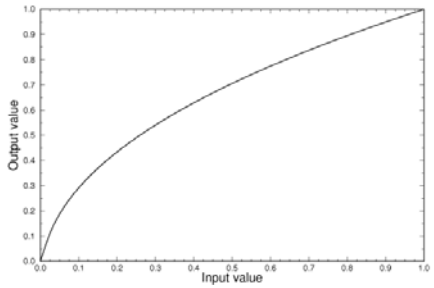
Measuring and Controlling Software Development

Approaches for Measuring and Controlling Software Development

- GAMMA
- GQM (Goal, Question, Metric)
- DBO (Design By Objective)

The GAMMA Paradigm

- Goal
- Attribute
- Measure
- Means of Measurement
- Action



The GAMMA Paradigm: Goal

- Set measurable targets for what is to be achieved.
- Obviously this applies to the overall constraints of cost and schedule.
  - E.g. Failure rate will not exceed 1 per 10,000 operating hours.

The GAMMA Paradigm: Attribute

- We must select an entity to measure, and define the attributes of each entity that is of interest from the point of view of managing the project.
  - E.g. Considering the above goal, entity would be the delivered system and the measure would be reliability

The GAMMA Paradigm: Measure

- We must define meaningful measures, which represent each of the attributes.
  - E.g. for the goal above, a measure of reliability (here the failure rate is expressed as number of failures per operating hour)

The GAMMA Paradigm: Means of Measurement

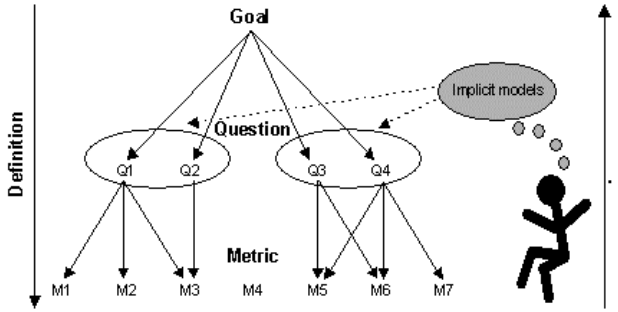
- In addition to having a defined measure, we must also have the means whereby we can actually evaluate it in any particular case.
  - E.g. Times of failures and amount of operating time will be recorded during a representative trial of a suitable length, and the failure rate calculated using a reliability growth model.

The GAMMA Paradigm: Action

- It is finally, necessary to take some action depending on the value obtained.
  - E.g. If the failure rate is too high, we could negotiate an extension to the deadline to allow further testing, or perhaps cancel the project.

Goal-Question-Metric (GQM)

- **Goal-Question-Metric** (GQM) method of experiment design is an approach in which management selects certain goals which lead to certain questions and in turn lead to certain metrics.

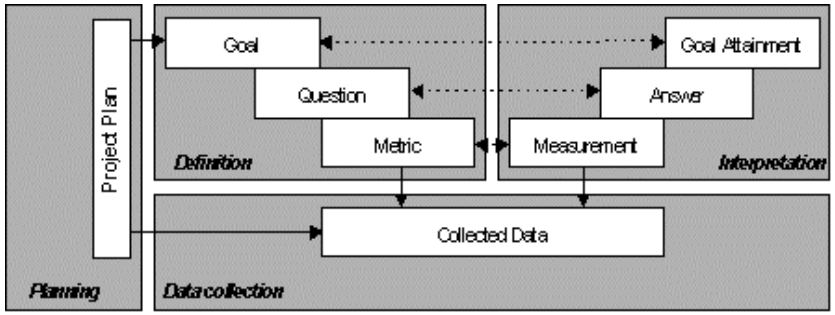


Goal-Question-Metric (GQM)

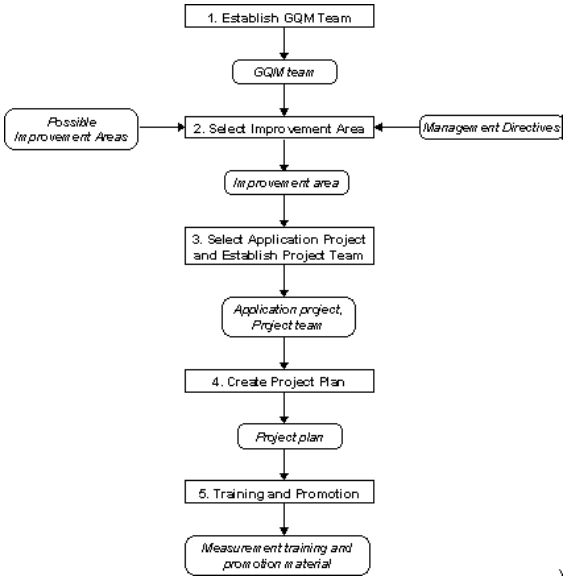
- The **Goals** define the abstract level of the experiment.
  - E.g. “Improve programmer productivity”
- The **Questions** identify the operational level (what we want to learn).
  - E.g. “How much code does a programmer produce?”
- The **Metrics** define the quantitative level (what we will measure).
  - E.g. “KLOC produced per month”

The 4 Phases of the GQM Method

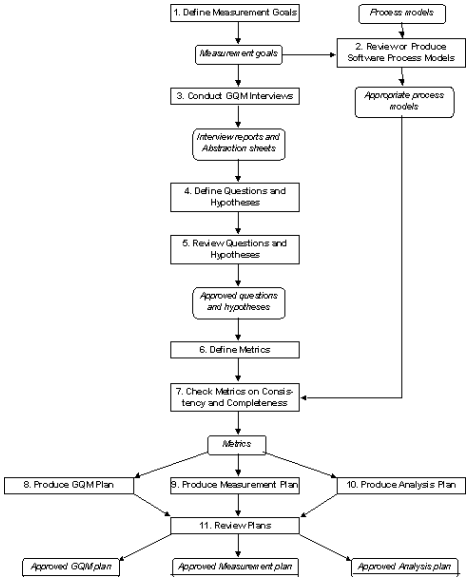
- There are 4 phases in the GQM method:
  - Planning
  - Definition
  - Data Collection
  - Interpretation



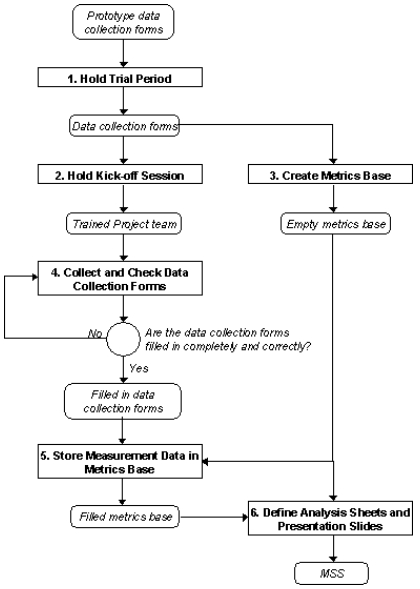
GQM Planning Procedures



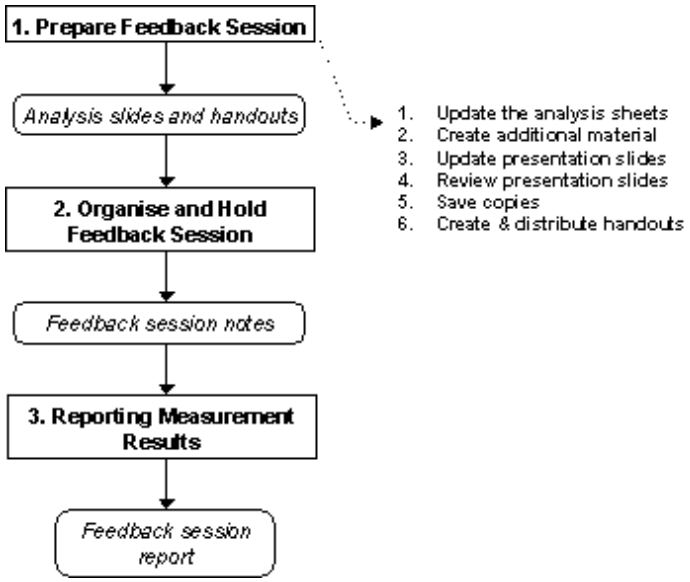
GQM Definition Procedures



Data Collection Start-up Procedures



GQM Interpretation Procedures



Design By Objective (DBO)

- In this approach customer and developer agree upon certain measurable quality objectives and by which they can be measured in order to assess objectively that the objectives have been achieved and that improvement over the earlier products has been made.

Capability Maturity Model (CMM)

- The **Capability Maturity Model (CMM)** is a method for evaluating and measuring the maturity of the software development process of organizations on a scale of 1 to 5.
- It is a formal archetype of the evolutionary stages that lead toward a desired level of competency in a software engineering.
- It has been used extensively for avionics software and for government projects since it was created in the mid-1980s.
- A maturity model is a structured collection of elements that describe characteristics of effective processes.

CMM is NOT prescriptive that it does NOT tell an organization how to improve.

What can a CMM Provide?

- A maturity model provides:
  - A place to start
  - The benefit of a community's prior experiences
  - A common language and a shared vision
  - A framework for prioritizing actions
  - A way to define what improvement means for your organization
  - A maturity model can be used as a benchmark for assessing different organizations for equivalent comparison.

5 Process Maturity Levels of CMM

- A system for organizational evaluation which is based on the idea that an organization should improve its management processes **(*Capabilities*)** as it gains experience **(*Matures*)**.
- Organizations can progress through 5 levels of maturity with prominent characteristics at each plateau.
  - Level 1: Initial
  - Level 2: Repeatable
  - Level 3: Defined
  - Level 4: Managed
  - Level 5: Optimizing

CMM Level 1: Initial

- The software process is characterized as ad hoc and occasionally even chaotic.
- Few processes are defined, and success depends on individual effort.

CMM Level 2: Repeatable

- Basic project management processes are established to track cost, schedule, and functionality.
- The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

CMM Level 3: Defined

- The software processes for both management and engineering activities is documented, standardized, and integrated into an organization wide software process.
- All projects use a documented and approved version of the organization's process for developing and supporting software.
- This level includes all characteristics defined for level 2.

CMM Level 4: Managed

- Detailed measures of the software process and product quality are collected.
- Both the software process and products are quantitatively understood and controlled using detailed measures.
- This level includes all characteristics defined for level 3.

CMM Level 5: Optimizing

- Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies.
- This level includes all characteristics defined for level 4.

Problems with Conventional Specification

- Contradictions
- Ambiguities
- Vagueness
- Imcompleteness
- Mixed Levels of Abstraction

Formal Methods Model

- **Formal Methods Model** encompasses a set of activities that leads to formal mathematical specification of computer software.
- Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.

Formal Methods Concepts

- **Data Invariant** – a condition that is true throughout the execution of the system that contains a collection of data
- **State** – The stored data which a system accesses and alters
- **Operation** – An action that takes place in a system and reads or writes data to a state
  - **Precondition** defines the circumstances in which a particular operation is valid
  - **Post-condition** defines what happens when an operation has completed its action

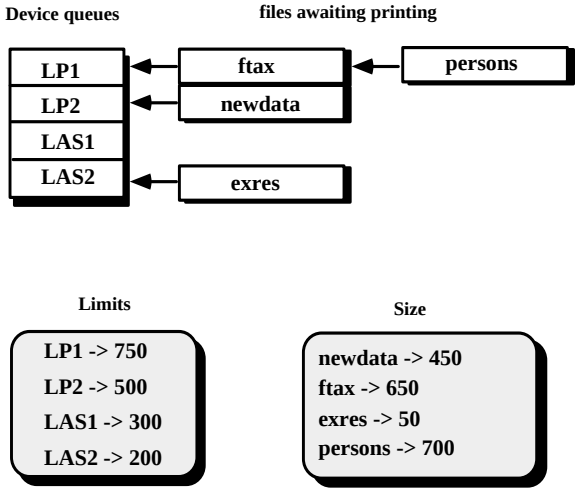
Benefits of Formal Methods Model

- When used during development
  - They provide a mechanism for eliminating many of the problems that are difficult to overcome.
  - Ambiguity, incompleteness, and inconsistency can be discovered and corrected and corrected more easily.
- When used during design
  - They serve as a basis for program verification and therefore enable the software engineer to discover and correct errors that might go undetected.

Limitations of Formal Methods Model

- Time consuming and expensive.
- Few software developers have the necessary background, extensive training is required.
- Difficult to use the models as a communication mechanism for technically unsophisticated customers.

An Example: Print Spooler



States and Data Invariant

- The state of the spooler is represented by the 4 components: Queues, Output Devices, Limits, and Sizes.
- The data invariant has five components:
  - Each output device is associated with an upper limit of print lines
  - Each output device is associated with a possibly nonempty queue of files awaiting printing
  - Each file is associated with a size
  - Each queue associated with an output device contains files that have a size less than the upper limit of the output device
  - There will be no more than MaxDevs output devices administered by the spooler

Operations

- An operation which adds a new output device to the spooler together with its associated print limit
- An operation which removes a file from the queue associated with a particular output device
- An operation which adds a file to the queue associated with a particular output device
- An operation which alters the upper limit of print lines for a particular output device
- An operation which moves a file from a queue associated with an output device to another queue associated with a second output device

Pre- & Postconditions

- For the first operation (adds a new output device to the spooler together with its associated print limit):
- Precondition: the output device name does not already exist and that there are currently less than MaxDevs output devices known to the spooler
- Post-condition: the name of the new device is added to the collection of existing device names, a new entry is formed for the device with no files being associated with its queue, and the device is associated with its print limit.

Component-based Systems Engineering (CBSE)

- **Component-based Systems Engineering** (CBSE) seems quite similar to conventional or object-oriented software engineering.
- The process begins when a software team establishes requirements for the system to be built using conventional requirements elicitation techniques.
- An architectural design is established, but rather than moving immediately into more detailed design tasks, the team examines requirements to determine what subset is directly amenable to composition, rather than construction.

The Key Questions

- When faced with the possibility of reuse, the software team asks:
  - Are commercial off-the-shelf (COTS) components available to implement the requirement?
  - Are internally-developed reusable components available to implement the requirement?
  - Are the interfaces for available components compatible within the architecture of the system to be built?
- At the same time, they are faced with the following impediments to reuse ...

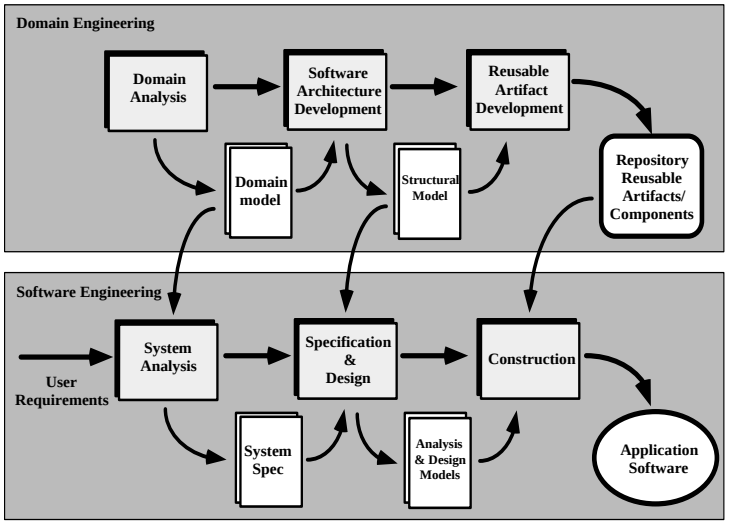
Impediments to Reuse

- Few companies and organizations have anything that even slightly resembles a comprehensive software reusability plan.
- Although an increasing number of software vendors currently sell tools or components that provide direct assistance for software reuse, the majority of software developers do not use them.
- Relatively little training is available to help software engineers and managers understand and apply reuse.
- Many software practitioners continue to believe that reuse is “more trouble than it’s worth.”
- Many companies continue to encourage of software development methodologies which do not facilitate reuse
- Few companies provide an incentives to produce reusable program components.

Component-based Systems Engineering (CBSE)

- The team attempts to modify or remove those system requirements that cannot be implemented with COTS or in-house components.
- If the requirement cannot be changed or deleted, those new components will be developed.

The CBSE Process



Domain Engineering

- Define the domain to be investigated.
- Categorize the items extracted from the domain.
- Collect a representative sample of applications in the domain.
- Analyze each application in the sample.
- Develop an analysis model for the objects.

Identifying Reusable Components

- Is component functionality required on future implementations?
- How common is the component's function within the domain?
- Is there duplication of the component's function within the domain?
- Is the component hardware-dependent?
- Does the hardware remain unchanged between implementations?
- Can the hardware specifics be removed to another component?
- Is the design optimized enough for the next implementation?
- Can we parameterize a non-reusable component so that it becomes reusable?
- Is the component reusable in many implementations with only minor changes?
- Is reuse through modification feasible?
- Can a non-reusable component be decomposed to yield reusable components?
- How valid is component decomposition for reuse?

Structural Modeling

- Every application has structural patterns that have the potential for reuse
- A **Structure Point** is a construct with the structure
  - A structure point is an abstraction that should have a limited number of instances. Restating this in object-oriented jargon , the size of the class hierarchy should be small.
  - The rules that govern the use of the structure point should be easily understood. In addition, the interface to the structure point should be relatively simple.
  - The structure point should implement information hiding by hiding all complexity contained within the structure point itself. This reduces the perceived complexity of the overall system.

Software Components of CBSE

- **Qualified Components** – Assessed by software engineers to ensure that not only functionality, but performance, reliability, usability, and other quality factors conform to the requirements of the system to be built.
- **Adapted Components** – Adapted to modify unwanted or undesirable characteristics.
- **Assembled Components** – Integrated into an architectural style and interconnected with an appropriate infrastructure that allows the components to be coordinated and managed effectively.
- **Updated Components** – Replacing existing software as new versions of components become available.

CBSE Activities

- For those requirements that are addressed with available components, the following activities commences:
  - **Component Qualification** – To qualify the fitness of component.
  - **Component Adaptation** – In order to meet the architecture's design rules.
  - **Component Composition** – Identifying connection and coordination mechanisms.
  - **Component Update** – Update is complicated by the imposition of a third party who implement the component.

Consideration in Qualification

- Before a component can be used, you must consider:
  - Application Programming Interface (API)
  - Development and integration tools required by the component
  - Run-time requirements including resource usage (e.g., memory or storage), timing or speed, and network protocol
  - Service requirements including operating system interfaces and support from other components
  - Security features including access controls and authentication protocol
  - Embedded design assumptions including the use of specific numerical or non-numerical algorithms
  - Exception handling

Adaptation

- The implication of **Easy Integration** is:
  - That consistent methods of resource management have been implemented for all components in the library;
  - That common activities such as data management exist for all components, and
  - That interfaces within the architecture and with the external environment have been implemented in a consistent manner.

Composition

- An infrastructure must be established to bind components together
- Architectural ingredients for composition include:
  - Data exchange model
  - Automation
  - Structured storage
  - Underlying object model

Classification

- **Enumerated Classification** – Components are described by defining a hierarchical structure in which classes and varying levels of subclasses of software components are defined
- **Faceted Classification** – A domain area is analyzed and a set of basic descriptive features are identified
- **Attribute-value Classification** – A set of attributes are defined for all components in a domain area

Impact on Quality for CBSE

- In an ideal setting, a software component that is developed for reuse would be verified to be correct and contain no defects.
- In reality, formal verification is not carried out routinely, and defects can and do occur.
- However, with each reuse, defects are found and eliminated, and a component's quality improves as a result.

Impact on Productivity for CSBE

- Less time is spent creating the plans, models, documents, code, and data that are required to create a deliverable system. It follows that the same level of functionality is delivered to the customer with less input effort.
- It appears that 30-50% reuse can result in productivity improvements in 25-40%.

Impact on Cost for CBSE

- The net cost saving for reuse are estimated by projecting the cost of the project if it were developed from scratch and then subtracting the sum of the costs associated with reuse and the actual cost of the software as delivered.
- Reuse costs include
  - Domain analysis & modeling
  - Domain architecture development
  - Increased documentation to facilitate reuse
  - Support and enhancement for reuse components
  - Licenses for externally acquired components
  - Creation or acquisition and operation of a reuse repository
  - Training of personnel in design and construction for reuse

Component-Based Development

- A library of components must be available
- Components should have a consistent structure
- A standard should exist, e.g.,
  - OMG/CORBA
  - MS COM
  - JavaBeans