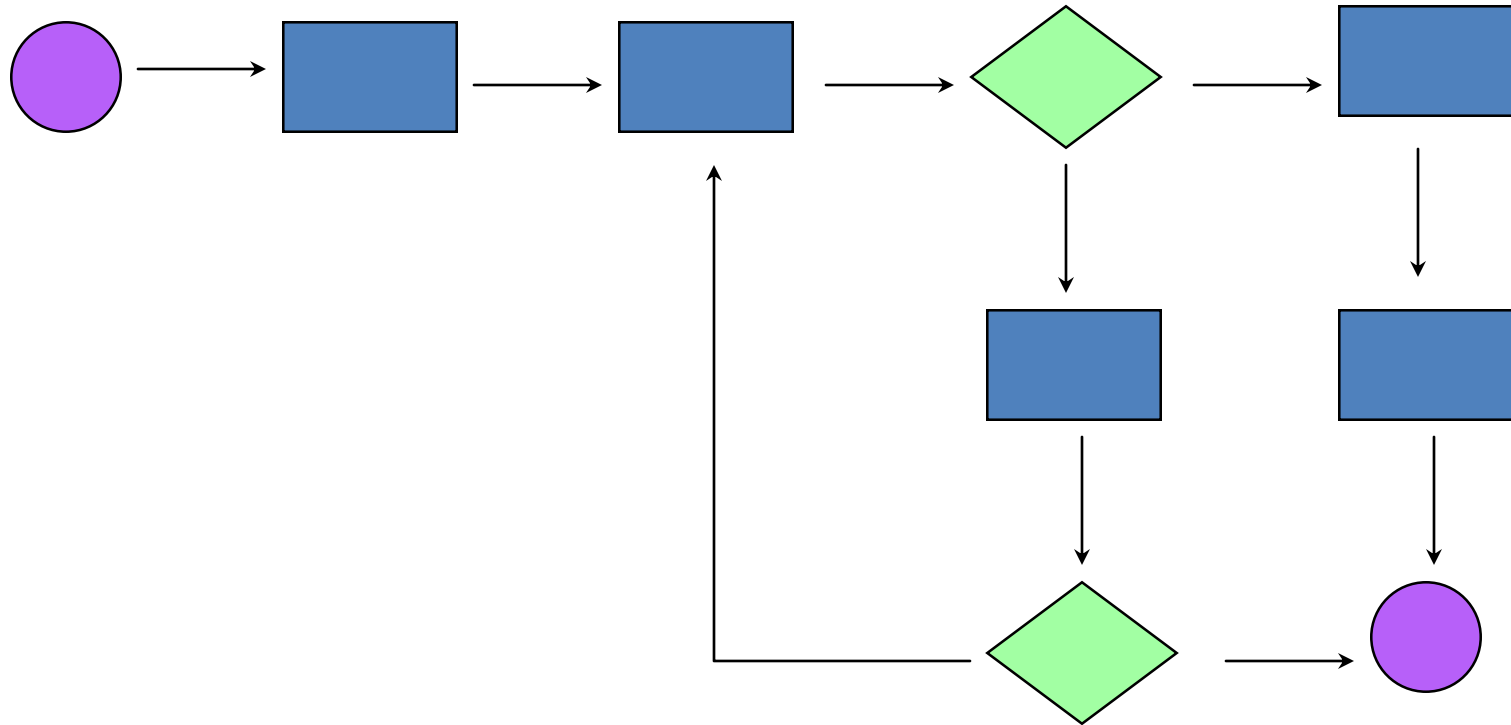
-- Dr. W. Edwards Deming.

Flowchart

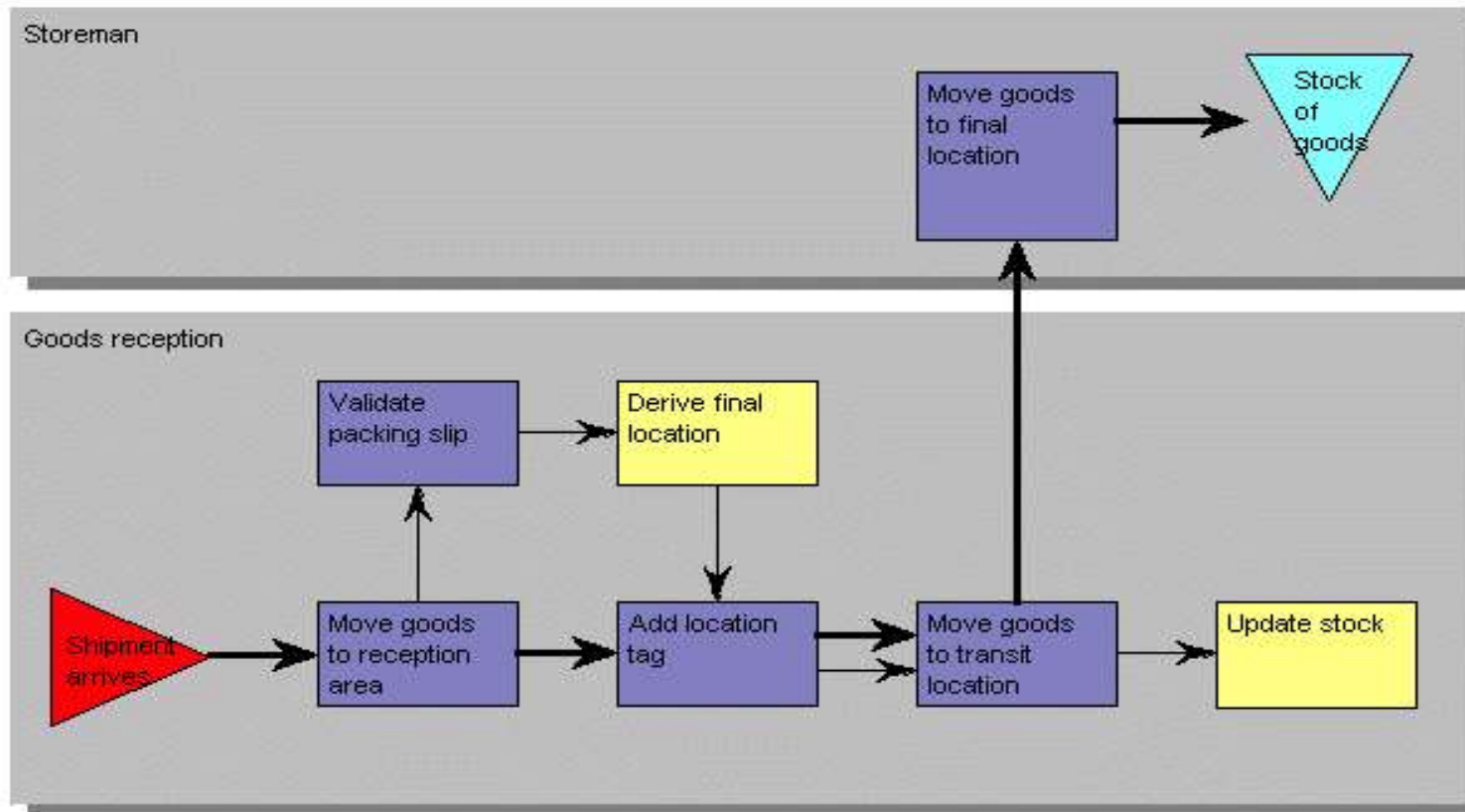


7 Quality Tools

Flowchart



Flow Diagrams

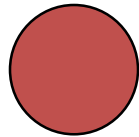


The thicker flow is the flow of material.

The thinner flow is the flow of operation/information.

Activities can be color coded: yellow means IT support, blue means manual activities.

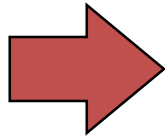
Process Chart Symbols



Operations



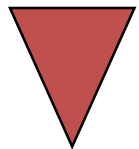
Inspection



Transportation



Delay



Storage

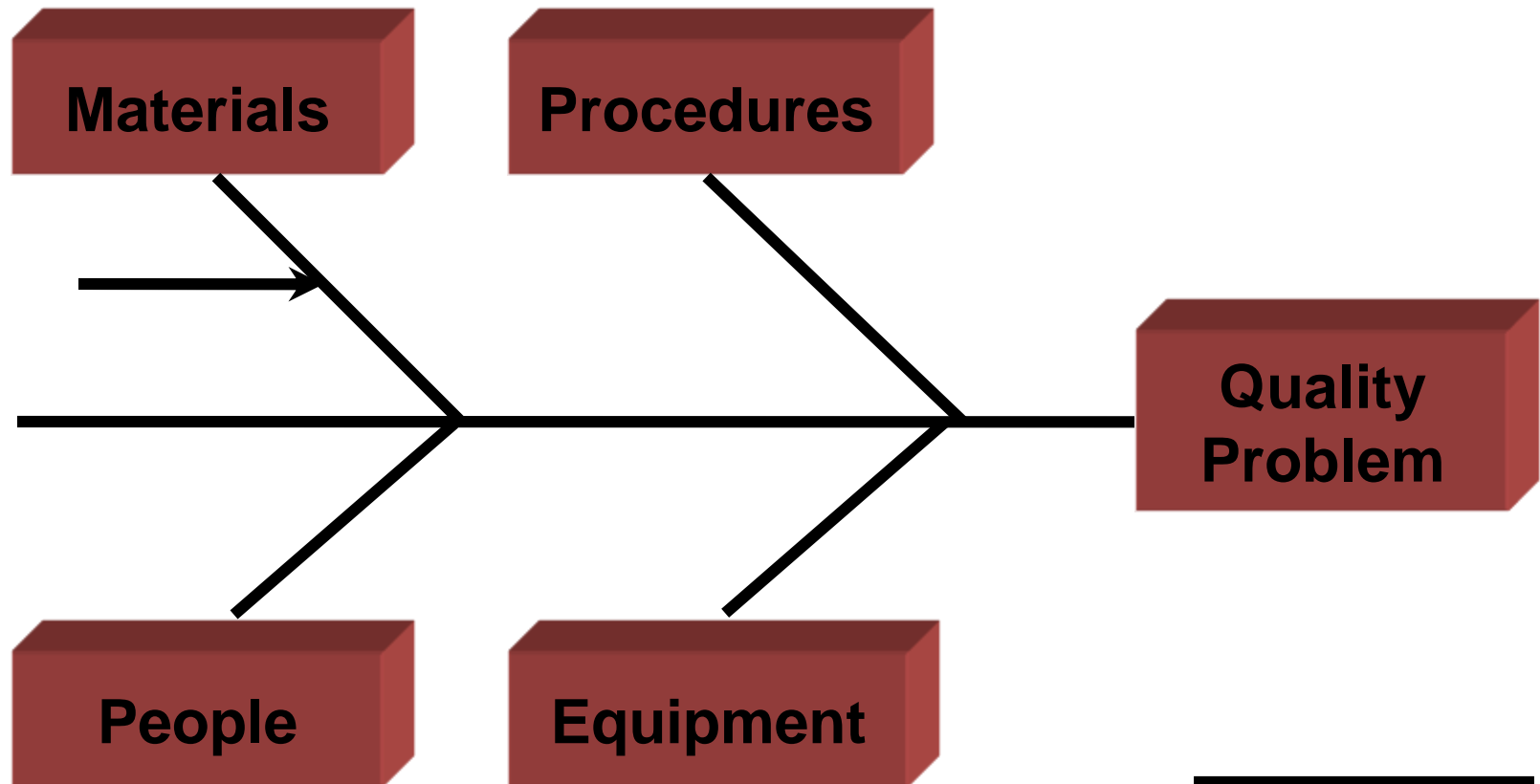
Check Sheet

Defect Type	Shifts			
	✓ ✓ ✓	✓ ✓ ✓ ✓	✓	✓ ✓ ✓
	✓ ✓	✓ ✓ ✓		
		✓ ✓ ✓ ✓		✓ ✓ ✓
		✓ ✓	✓	

Cause-and-Effect Diagrams

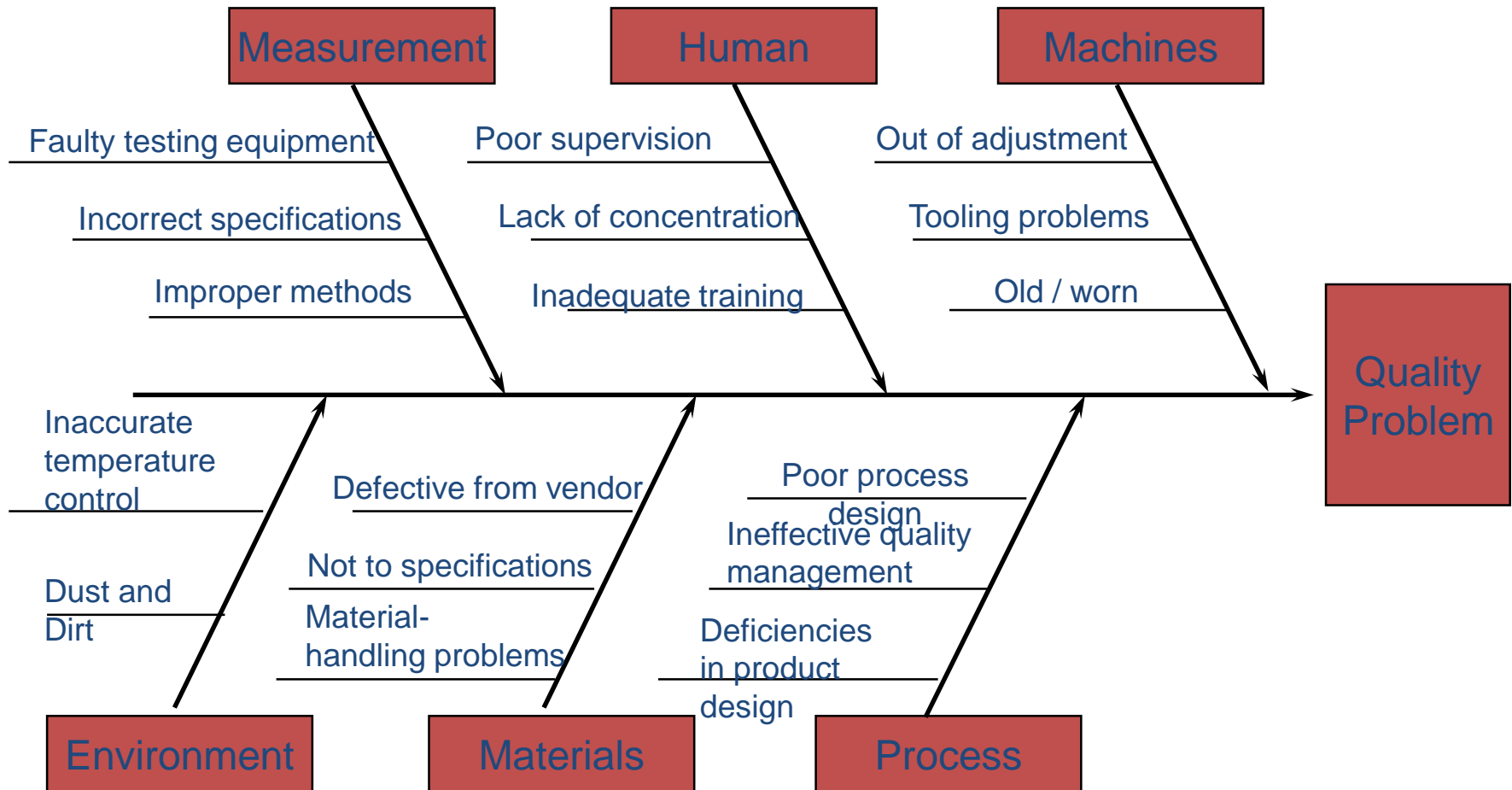
- Show the relationships between a problem and its possible causes.
- Developed by Kaoru Ishikawa (1953)
- Also known as ...
 - Fishbone diagrams
 - Ishikawa diagrams

Cause and Effect “Skeleton”



7 Quality Tools

Fishbone Diagram



Cause and effect diagrams

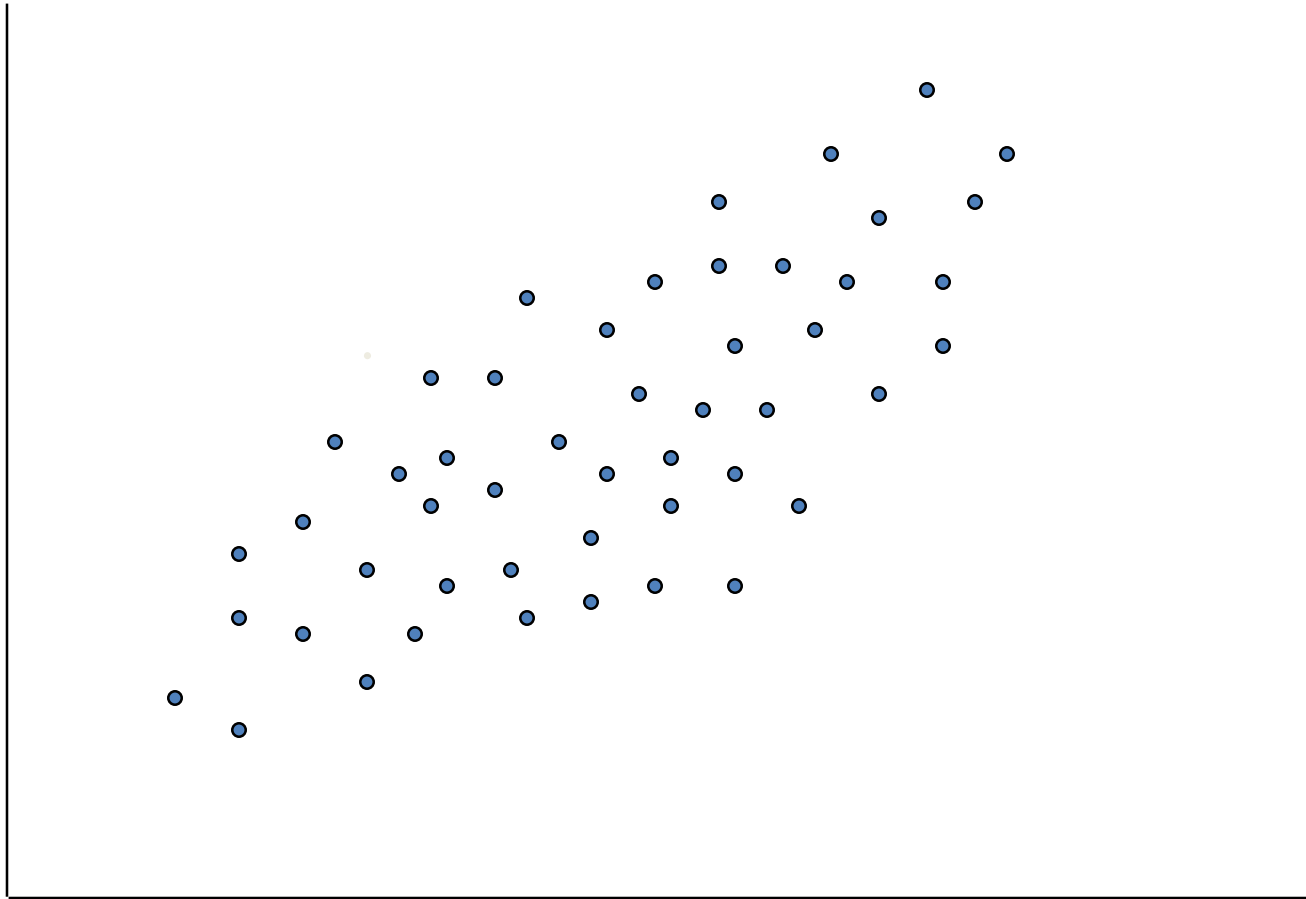
- Advantages
 - making the diagram is educational in itself
 - diagram demonstrates knowledge of problem solving team
 - diagram results in active searches for causes
 - diagram is a guide for data collection

Cause and effect diagrams

To construct the skeleton, remember:

- For manufacturing - the 4 M's
 - ✓ man, method, machine, material
- For service applications
 - ✓ equipment, policies, procedures, people

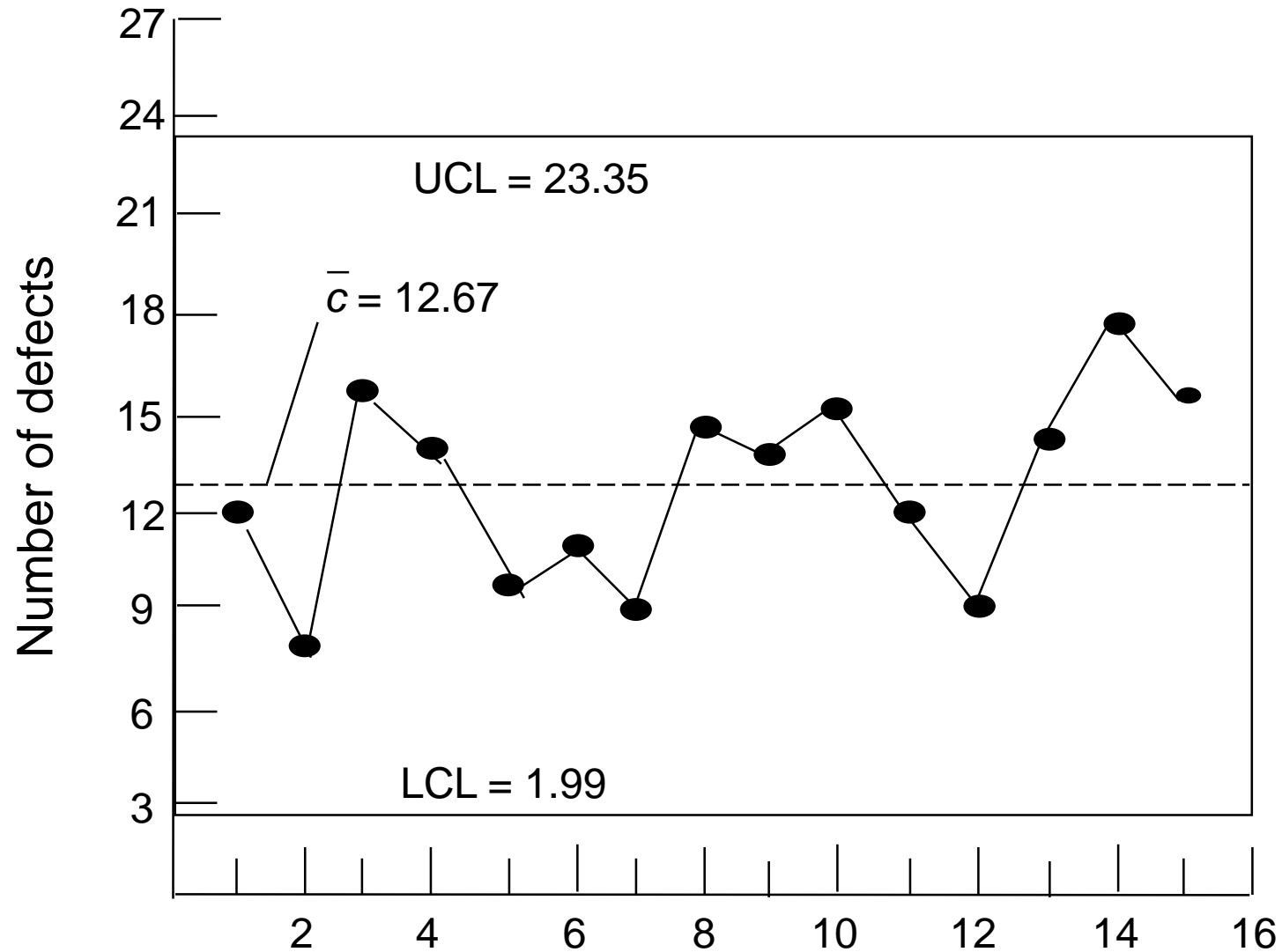
Scatter Diagram



Run Charts

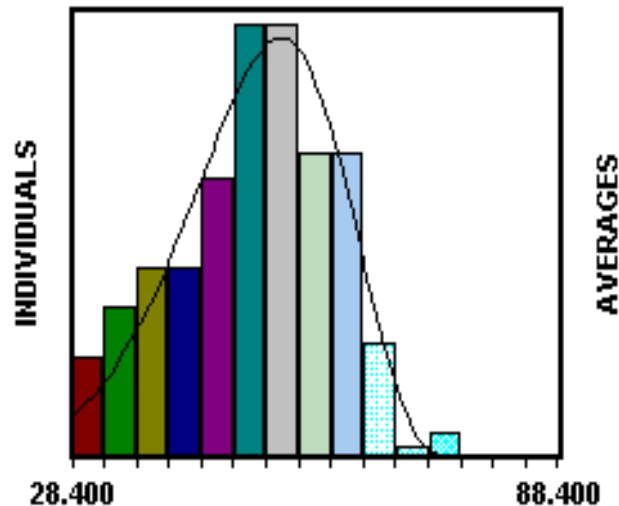
- Run Charts (time series plot)
 - Examine the behavior of a variable over time.
 - Basis for Control Charts

Control Chart

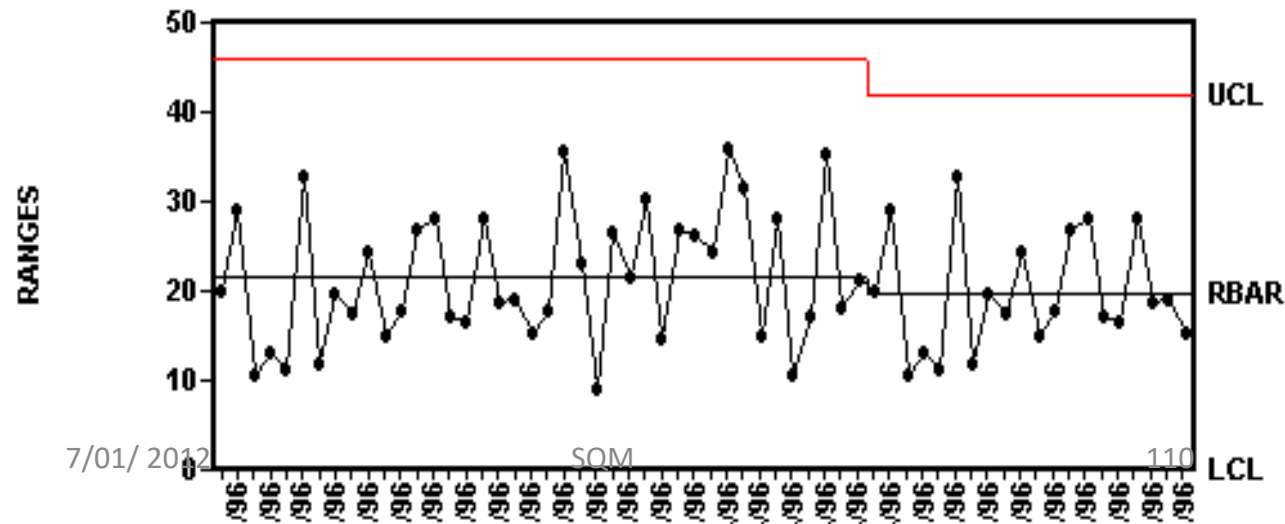
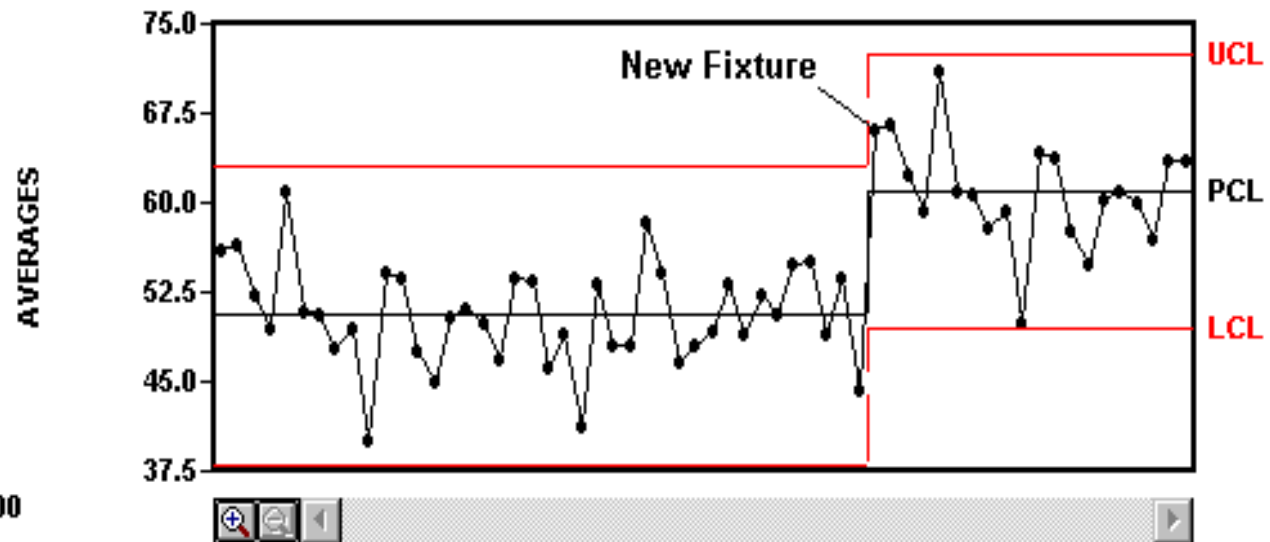


Control Charts

Order Processing Times January 1996 - October 1996



For Groups 1-40:
Auto drop : OFF
CL Ordinate: 3.000
Curve: Johnson Sb.
K-S: 0.997
AVERAGE(m) : 50.709
PROCESS SIGMA : 9.344
UCL (for group size 5) : 63.246
LCL (for group size 5) : 38.172





Introduction to Function Points

Mauricio Aguiar
International Function Point Users Group
191 Clarksville Rd.
Princeton Junction, NJ 08550

Tel: 609-799-4900
Email: ifpug@ifpug.org
Web: www.ifpug.org



Introduction to Function Points

Credits:

The International Function Point Users Group (IFPUG) would like to thank the following individuals and companies for their contributions to this presentation:

- **Mary S. Bradley - MSB2 Consulting**
- **Mick Burn-Murdoch - Software Measurement Services, Ltd.**
- **Carol Dekkers - Quality Plus Technologies, Inc.**
- **Sheila Dennis - DFAS**
- **David Garmus - David Consulting Group**
- **Scott Goldfarb - Q/P Management Group, Inc.**
- **Cindy Woodrow - GEICO**
- **Steven Woodward - Q/P Management Group of Companies**



Agenda

- Introduction
- Why use Function Points
 - Managing Your Software
 - Managing Your Organization
 - Function Points vs. Lines of Code
- How to Count Function Points
- Questions

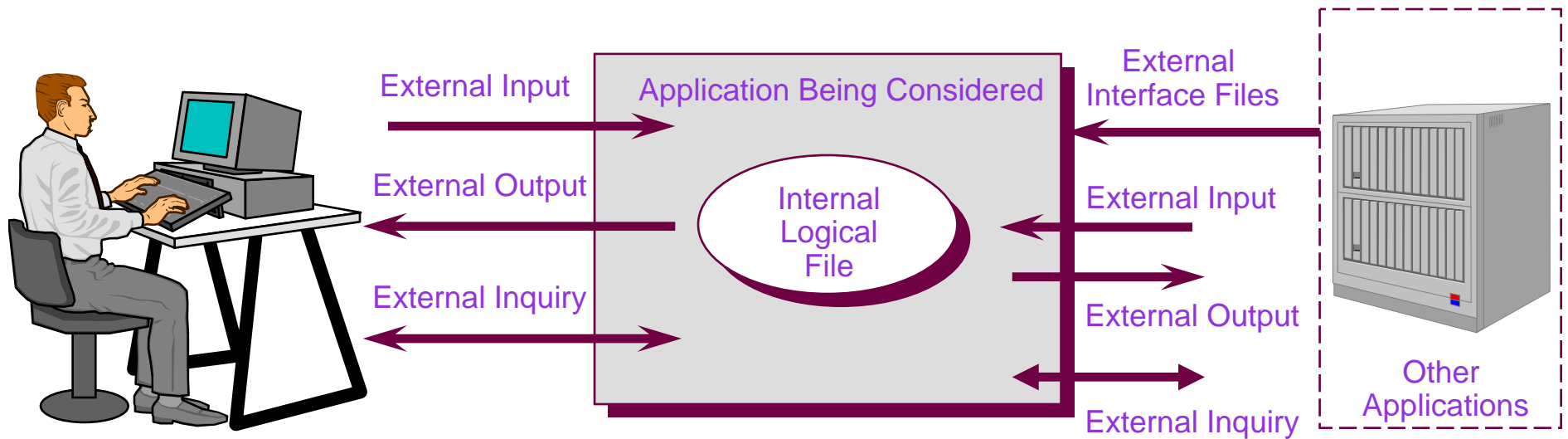


Objectives of Function Point Analysis

- Measures software by quantifying the functionality requested by and provided to the customer based primarily on logical design
- Measures software development and maintenance independently of technology used for implementation
- Measures software development and maintenance consistently across all projects and organizations



Function Points are a Unit of Measure

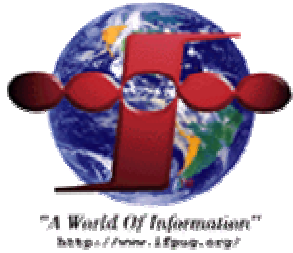


- Functionality as viewed from the user's perspective



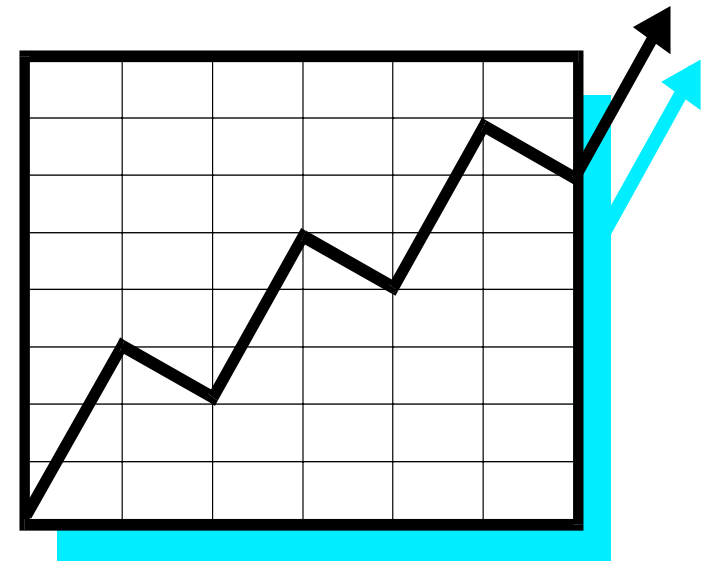
Why Use Function Points

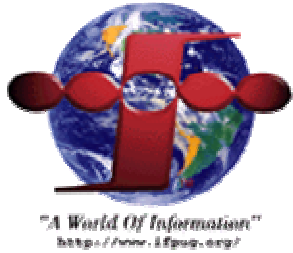
- Managing Your Software



Software Development Challenges

- Size of Requirements
- Changes to Requirements
- Estimation Based on Requirements
- Measuring and Improving Productivity and Quality

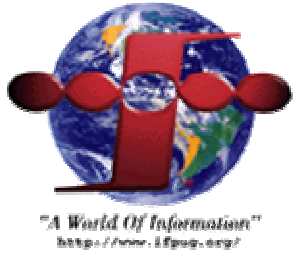




Size of Requirements

- Requirements
 - Complete
 - Business Terms
 - Mutual Understanding
 - Document Assumptions
 - Size





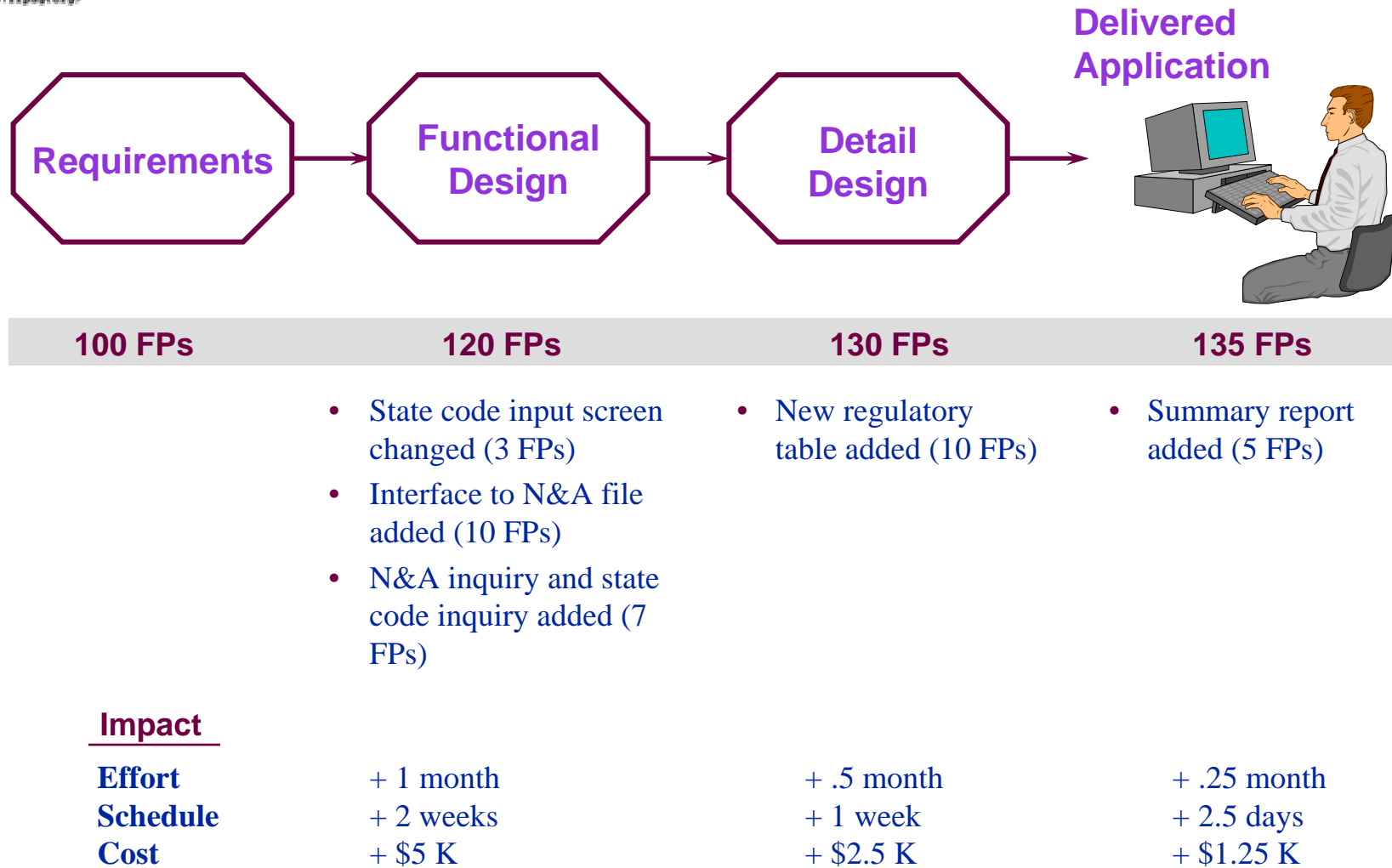
Changes to Requirements

- Changes to Requirements
 - Change Inevitable
 - Trade-offs
 - Customer Definition of Quality
 - Size





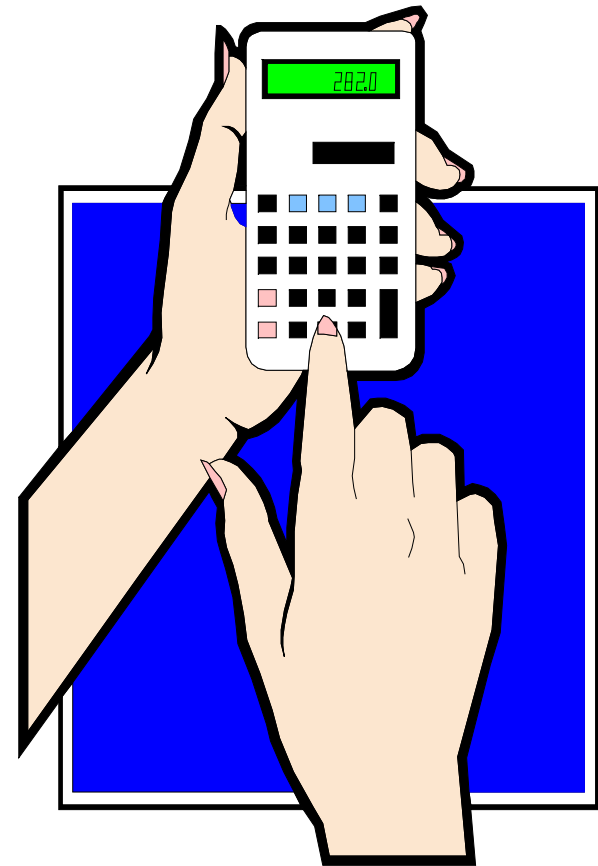
Changes to Requirements





Estimation Based on Requirements

- Estimation Based on Requirements
 - Multiple Models
 - Weighted Inputs:
 - Language
 - Skills
 - Methodology
 - Risk Factors
 - Size
 - Historical Base





Estimating Examples

Function Point Size

Project A – 100 FPs

Project B – 100 FPs

Project Variables

- On-line/database
- New development
- C++
- Highly experienced development staff

- Batch
- Enhancement
- Cobol
- Average experienced development staff

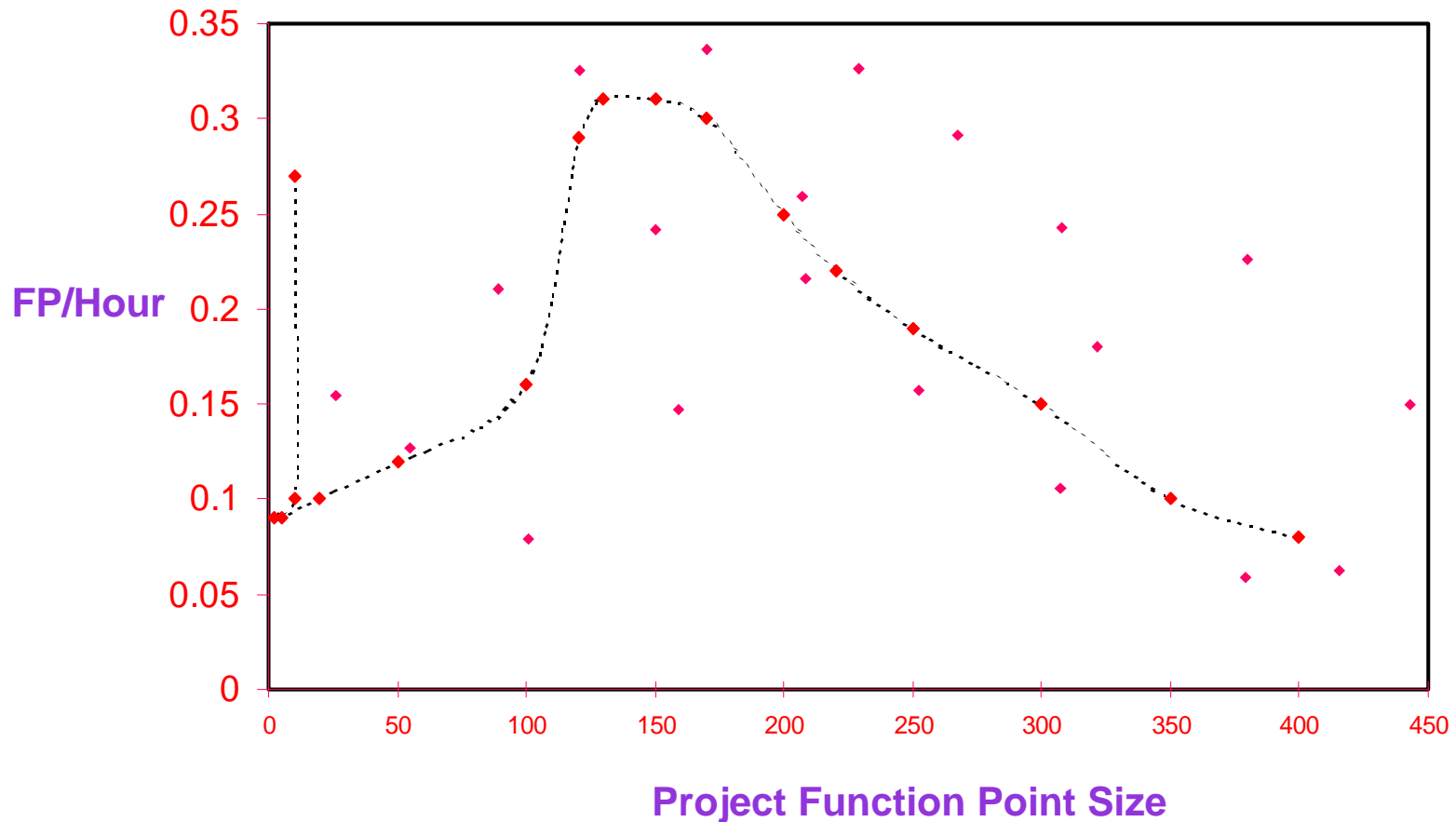
Project Estimate Based on Historical Data and/or Vendor Tool

Effort = 5 months
Schedule = 3 months
Cost (@ \$5K) = \$25,000
KLOC = 6
Delivered Defects = 25
Productivity Rate = 20 FP/Month.

Effort = 20 months
Schedule = 6 months
Cost (@ \$5K) = \$100,000
KLOC = 10
Delivered Defects = 100
Productivity Rate = 5 FP/Month



Measuring and Improving Productivity



- Every organization has an optimum size/productivity range



Why Use Function Points

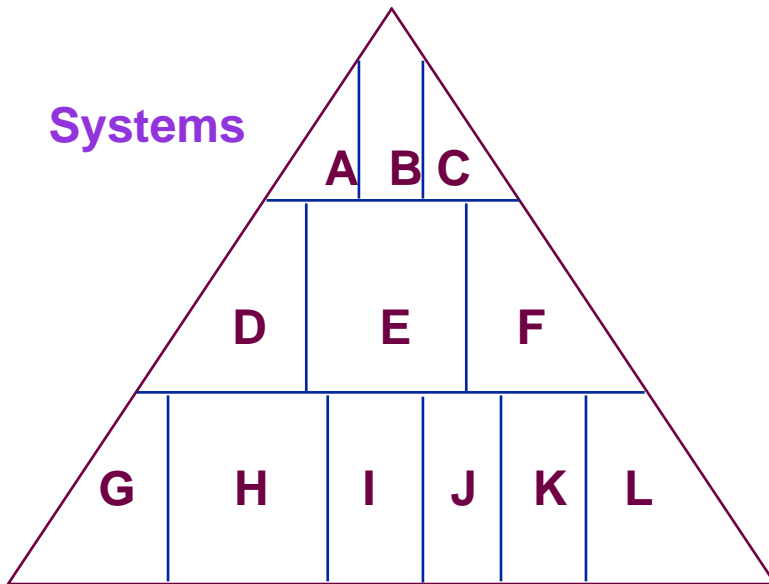
- Managing Your Organization



Asset Management

Application Portfolio

Systems



Size = 50,000 Function Points

Replacement Cost = \$300,000,000

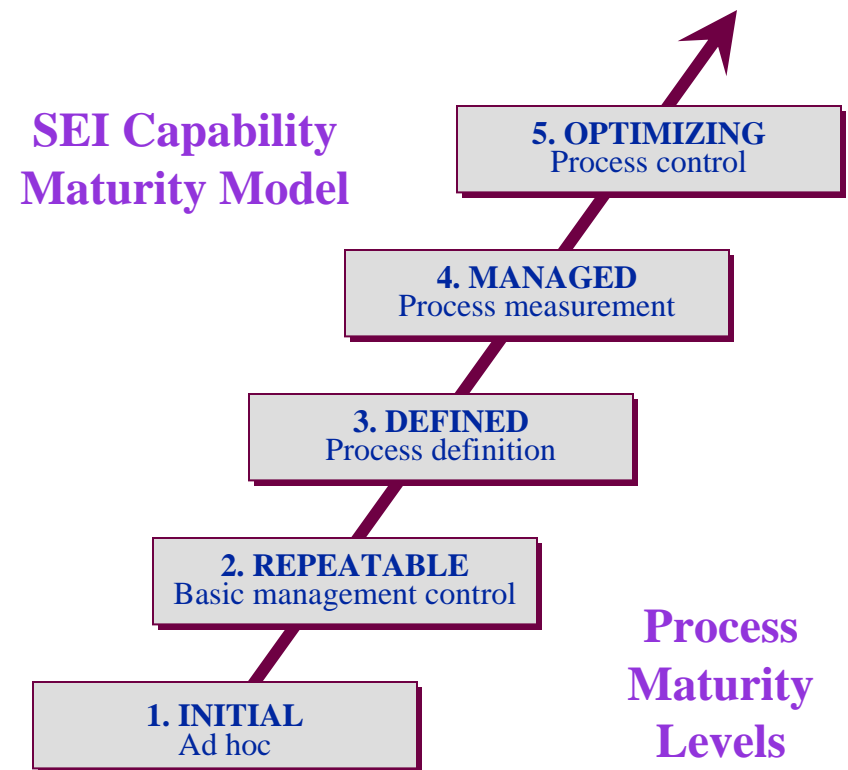
Growth = 7% per year

Support Cost = \$20,000,000 per year



Function Points and the CMM/CMMI

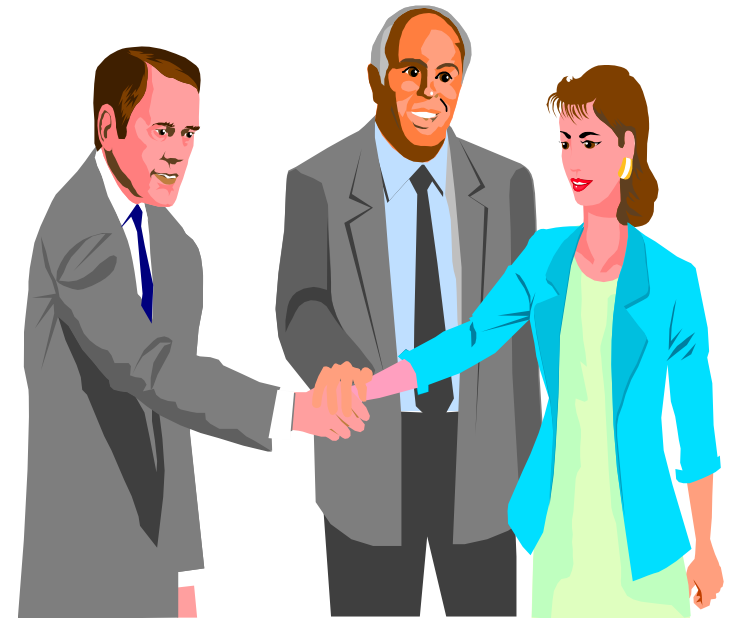
- Function Points are the metric of choice for many of the activities required in the SEI CMM and CMMI Level 2
- With CMMI, metrics becomes a Key Process Area in its own right





Improving Customer Relations

- Predictable Time scales
- Predictable Costs
- Predictable Functionality





Organizational Improvement

- Process Measurement
- Project Management Metrics
 - Estimates
 - Productivity
 - Defect Densities
 - etc.
- Benchmarking





Function Points & Metrics Help

- Evaluate current in-house and contractor performance
- Establish quantifiable expectations
- Demonstrate objectives for contract/outsourcing are met
- Establish realistic commitments
- Determine fair compensation
- Establish “win win” relationships



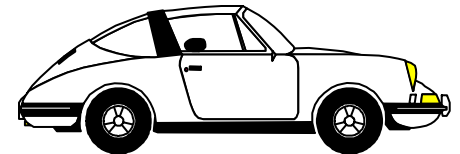
Function Points vs. Lines of Code

- Technology and platform independence
- Available from early requirements phase
- Consistent and objective unit of measure throughout the life cycle
- Objectively defines software application from the customer perspective
- Objectively defines a series of software applications from the customer's, not the technician's perspective
- Is expressed in terms that users can readily understand about their software



What is Wrong with LOC?

- There is no standard for a line of code
- Lines of Code measure components, not completed products
 - Don't measure the panels produced; measure the number of cars assembled
- Measuring lines of code
 - Rewards profligate design
 - Penalizes tight design
- Positively misleading?





Classic Productivity Paradox

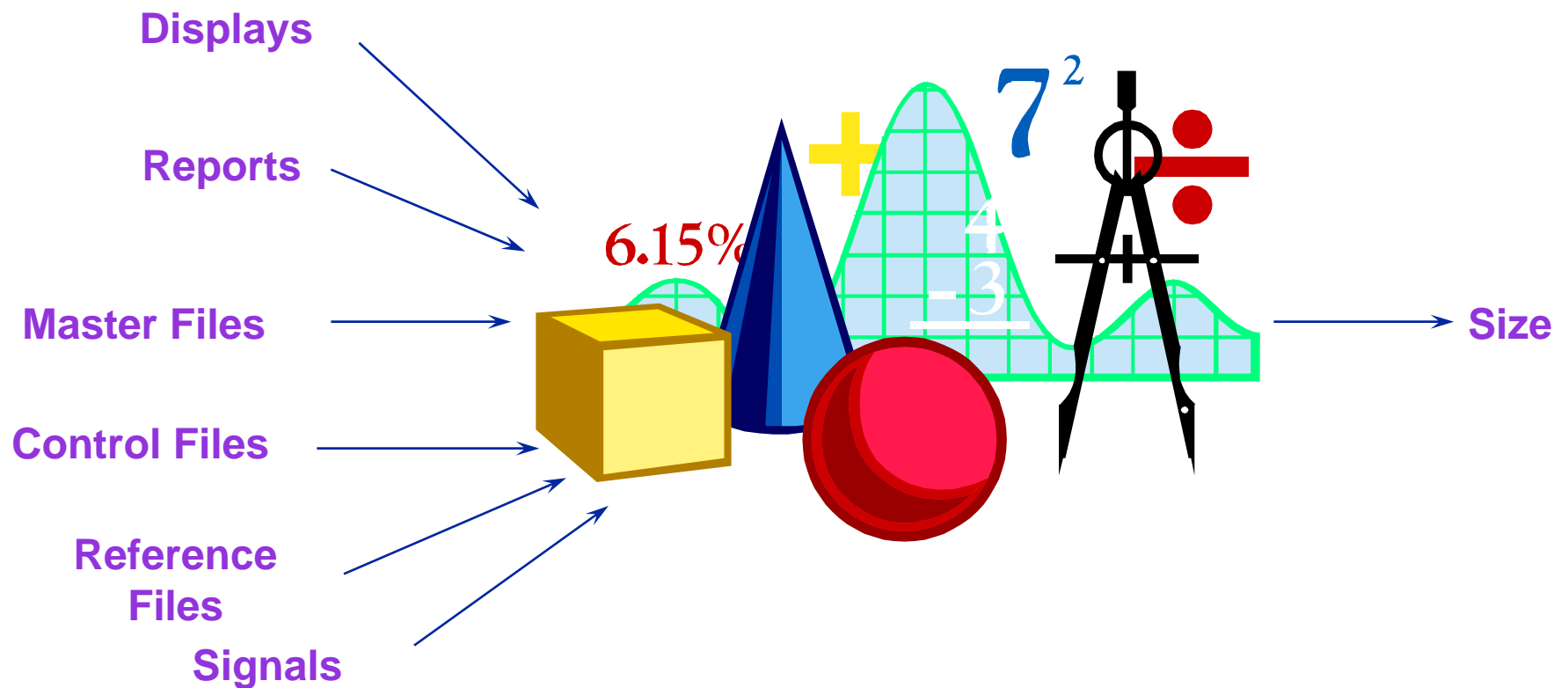
Lines of Code	10,000	3,000
Function Points	25	25
Total Months effort	25	15
Total Costs	\$125,000	\$75,000
Cost per Source Line	\$12.50	\$25.00
Lines per Person month	400	200
FPS per Person month	1.2	2
Cost per FP	\$5,000	\$3,000

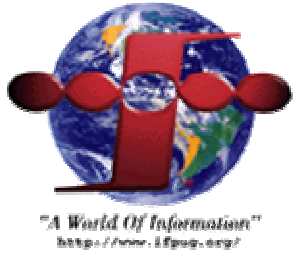


How to Count Function Points



How to Count Function Points



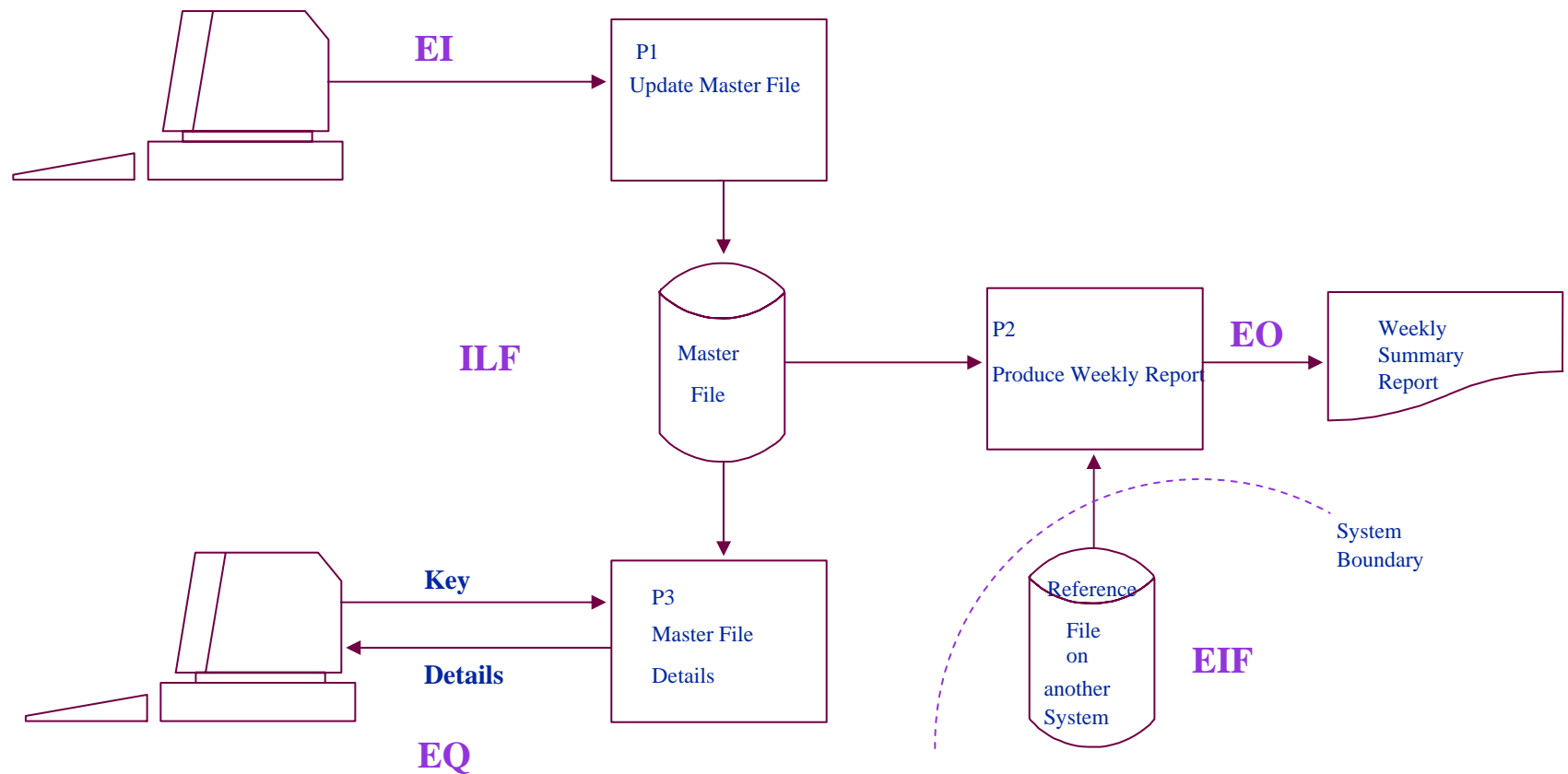


Steps in FP Counting

- Determine Type of Count (3 Types)
 - Enhancement (Project) Function Point Count
 - Application Function Point Count
 - Development (Project) Function Point Count
- Identify Counting Scope and Application Boundary
- Count Data Functions
- Count Transactional Functions
- Determine Unadjusted Function Point Count
- Determine Value Adjustment Factor
- Calculate Adjusted Function Point Count



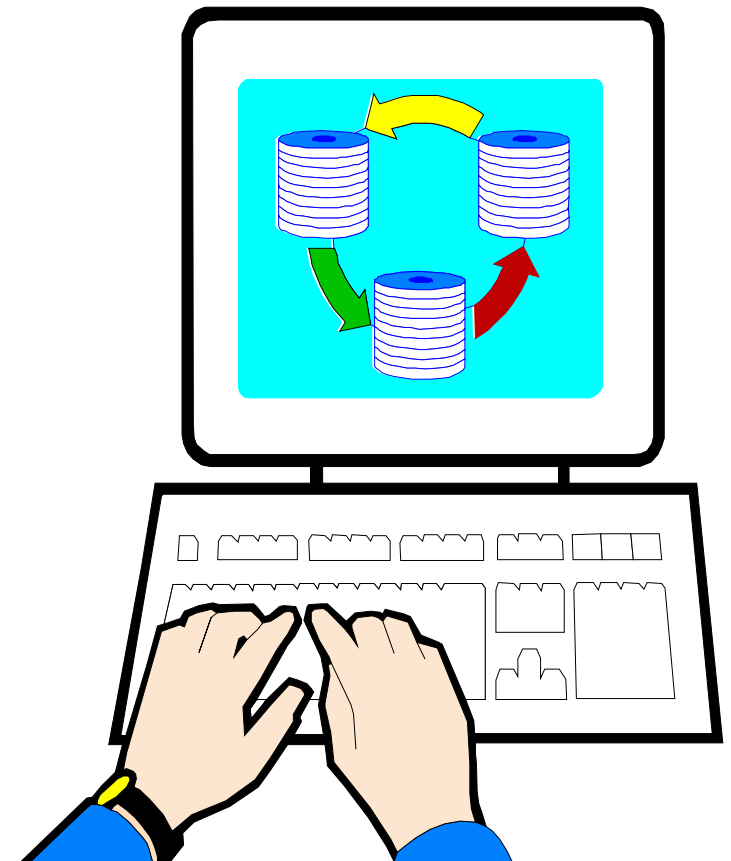
FP Overview: What Is Counted





Data Storage

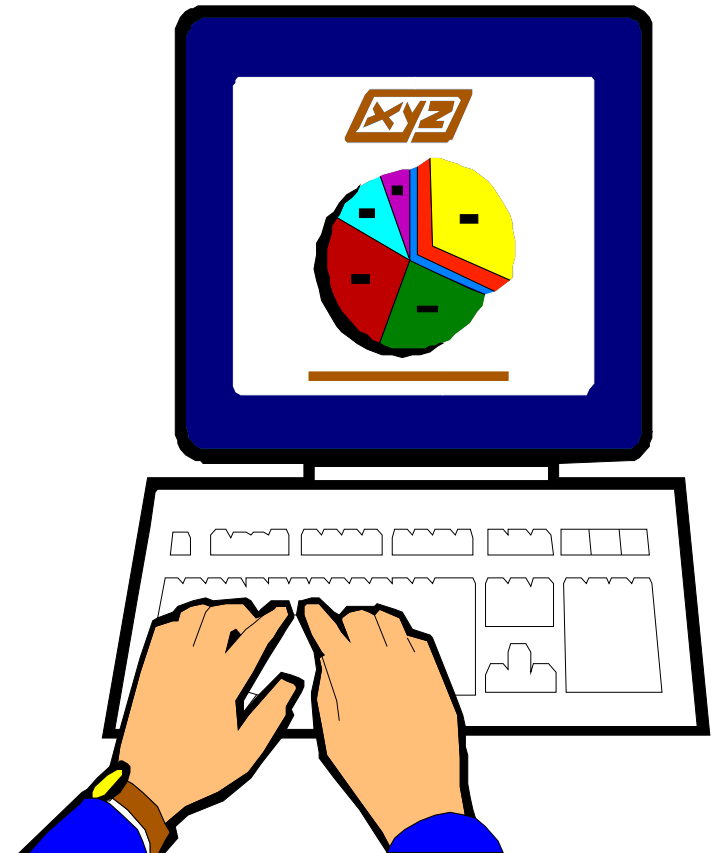
- Internal Logical File (ILF)
Logical group of data maintained by the application (e.g., Employee file)
- External Interface File (EIF)
Logical group of data referenced but not maintained (e.g., Global state table)

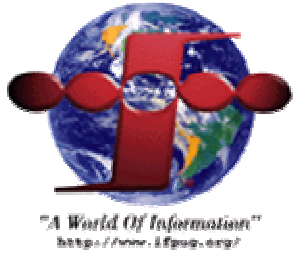




Transactions

- External Input (EI)
Maintains ILF or passes control data into the application
- External Output (EO)
Formatted data sent out of application with added value (e.g., calculated totals)
- External Query (EQ)
Formatted data sent out of application without added value





Functions are Weighted Based on Complexity

Data Element Types (DETs)

- Number of user recognizable non-repeated fields
- Applies to data and transactional functions

File Types Referenced (FTRs)

- Number of files referenced, read, created, or updated
- Applies to transactional functions

Record Element Types (RETs)

- Number of data sub-groupings or unique record formats
- Applies to data functions



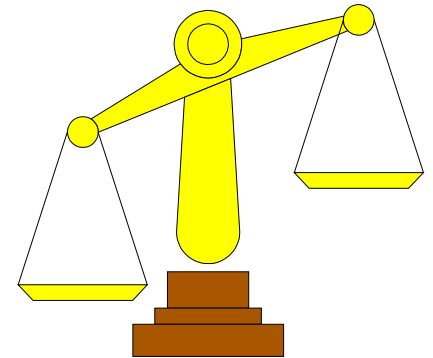
Functional Size (Unadjusted Function Size)

Function Type	Low	Average	High
EI	x 3	x 4	x 6
EO	x 4	x 5	x 7
EQ	x 3	x 4	x 6
ILF	x 7	x 10	x 15
EIF	x 5	x 7	x 10



Value Adjustment Factor

- Based on 14 General System Characteristics (User Business Constraints Independent of Technology)
 - Examples: data communications, response times, end user efficiency, multiple sites and flexibility
- Adjusts FP count by up to + / - 35%





Questions ?