1. State the most important topic covered by DeMarco and Lister in “Peopleware” and discuss it in detail. You must support your assertion it was the most important topic! Your question must not relate to another question on the test.
   1. **What does affect quality and productivity? The hope is that the answer to this question will allow individuals and companies to improve their performance.**

**Traditional Ideas:**

1. **Choice of programming language.**
2. **Experience is an asset.**
3. **Increasing development time helps reduce the number of defects in the final product.**
4. **Salary**

**Choice of programming language.**

**Programming language is the traditional choice of the propeller beanie heads in the organization. Reality: language has very little effect on either productivity or quality. Those who used older languages in the study such as Fortran or COBOL did essentially as well as those who used new languages such as Pascal or “C.”**

**The productivity and quality spread within each language group was much like the overall spread of performance.**

**The only exception was assembly language. Individuals using assembly language were badly left behind all other groups (but they are used to this phenomenon).**

**Higher reported productivity rates in languages such as C++, Ada, and other new languages is apparently not a result of the language but rather the use of a development paradigm such as Jackson’s Technique or Object Oriented Design and Programming. The actual language used to implement the paradigm is not important.**

**Years of Experience.**

**It is widely believed that experience is an important determinant of productivity and quality.**

**People who had ten years of experience in the study did not out perform those with two years of experience.**

**There was no correlation between experience and performance except those with less than 6 months experience with the language used in the exercise did not perform as well as the rest of the sample.**

**A person right out of college is likely to perform better than a person with 10 or more years of experience.**

**Individuals do not perform to their maximum capacity until they have learned and adjusted too the corporate culture!**

**These results compare favorably to other studies.**

**More development time helps reduce the number of defects in the final product.**

**It is widely believed that if you wish to improve quality by reducing the error rate, you must accept longer delivery times. That is there is a time versus quality trade off.**

**Nearly a third of the participants in the war games completed the exercise with zero defects. As a group, the zero defect workers paid no performance penalty for doing more precise work. In fact, they took slightly less time on the average, to complete the exercise than those who had one or more defects!**

**Salary**

**The most productive workers are remunerated accordingly.**

**Salary levels varied widely over the sample. There was a weak relationship between salary and performance.**

**The half above the median made less than 10% more than the half below the median.**

**Mediocrity pays!**

**Remember, the half above the median is approximately twice as productive and a third more likely to produce zero defect work.**

**The performance spread at any given salary level was nearly as wide as the whole sample.**

**Stated another way: In most organizations, people are not paid according to the quantity and quality of work they do. In most organizations, mediocrity pays! Apparently management does not know how to separate the wheat from the chaff.**

**What does affect productivity and quality?**

**Environment of the Best and the Worst Performers in the Coding War Games**

|  |  |  |
| --- | --- | --- |
| **Environmental Factor** | **1st Quartile** | **4th Quartile** |
| **How much dedicated work space do you have?** | **78 sq. ft.** | **46 sq. ft.** |
| **Is it acceptably quiet?** | **57% yes** | **29% yes** |
| **Is it acceptably private?** | **62%** | **19%** |
| **Can you silence your phone?** | **52%** | **10%** |
| **Can you divert your calls when needed?** | **76%** | **19%** |
| **Do people interrupt you needlessly and often?** | **38%** | **76%** |

**This table was based on participant surveys.**

\*\* The best performers tend to gravitate to organizations that provide a better working environment.

**\*\* Remember those employees above the median are about twice as productive as those below the median. They are one third more likely to produce zero defect work and do it in less time.**

**Cutting cost on office equipment such as desks, chairs, computers, large monitors, software products, documentation, and office space has a high hidden cost!**

“I get my best work done in the early morning before everybody else arrives!”

**“In one evening, I can accomplish two or three days worth of work!”**

**“The office is a zoo all day, but after 5:30 P.M., things settle down and you can really accomplish something.**

**Do you or your employees are making these statements, arrive early or stay late to accomplish their jobs?**

**Are employees hiding in conference rooms, storerooms or elsewhere?**

**Do employees feign sickness just prior to crucial deadlines to stay home and catch up?**

**If so it is quite possibly an indictment against your leadership and management abilities!**

**It is not unusual for these conditions to exist. What is unusual and hard to believe is that everyone knows you cannot accomplish anything in the office no effort is made to correct the problem!**

**Even if you can prove that two workers can normally function adequately in 100 square feet, the noise level will be substantially higher. You risk losing the more valuable creative leap when it is needed!**

1. The most important technique covered in class to date with respect to producing high quality software this semester is called a “Structured Walk Through” or “Technical Review.” First explain why this is true. Second discuss how the walk through is accomplished and when in detail.
   1. ***Walk-throughs are most productive when:***
   2. **The material to be discussed is distributed to the attendees for review two or three days before the walk through. *Management must support the position that all attendees do their home work prior to the walk-through to avoid wasting the groups time.***
   3. ***The number of attendees is limited to four or five and the session to about one hour.* The project manager may be barred from the walk-through if they are not technically competent or it is felt their presence may inhibit the frank and open exchange of ideas. Education of junior members is a goal of the review process.**
   4. **In some cases, it may be necessary to include an impartial mediator to resolve disputes. Raise issues, don't resolve them. Watch your language, be humble, avoid critical negative phrasing. Stick to the issues. There is always some danger an overly aggressive group will be detrimental to a designer or programmer who is not very sure of their work. On the other hand, an overly aggressive individual may overwhelm the suggestions or criticisms of the group.**
   5. **Technical reviews should specifically "avoid" redesigning systems or programs. The purpose of the meeting is to verify the specification/design/code accomplishes the desired purpose and identify problems. Spontaneous group designs almost always end in failure.**
   6. **Formal notes should be taken and the individual or group that is the subject of the inspection should be expected to respond to the groups suggestions in a reasonable time.**
   7. ***The entire "group" is responsible for any flaws or bugs in the final product.***
   8. ***To illustrate, assume***
   9. **the cost of a error uncovered during *design cost 1.0* monetary units to correct.**
   10. **The same error uncovered just *before testing commences will cost 6.5 units* (relatively speaking).**
   11. ***During testing, 15 units*.**
   12. **If not found until *after release, 60 to 100 units*.**
   13. ***Then the development cost comparison:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Errors Found** | **Number** | **Cost Unit** | **Total** |
|  | **Reviews** | **Conducted** |  |
| **During design** | **22** | **1.5** | **33** |
| **Before test** | **36** | **6.5** | **234** |
| **During test** | **15** | **15** | **315** |
| **After release** | **3** | **67** | **201** |
|  |  |  | ***783*** |
|  | **No Review** | **Conducted** |  |
| **Before test** | **22** | **6.5** | **143** |
| **During test** | **82** | **15** | **1230** |
| **After release** | **12** | **67** | **804** |
|  |  |  | ***2177*** |

1. Discuss “white box” testing in detail including McCabe’s Measure.
   1. **White/Glass Box Test:**
   2. **Knowing the internal workings of a product, tests can be conducted to be sure that all paths are exercised. Exhaustive testing is seldom feasible (compiler – infinite number of legal/illegal; license plates – large upper bound).**
   3. **White box testing typically discovers from *50% to 70%* of software errors.**
   4. **Most popular form of traditional testing.**
   5. **White Box Testing**
   6. **Ideally all datum should be tested exhaustively but this is seldom feasible. For example, assume a license plate consisting of letters and digits in the form LLL-DDD. The number of combinations of data is 26\*26\*26\*10\*10\*10 = 17,576,000 possibilities.**
   7. **A language translator must accept an infinite number of correct programs and reject an infinite number bad programs.**
   8. **We compromise by:**
      1. **guarantee that all independent paths within a module have been exercised at least once**
      2. **exercise all logical decisions on their TRUE and FALSE sides**
      3. **execute all loops at their boundaries and within their operational bounds**
      4. **exercise all internal data structures to assure their validity**
   9. **Logic errors and incorrect assumptions are inversely proportional to the probability that a program path will be executed.**
   10. **We often believe that a logical path is not likely to be executed when, in fact, it may be executed on a regular basis.**
   11. **Typographical errors are random.**
   12. **Errors are seldom uniformly distributed.**
   13. **"Bugs lurk in corners and congregate at boundaries." White box testing is far more likely to uncover them.**
2. Discuss the relationship between schedule, functionality and quality using Fredrick Brooks and Larry Putman’s material as a guide. You must evaluate each component in detail to receive credit. Do not select this option unless you can provide more than a page of pertinent information.
   1. **Development Trade Offs**

Schedule

**(delivery date)**

Functionality Quality

**(feature richness) (absence of defects)**

**To minimize one goal you must sacrifice one or both of the others!**

* 1. **Larry Putman in “Measures of Excellence” demonstrated the tradeoffs between schedule, functionality, and quality is not linearly related. If you double the number of required features and hold quality (including cost and manpower) constant then the schedule is likely to considerably more than double.**
  2. **Quality is not free. A policy of “quality as time permits” almost assures that no quality at all will sneak into the product!**

**Affect of Deadlines!**

**Assume that we are not going to be able to meet a hard deadline under current circumstances. What can be done to improve our situation?**

1. **Increase staff and or/other resources. This is normally not an option.**

Even if it is an option, large short-term increases in staff are seldom productive. They actually slow development. New people are not immediately productive as they must determine what has been done and what needs to be done. Unfortunately this requires the personnel on the project to stop working directly on the project to help. The result is at least a short-term drop in productivity.

1. **Reduce the software functionality.**
2. **Reduce Quality!**

**Quality reduction resulting from efforts to save time is not always immediately obvious.**

**“Just Good – Enough Quality”**

**When queried about MS-Word, a spokesman for Microsoft responded as follows:**

**Version 1 was approximately 27,000 lines.**

**Version 2 was approximately 2,000,000 lines.**

**When asked:**

* 1. **How many people where on the project?**
  2. **How many lines of code were generated per person-day?**
  3. **How much of the code written was deliverable and how much was discarded?**
  4. **How efficient was the project?**
  5. **How many bugs were shipped and how could it have been done better?**

**MS Executive Response: Who cares if we used 100 or 200 programmers? What difference does it make if the project was efficient?**

### We sold 17 million copies!

**If you sell enough copies, efficiency is not an issue!**

***“Good Enough” is the Enemy of Quality!***

**Traditional Western Business Attitude**

**Managers must balance the desire of the builder to produce a high quality product free from errors with the realities of the market place.**

**These realities include the level of quality with which the customer is willing to live in conjunction with the price they are willing to pay!**

**We must also consider the influence of the competition when establishing software delivery deadlines.**

**It does no good to deliver a perfect product six months behind an acceptable product from the competition. *The consumer will not wait, they will not change products after the startup cost incurred to use the competitor’s product, and they are usually not willing to pay extra for a perfect product*.**

1. Discuss the results of 1985 Survey by Lawrence and Jeffrey and the Metric Premise by Weinberg-Schulmam (“Human Factors,” 1974, Vol. 16, pp 70-77). You must convenience me you read the material and mastered the conclusions. Do not select this option unless you can provide more than a page of pertinent information.

**The goal was to determine the effect on productivity by various influences on estimating. These include:**

1. **Managers are the best estimators for projects. Developers should not be given more time than is actually required or the additional time will be squandered. In fact many believe the deadline should be a little shorter than what is actually required to encourage high productivity and some voluntary overtime to meet the attainable goal.**
2. **Development staff is the best estimators of time to complete a project. They know what they can accomplish better than anyone else. Hence, deadlines for project deliverables should be established by development staff (an almost folkloric belief in our industry).**
3. **Deadlines should be set by a systems analyst who will have absolutely no involvement in the project other than to estimate the most realistic delivery schedule.**

**All measurements of productivity were made using Barry Boehm’s CoCoMo methodology. CoCoMo is a unit less number. The higher the CoCoMo number the higher the productivity.**

**Actual industrial projects were used to collect the data.**

Results of 1985 Survey by Lawrence and Jeffrey

|  |  |  |
| --- | --- | --- |
| **Effort Estimate Prepared by:** | **Average Measured Productivity** | **Number of Projects** |
| Developer Alone | **8.0** | **19** |
| **Supervisor alone** | **6.6** | **23** |
| **Developer and Supervisor Compromise** | **7.8** | **16** |
| **Systems Analyst (not involved with project).** | **9.5** | **21** |
| **Control Group**  **(No deadline, wake me when through)** | **12.0** | **24** |

**Projects where time pressure was not used as an inducement to increase productivity experienced the highest productivity.**

**The best management strategy might be “Just wake me up when you are done.”**

**The best estimate of actual performance was by a systems analyst that did not participate in the project except to project the most likely time to completion with a fixed set of resources.**

**Developers tend to under estimate the time they require to complete a project. Apparently their egos get involved.**

Managers tend to be more interested in what is desired, or must occur, as opposed to what is possible with a fixed set of resources.

**The proper function of management is not to make people work, but to make it possible for them to work!**

1. Discuss “Flow” in detail.

**Flow**

Psychologists define flow as a deep, nearly meditative involvement with work attained during single-minded work with no interruptions. This is an individual’s most creative period.

Flow takes approximately 15 minutes to attain. A five minute phone call cost 20 minutes, 5 for the call and 15 to re-attain flow. A dozen phone calls results in loss of a half days work.

## Most Creative Period

**Recommendation:**

1. **Allow phone to be silenced and re-routed.**
2. **Do not use four-foot partitions to form work spaces. If partitions must be used, they should reach all the way to the ceiling.**
3. **Do not locate information workers in high traffic areas such as around water coolers or break rooms. Avoid “broadcast” intercoms and phones that everyone hears ringing.**

### Environmental Factor

**Uninterrupted hours**

**E-Factor = ----------------------------------**

**Body present hours**

**Work should not be measured in *“body present hours,”* rather in uninterrupted hours. Remember the cost of flow. A phone call every 20 minutes, the intercom page heard by everyone, or someone stopping to say hello will guarantee that no work is accomplished in an 8 hour day.**

*E-factors below 40% result in frustrated employees. The best employees will quit and look for a better work environment.*

**Technical workers like their jobs. *The ability to seek gratification from their work is more important than money.* The best employees migrate to companies that provide the best work environments.**

Evidence suggests that top performers in a company typically out perform the low end by a ratio of 10:1 to as much as 20:1.

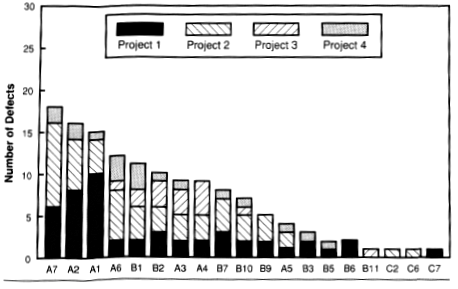
The group of employees above the median generally produces at least twice as much as the group below the median!

**Money saved short term by skimping on space, creature comforts, or decisions that increase distractions in this group is lost everyday from that point on in the form of productivity and quality!**

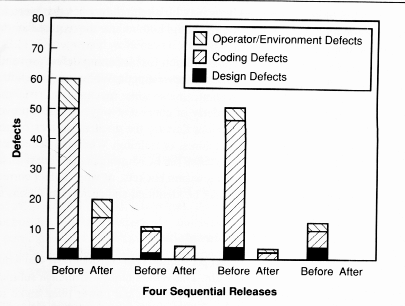
**Cheap is extrodinairly expensive!**

**(DeMarco & Lister)**

**Consider holding important meetings off-site where there are no phones and the rest of the company does not know how to contact the desired individual directly!**

1. Discuss statistical Quality Assurance (SQA) in detail including the Pareto principle and work of Robert Grady and Deborah Caswell.
   1. ***Statistical Quality Assurance implies:***
      1. **Information about software defects is collected and categorized.**
      2. **An attempt is made to trace each defect to its underlying cause (e.g., specification, design, violation of standards, poor communications with the customer).**
      3. **The Pareto principle (80% of the defects can be traced to 20% of all possible causes) is used to isolate the 20% of major error types.**
      4. **Once the vital few categories have been identified, the majority of available resources are committed to correct the problems that have caused these defects.**
   2. **Practitioners should spend their time and money on things that really matter. The purpose of Statistical Quality Assurance (SQA) is to determine what really matters!**
   3. 

**This Pareto distribution, a ranking of the number of defects by cause starting with the most prevalent on the left, is from "Software Metrics; Establishing a Company Wide Program" by Robert B. Grady and Deborah L. Caswell, 1987, p125, Englewood Cliffs, NJ: Prentice-Hall. See also "Measuring and managing software maintenance," IEEE Software, September 1987. Using this data from four programs, Hewlett-Packard was able to establish that *over one third of the defects were constrained to only three categories*. By focusing their process improvement in these areas they reported significant quality improvements.**



**According to Grady more than a third of the errors were due to poor understanding of the user's interface requirements. By establishing an extensive prototyping and design review program, *the number of defects for four successive releases dropped from 25% of the total for the first two releases to less than 10 percent for the third release, and to zero in the forth release.*  This process is called *Cause Analysis*.**

**At least one company using defect prevention methods reports a 50% reduction in defects found during development and a 78% reduction in errors shipped.**

**Typical error categories include:**

* **Incomplete or erroneous specifications (IES).**
* **Misinterpretation of customer communications (MCC).**
* **Intentional deviation from standards (IDS)**
* **Violation of programming standards (VPS).**
* **Error in data definition or representation (EDR).**
* **Inconsistent module interface (IMI).**
* **Error in design logic (EDL).**
* **Inaccurate or erroneous testing (IET).**
* **Inaccurate or incomplete documentation (IID).**
* **Error in programming language translation of design (PLT).**
* **Ambiguous or inconsistent human-computer interface (HCI).**
* **Miscellaneous (MIS).**

1. Discuss developing a test plan in detail. What should be used to develop a plan. What and how should testing be accomplished. Include the specification, design, code and other pertinent information in your response.
2. First explain how a Structured Walk Through is conducted. Second explain why it may be the most important tool available to improve software quality. Explain how they should be done, when, by whom, and how effective they are based on empirical evidence.

***Formal Technical Reviews or***

***Structured Walk Throughs***

***Walk-throughs are most productive when:***

**The material to be discussed is distributed to the attendees for review two or three days before the walk through. *Management must support the position that all attendees do their home work prior to the walk-through to avoid wasting the groups time.***

***The number of attendees is limited to four or five and the session to about one hour.* The project manager may be barred from the walk-through if they are not technically competent or it is felt their presence may inhibit the frank and open exchange of ideas. Education of junior members is a goal of the review process.**

**In some cases, it may be necessary to include an impartial mediator to resolve disputes. Raise issues, don't resolve them. Watch your language, be humble, avoid critical negative phrasing. Stick to the issues. There is always some danger an overly aggressive group will be detrimental to a designer or programmer who is not very sure of their work. On the other hand, an overly aggressive individual may overwhelm the suggestions or criticisms of the group.**

**Technical reviews should specifically "avoid" redesigning systems or programs. The purpose of the meeting is to verify the specification/design/code accomplishes the desired purpose and identify problems. Spontaneous group designs almost always end in failure.**

**Formal notes should be taken and the individual or group that is the subject of the inspection should be expected to respond to the groups suggestions in a reasonable time.**

***The entire "group" is responsible for any flaws or bugs in the final product.***

***To illustrate, assume***

**the cost of a error uncovered during *design cost 1.0* monetary units to correct.**

**The same error uncovered just *before testing commences will cost 6.5 units* (relatively speaking).**

***During testing, 15 units*.**

**If not found until *after release, 60 to 100 units*.**

***Then the development cost comparison:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Errors Found** | **Number** | **Cost Unit** | **Total** |
|  | **Reviews** | **Conducted** |  |
| **During design** | **22** | **1.5** | **33** |
| **Before test** | **36** | **6.5** | **234** |
| **During test** | **15** | **15** | **315** |
| **After release** | **3** | **67** | **201** |
|  |  |  | ***783*** |
|  | **No Review** | **Conducted** |  |
| **Before test** | **22** | **6.5** | **143** |
| **During test** | **82** | **15** | **1230** |
| **After release** | **12** | **67** | **804** |
|  |  |  | ***2177*** |

**M. Fagan in "Design and Code Inspections, and Process Control in the Development of Programs," IBM Technical Report 21.572, December 1974 found that *code inspections after design and coding save about one programming month per 1000 non comment source statements.***

1. State and explain the four methods discussed in class to produce software capable of detecting a run time error has occurred.

**Since errors cluster at boundaries rather than at the center of an input domain, tests should be developed:**

* + 1. **For an input with a range bounded by values *A* and *B*, test cases should be designed with values equal to *A* and *B*, just above and just below *A* and *B* respectively.**
    2. **If an input specifies a number of discrete values, test cases should be developed that exercise the minimum and maximum numbers. Test cases should also be tried that are just above and below the extremes.**
    3. **Guidelines 1 and 2 should be applied to *output conditions.* For example if a table of shuttle re-entry parameters is to be generated, test cases should be generated to produce the minimum and maximum size table.**
    4. **Be sure to test all internal data structures at their boundaries. Programmers do not like to do this.**

***Fact***

**The cost of finding and repairing defects *increases exponentially the latter they are found in the process.***

**As a result, *preventing defects is generally less expensive than finding errors*, even early in the process.**

***Testing can never be expected to locate all the problems* in software.**

**Most organizations do not generate uniformly distributed random errors, rather the *errors they generate fall into repetitive classes*. It doesn't make sense to uniformly spend time and money attempting to improve quality, rather resources should be concentrated in the error prone areas.**

**Programming *changes are highly error-prone*, all changes should be view as potential sources of error injection.**

**As a result World Class Software Organizations collect defect statistics on both the Upstream (specification and design) as well as Down-stream process (coding and testing).**

**System Changeover**

1. **Direct Method**: **Complete one-time conversion from the old system to the new system.** A **“big bang” if it fails!**
2. **Pilot Method: Bring up the entire system for a single user group, but not all users at the same time. This is a “little bang” if it fails.**
3. **Phased (modular) Method: Seeks to divide a system into pieces and install a piece at a time. Minimal shock to user community.**
4. **Parallel: Often preferred in conjunction with the phased approach. Both the old and the new system are run in parallel until the new system operation is verified and the users are ready for the change. Expensive but frequently the safest approach in critical situations. This requires that every transaction be captured and processed on both systems.**

**Alpha Testing – user with developer**

**Beta Testing – user by self**

1. Discuss in detail “Black Team” methodology, management support, utilization, and why they tend to be very successful. A very detailed response with project specific information is expected. Include all areas from which team should be expected to extract test cases and how many test cases they should develop based on program logic.

**The best testing is done by a separate test team.**

A separate team attempts to show the software has errors, not that its correct. Their ego is not attached to the quality of the product as they did not write it.

**By concentrating on testing, they get practice and develop techniques regular programmers have neither the time or inclination to develop.**

The team should be judged solely on how well they "test."

**To prevent interference from management, the team should report to a different manager than the management responsible for developing the software.**

The team should not be allowed to work directly on the software.

**IBM “Black Team”**

 Back in the dawn of time (relatively speaking), there was a company in upper New York State that made large blue computers. The company also made software to run on these computers. Customers of this company were nice enough folks, but just between us, they could be awfully poor sports about software delivered with bugs. For a while, the company put its efforts into training the customers to make them more tolerant of bugs. But this approach didn’t work out, so they bit the bullet and decided to get rid of the bugs instead. The easy and obvious approach was to have the programmers remove all bugs prior to delivery. For some reason, this didn’t work too well either. It seems that the programmers (at least the ones back in those days) were rather too inclined to believe the best of their programs. Try as they might, they couldn’t find the last remaining bugs, so they often declared the software to be done when there were still lots of bugs. Finding the last bug was hard, but some testers were better than others.

The company formed a group of these particularly talented testers and gave them the charter to do final testing on critical software before it was sent to the customers. Thus was born the legendary Black Team. The Black Team was initially made up of people who had proved themselves to be slightly better at testing than their peers. They were slightly more motivated. They also were testing code that had been written by someone else, so they were free of the cognitive dissonance that hampers developers when testing their own programs. All in all, those who formed the team might have expected it to achieve at least a modest improvement in product quality, but they didn’t expect more than that. What they got was much more than that. The most surprising thing about the Black Team was not how good it was at the beginning, but how much it improved during the next year. Some magic was happening: The team was forming a personality of its own. This personality was being shaped by an adversary philosophy of testing that evolved among group members, a philosophy that they had to want and expect to find defects. They weren’t rooting for the developers at all, quite the opposite. They were delighting in submitting the program (and the

programmer) to a sequence that was not just a test, but an ordeal. Bringing your program in for Black Team testing was like appearing before Ming the Merciless.

**Pitiful Earthlings, What Can Save You Now?**

At first it was simply a joke that the tests they ran were mean and nasty, and that the team members actually loved to make your code fail. Then it wasn’t a joke at all. They began to cultivate an image of destroyers. What they destroyed was not only your code but your whole day. They did monstrously unfair things to elicit failure, overloading the buffers, comparing empty files, and keying in outrageous input sequences. Grown men and women were reduced to tears by watching their programs misbehave under the demented handling of these fiends. The worse they made you feel, the more they enjoyed it. To enhance the growing image of nastiness, team members began to dress in black (hence the name Black Team). They took to cackling horribly whenever a program failed. Some of the members grew long mustaches that they could twirl in Simon Legree fashion. They’d get together and work out ever more awful testing ploys. Programmers began to mutter about the diseased minds on the Black Team. Needless to say, the company was delighted. Every defect the team found was one that the customers wouldn’t find. The team was a success. It succeeded as a test group, but more important for our purposes here, it succeeded as a social unit. People on the team got such a kick out of what they were doing that colleagues outside the team were positively jealous. The black outfits and the silly exaggerated behavior were part of the fun, but there was something much more fundamental going on. The chemistry within the group had become an end in itself.

**Footnote**

Over time, members of the team moved on one at a time to other things. Since the team function was important to the company, departing members were replaced immediately. This continued until finally there wasn’t a single

member left of the original group. But there was still a Black Team. The team survived the loss of all of its original staff, and it emerged with its energy and its personality intact.

1. Discuss either decision tables or finite state automata (not both) in detail stating their advantages and disadvantages. Be sure to include how they can be used to minimize the cost of making software changes due to a changing user environment.
2. As you know the push to colonize the new territories has placed a tremendous strain on our resources. It is imperative that you create software to allow us to track all shuttles, X-wing fighters and space stations from a central control facility to be created in each sector of colonized space. Each shuttle, X-wing and space station will be required to identify itself on demand, give its location/destination and present course to the central control facility. Shuttles and space stations must accept course adjustments on demand from the central control facility. Each space station, shuttle, and X-wing have a unique serial number and alphanumeric name to a maximum of 20 characters in length. X-wings are launched and retrieved from space stations. Shuttles are used to transfer personnel and equipment between the central control facility and space stations. Space stations have photon torpedoes and proton guns. X-wings are limited to proton guns. X-wing fighters must protect all other entities from alien vessels which also have proton guns. Shuttles have cargo bays and no armaments. X-wing fighters and shuttles use ion propulsion systems using the same methods to start and stop their engines. All entities utilize the same life support and navigation systems. Space stations use warp engines and require a different procedure to start their engines. **Implement the Object Model in OOD using OMT including an inheritance hierarchy to reduce implementation cost while increasing reliability. You may but are not required to submit timing diagrams or state model.**

**OBJECT MODEL: The object model describes a system in terms of the objects of which it is constructed - it shows their identity, their relationship to other objects, their attributes and operations. The object model provides an essential framework for understanding the dynamic and functional models.**

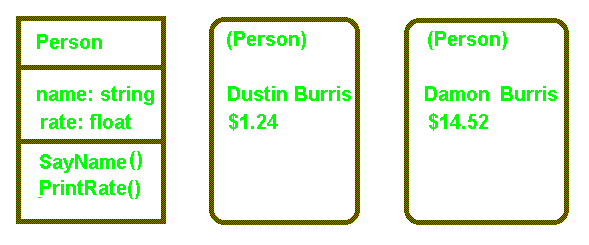
***Object Modeling Technique Notation ( OMT )***

**Notation for Object Classes:**

|  |
| --- |
| **Class Name** |
| **attribute-name-1: data-type-1 = default-value-1**  **attribute-name-2: data-type-2 = default-value-2**  **\* \* \*** |
| **operation-name-1(argument-list): result-type-1**  **operation-name-2(argument-list): result-type-2**  **\* \* \*** |

**A *class diagram* is a pattern or template that describes all possible instances of data and methods that an object of the class might have.**

**An *object or instance diagram* describes a particular object.**



**Class Definition Objects with values**

**Explicit object identifiers are not needed. Each object has its own unique identity. Two "people" (objects) with the same name and pay rate would still be unique and the systems could differentiate between them. You may *invoke methods* on the objects Dustin and Damon to *Pay* them or to ask them to *say their name* (SayName).**

How many shuttles would you like? 3

Shuttle named ShuttleA bound for Freedom course 45.

Shuttle named ShuttleAA bound for Freedom course 45.

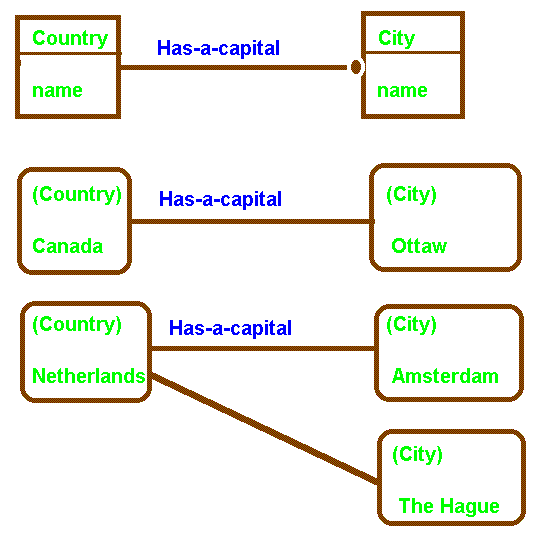
Shuttle named ShuttleAAA bound for Freedom course 45.

ShuttleAA change course to 38.

Shuttle named ShuttleAA bound for Freedom course 38.

***LINKS AND ASSOCIATIONS***

**A *link* is a physical or conceptual association between objects. A link is an instance of an association. A group of links with common structure and semantics is called an *association*. Associations are *inherently bi-directional* and frequently *appear as verbs* in a problem description.**

****

**Cultural/Political capitals**

**One-to-Many Association and Links (1 : M)**



**One-to-One (1 : 1)**

Entity Relationships Expressed

Using Entity-Relationship (ER) Database Diagrams

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Car** |  | **Driver** |  |
|  |  | **1:1** |  |  |
|  |  |  |  |  |
|  | **Manager** |  | **Employee** |  |
|  |  | **1:M** |  |  |
|  |  |  |  |  |
|  | **Manager** |  | **Employee** |  |
|  |  | **M:1** |  |  |
|  |  |  |  |  |
|  | **Rental Car** |  | **Customer** |  |
|  |  | **M:N** |  |  |

**Flat files emphasize the storage of data**. Programmers produce relationships between data items. **Relational databases emphasize storage of data and relationships.** **Object oriented databases emphasize storage of data, user relationships, inheritance, and most importantly object behavior.** In developing a relational definition for a database, the DBA (team) must:

1. determine what datum must be retained,
2. which datum exist only in the presence of other datum (records), and
3. how collections of datum (records, tuples, etcetera) are related.

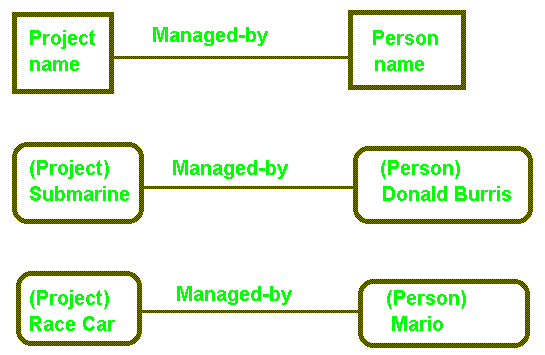
How collections of records are related is the most difficult thing to determine and potentially the most crucial to the long-term health of the database. The ability of users to manipulate data to produce information is directly related to how well the DBA has provided users the ability to utilize linkages.

These linkages or “relations” are a part of the way an organization conducts business. **Only a person intimately involved with the operation of the enterprise will be likely to determine all such relationships. Relationships not anticipated at the time the database is defined may be hard or impossible to add latter**.

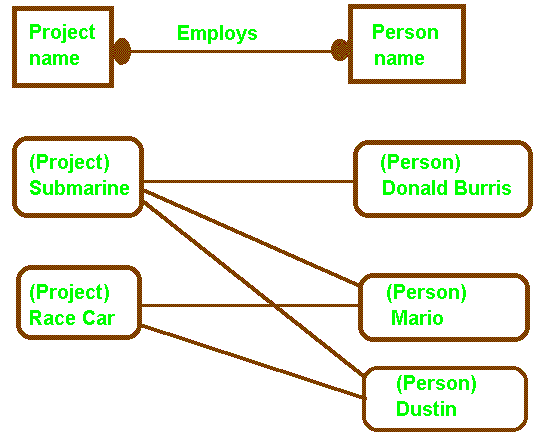
Most business relationships are not ad hoc. A good designer should have no trouble anticipating 95% or more of the relationships a business utilizes and the frequency the relationship is required. These same principles apply to the development of object orient systems.

***Object Oriented Database***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  | **Class Tire** |  |  |  |  |
|  |  |  | **Data Structures**  **--** |  |  |  |  |
|  |  |  | **Methods:**  **M1( ) { -- }**  **M2( ) { -- }**  **Balance( ) { -- }** |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | **Class Armored** |  |  |  | **Class Jet** |  |  |
|  | **Data Structures**  **--** |  |  |  | **Data Structures:**  **--** |  |  |
|  | **Methods:**  **M3( ) { -- }** |  |  |  | **Methods:**  **M4( ) { -- }** |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | **Class Commercial** |  | **Class Military** |  |  |
|  |  |  | **Data Structures:**  **--** |  | **Data Structures:**  **--** |  |  |
|  |  |  | **Methods:**  **--** |  | **Methods:**  **--**  **Balance( ) { -- }** |  |  |
|  |  |  |  |  |  |  |  |

****

***One-to-One* Association and Links for Objects**

****

**Many-to-Many Association and Links**

Many-to-many relationships frequently occur in CAD/CAM applications, networks, or any other application that can be represented by a graph. As in database, the M:N relationship is typically implemented using a bridge relation**. During specification the relationship is defined. The implementation bridge is created during design**.

Many-To-Many Implementation Solution

Is Use of a Bridge Relation in Database Design

\*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Course** |  | **Student** |  |

\*\* An M:N relationships may be reduced to two 1:M relationships via the bridge!

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | Course | Has-Enrolled | Grade | Enrolled-In | Student |
|  |  | 1:M |  | M:1 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | History |  | English |  | Spanish |  | Math |  | Com Sci |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Mary |  |  |  |  |  | B |  | D |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Jim |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Joe |  | B |  | B |  | C |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Sam |  |  |  |  |  | B |  | A |  | B |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Each “grade” in the bridge relation has exactly two pointers. One to represent its membership in the “Has-Enrolled” relation and one to represent its membership in the “Enrolled-In” relation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Grade Record Format | Grade | Course Point | Student Point |  |
|  |  |  |  |  |

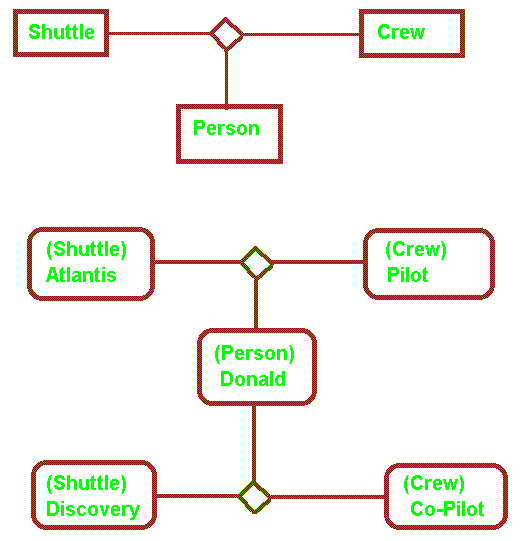
Typically a bridge composite key consists of the key for each one side of each 1:M relationship.

**Relational Database with “bridge” relation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **COURSE** |  |  |  |  |
|  | **Course** | **Room** | **Other** |  |  |
|  | **CS** | LDB 418 |  |  |  |
|  | **Mth** | LDB 216 |  |  | Key |
|  | **Eng** | EVN 212 |  |  | Course |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | **1:M** |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | **GRADE** |  |  |  |  |
|  | **Course** | **Grade** | **Name** |  | Key |
|  | **CS** | A | **Joe** |  | Course+Name |
|  | **CS** | B | **Betty** |  |  |
|  | **CS** | C | **Tom** |  |  |
|  | **Mth** | A | **Sara** |  | **Foreign Key**: |
|  | **Mth** | C | **Mary** |  | **Course** -> Course |
|  | **Eng** | C | **Mary** |  |  |
|  | **Eng** | B | **Betty** |  | **Grade** -> Student |
|  | **Eng** | C | **Sara** |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | **1:N** |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | **Student** |  |  |  |  |
|  | **Name** | **Phone** | **Other** |  | Key |
|  | **Joe** |  |  |  | Name |
|  | **Betty** |  |  |  |  |
|  | **Tom** |  |  |  |  |
|  | **Sara** |  |  |  |  |
|  | **Mary** |  |  |  |  |
|  |  |  |  |  |  |

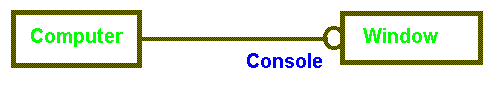
The key for the “bridge” relation is the combination of the keys on the “one” side of the one-to-many relationship. The key components in the bridge relation represent “foreign Keys” pointing to the relations being joined.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Course** |  | **Grade** |  | **Student** |
|  |  |  |  |  |
|  | **Course** |  | **Student** |  |

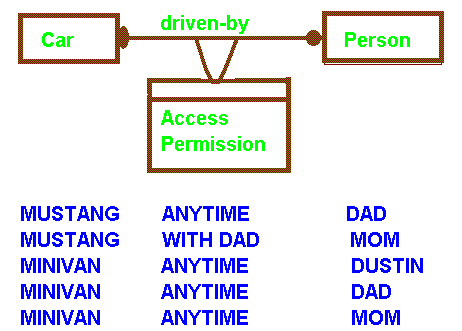


***Ternary* Association and Links**

**Most associations are *binary*, or a special restricted form of *ternary* called *qualified*. While high order relations are possible, they are more complicated to implement and should be avoided when possible.**

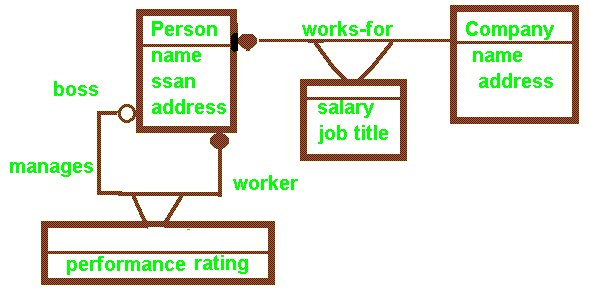


***Zero-or-One Multiplicity***

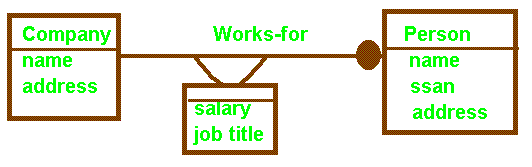
**A computer may have one or none of its windows designated as a console for controlling a network and receiving network message. Console is a *role name* in this example.**

***Link attribute for a many-to-many association***

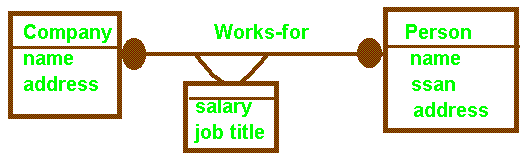
**A *link attribute* is a property of the links in an association. Permission to drive is a property of the link, a joint property of both classes. Attachment to objects of either class would result in loss of information.**

****

**Many *persons* are employed (*works-for*) by the same *company*. As a result of the employment, each person has a *salary* and *job title*. People may work unsupervised or several people may be *managed* by a *boss*. The boss is responsible for *performance ratings*.**



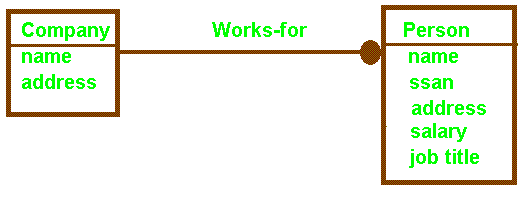
**One-to-Many: Works.**



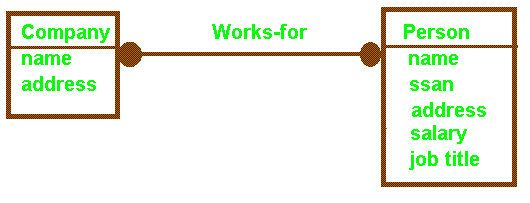
Many-to-Many: Works.

Example: Sam works for SHSU as a PGMR-1 and TDC as a MGR-3

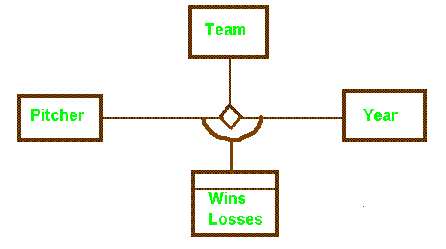
Avoid folding link attributes into one of the classes.



**One-to-Many: Works.**

****

**Many-to-Many: Fails without link attributes. Limits future use.**



Harry Eisenstat Cleveland Indians 1939 6 7

Harry Eisenstat Detroit Tigers 1939 2 2

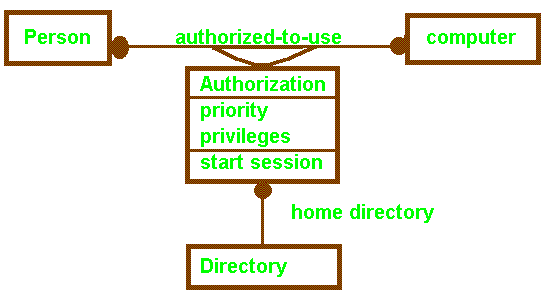
Willis Hudsin Cleveland Indians 1939 9 10

Willis Hudsin Cleveland Indians 1940 2 1

Willis Hudsin Washington Senators 1940 1 2

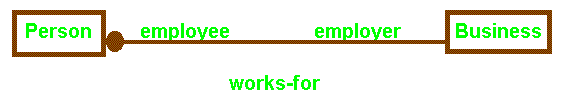
Willis Hudsin St. Louis Browns 1940 0 1

***Link attributes in a ternary association***



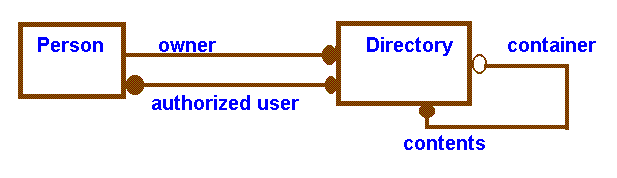
***Modeling an Association as a Class***

**When links can participate in associations with other objects or when links are subject to operations, it is useful to model the association as a class.**

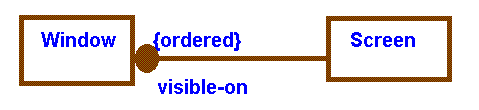
****

***Role names for an association***

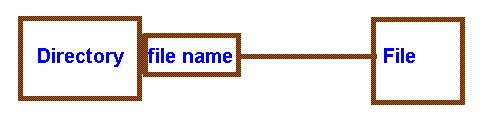
Role names frequently improve the understanding of relationships.



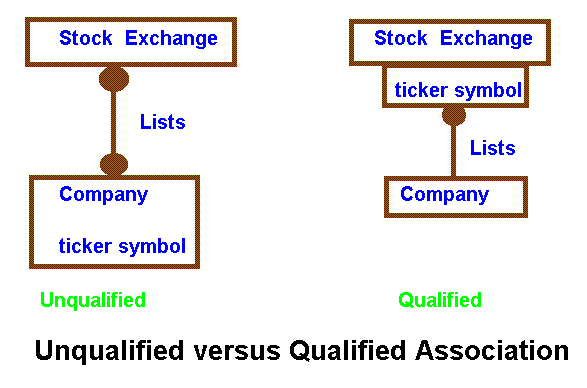
Role names help to distinguish multiple associations between the same pair of classes. They are required to distinguish associations between two objects of the same class, e.g. *boss* and *worker* in the *manages* association using class *person*.

****

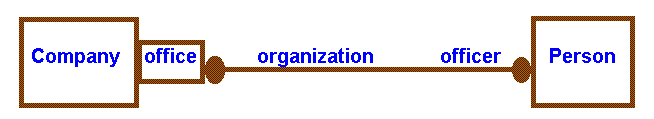
Windows on a screen are usually ordered. Only the topmost window is directly accessible.

****

***Qualification*** reduces a binary relationship to a lower level association. The *Directory + file name* identifies a *unique file* in DOS.



**In the unqualified association, knowing a company's ticker symbol does not identify the stock exchange. In the qualified association, knowledge of the ticker symbol associates the company with a particular stock exchange.**

****

**Many-to-Many Qualified Association**

**Cryogenics President David Burris**

**Cryogenics Treasurer Darlene Burris**

**Cryogenics Board of Directors Damon Burris**

**Cryogenics Board of Directors Dustin Burris**

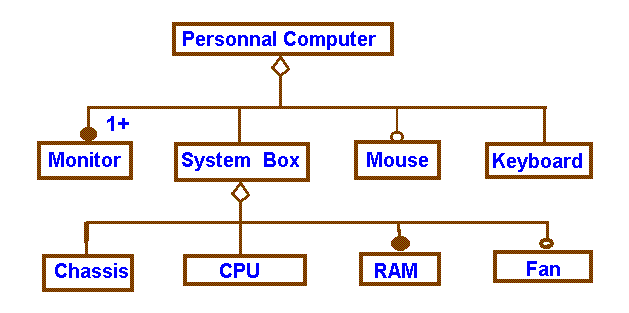
**Cryogenics Board of Directors Donald Burris**

**NASA President Good Person**

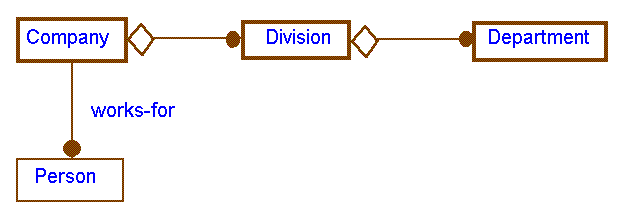
****

***Aggregation***

**Aggregation should be used if the parts do not have a meaningful role separate from the whole.**

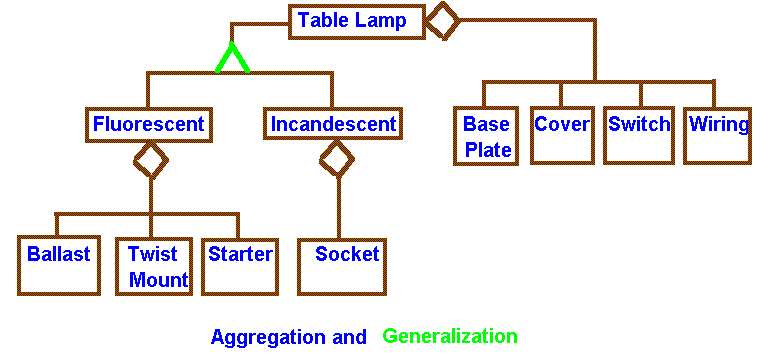
****

**Multilevel Aggregation**

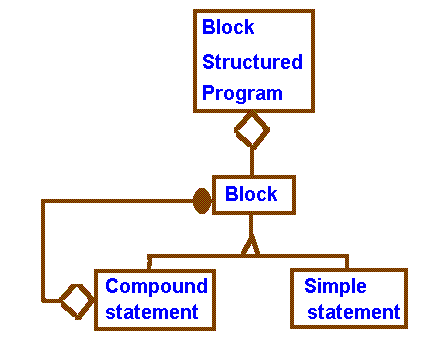
****

***Aggregation and Association***

**If two objects are tightly bound by a part-whole relationship, use aggregation. If they exist separately, use association.**

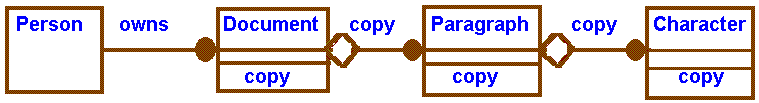
****

***Aggregation* relates instances, two distinct objects are involved where one is part of the other. *Generalization* relates classes. Generalization (inheritance) is a way of structuring the description of a single object.**

****

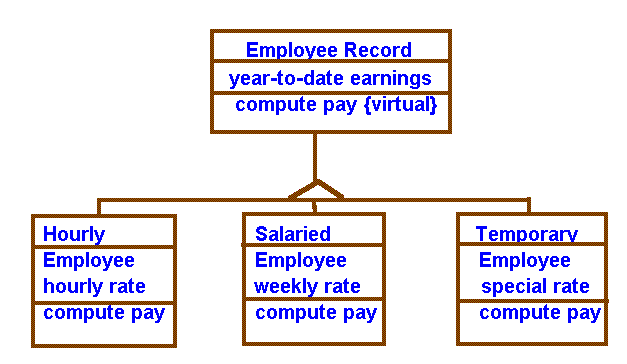
**Recursive Aggregate**

**The OMT notation may be used in conjunction with meta languages like BNF to improve readability. Note “Block” is recursively defined.**

****

***Propagation (Triggering) of Operations***

**To copy a document implies you copy paragraphs, which in turn implies you copy characters.**

****

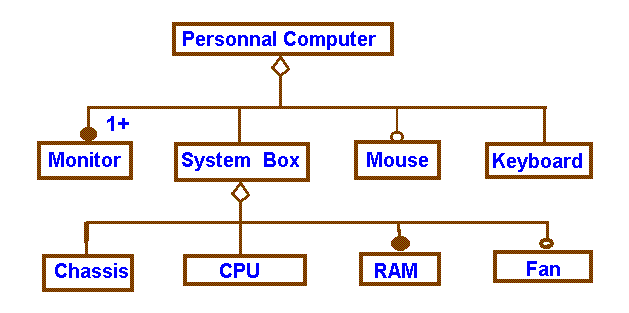
***Virtual Class and Virtual Method***

**A *virtual class* (meta-class) has no direct (concrete) instances. A *concrete class* has distinct instances. Virtual classes help describe and aid development through inheritance. *Virtual methods* indicate a need for "over loading" or polymorphic methods.**

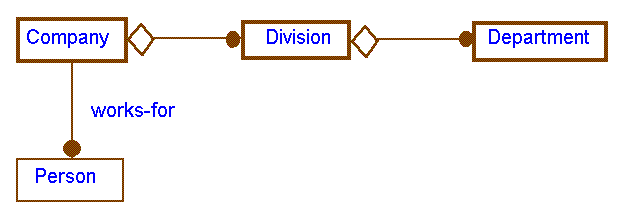
**Flat files emphasize the storage of data**. Programmers produce relationships between data items. **Relational databases emphasize storage of data and relationships.** **Object oriented databases emphasize storage of data, user relationships, inheritance, and most importantly object behavior.**

***Aggregation***

**Aggregation should be used if the parts do not have a meaningful role separate from the whole.**

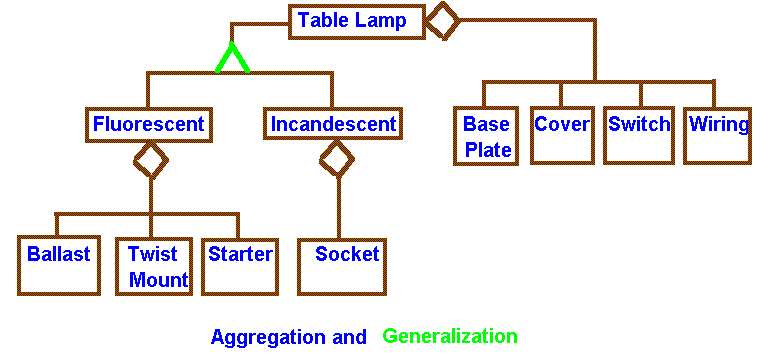
****

**Multilevel Aggregation**

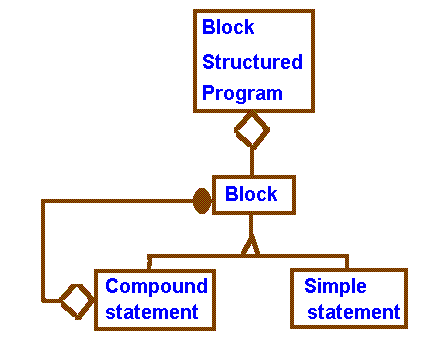
****

***Aggregation and Association***

**If two objects are tightly bound by a part-whole relationship, use aggregation. If they exist separately, use association.**

****

***Aggregation* relates instances, two distinct objects are involved where one is part of the other. *Generalization* relates classes. Generalization (inheritance) is a way of structuring the description of a single object.**

****

**Recursive Aggregate**

**The OMT notation may be used in conjunction with meta languages like BNF to improve readability. Note “Block” is recursively defined.**

***Virtual Class and Virtual Method***

**A *virtual class* (meta-class) has no direct (concrete) instances. A *concrete class* has distinct instances. Virtual classes help describe and aid development through inheritance. *Virtual methods* indicate a need for "over loading" or polymorphic methods.**

1. DeMarco and Lister are well known for “Coding War Games.” Do not discuss the war games. Rather discuss the results emphasizing what their studies showed does improve productivity and quality versus what does not improve productivity and quality. What are the lessons to be learned by industry with respect to staffing (hiring), attracting the best talent, and range of anticipated performance? First discuss the purpose for coding war games from the standpoint of the authors of “Peopleware.” What did they hope to accomplish? Next state the traditional items people felt affected the productivity and quality of software projects. For each item explain the conclusions.

War Games

From our years of running the Coding War Games, we have learned that the sometimes raucous, competitive, no-lose experience can be a delightful source of constructive disorder. Our games are tailored for the software community,

but the concept can be applied to virtually any field. Whatever your work, it can be an enjoyable experience to try your hand at a set of tailored problems, and to be able to compare your performance to a statistical performance

profile of your peers. (Of course, the experience is only enjoyable as long as the security and confidentiality guarantees described in Chapter 8, “You Never Get Anything Done around Here between 9 and 5.” are respected and you are thus assured that the game results won’t be used against you.) War games help you to evaluate your relative strengths and weaknesses and help the organization to observe its global strengths and weaknesses. For these reasons, two of ou client companies began a program of annual war games, used by their employees to gauge improvements in their own skills over time. Once a year, they subjected themselves to the confidential testing process, much as you would submit yourself to a physical exam. For the purpose of stimulating creative disorder, the most effective form of war game calls for participants to take part in teams. The following is one formula for such an exercise, a formula that we have tried out with some success (and enormous amusement):

• Select a small development project or well-defined task as a guinea pig. The best choice is an actual job from your organization, something that requires from one to two person-months of effort. Pick a problem that has some novelty and challenge, but that nonetheless makes broad use of your people’s typical working skills.

• Conduct the project in a normal fashion up through publication of a concrete statement of work.

• Announce a 24-hour Project Tournament to be conducted on an upcoming weekend. Make sure everybody understands that you’re not saving money at the expense of their weekend. Explain that the tournament is run over a weekend so the teams can have the place to themselves, not so you can save on manpower cost. Encourage people to form teams of four each and compete on a totally voluntary basis.

• Distribute the statement of work in advance, along with a statement of rules and objectives.

• On the day of the Tournament, only participants are present. Supply everything they need (food, machines, cots, copiers, conference rooms, whatever). Have all of the teams undertake the same work in head-to-head competition with each other.

• Have facilitators available to enforce the ground rules, ready to head off fatal problems and to make lots of noise over every milestone attained.

• Look for opportunities to make everyone a winner in some sense (elapsed-time winners, robust-product winners, clever-solution winners). Make a big fuss over any and all accomplishments. • Install the winning product, or perhaps several winning products in parallel. Keep careful track over time of product stability, number of defects, level of user acceptance, cost to change, and whatever other parameters affect project success. Report meaningful data back to the teams. When you pull this off successfully, people will tell you they’ve had the most exciting and enjoyable experience of their entire careers; nothing less than that is your goal. Expect to achieve that goal, though it may take a few tries. Some things to keep in mind about experiences like the Project Tournament: First, these affairs cost money. Don’t go into them hoping to gull your staff into building something on a Saturday that you would otherwise have to pay real wages for. Expect to spend several times more on a Project Tournament than you would on a conventional running of the same project. Second, invest a lot of time in making the problem specification particularly solid, bringing your facilitators up to speed, and building in lots of milestones and checkpoints. Third, invest some effort to assure the project’s scope is about the right size for the amount of time allocated (no one has any fun if all the teams are defeated or if the tournament is finished an hour after it begins). Finally, look for opportunities to spend money generously on meals (in one tournament, we ordered lovely picnic lunches catered by a New York City restaurant, had dinner delivered, and dragged everybody off to Chinatown for 2 A .M . snacks). Running the project through a whole night, for some reason, adds to the fun. People love an excuse to get tired together, to push back sleep and let their peers see them with their hair down, unshaved, rumpled, and grumpy, with no makeup or pretense. And it makes them feel more closely bound to each other.

**Measurement of Productivity**

**Teams of software implementers from different organizations compete against each other to complete a series of benchmark coding and testing task in minimal time with minimal defects!**

**Rules:**

1. **The basic unit is a pair of implementers from the same organization. The pair members do not work together, but rather in competition against each other to represent their organization. They also compete against all other teams. Hence an organization is represented by at least two people.**
2. **Both pair members perform exactly the same work designing, coding, and testing a medium sized program to a fixed specification.**
3. **Participants must record how time is spent on a log. They are guaranteed that management will never see the log. It is anticipated that participants will have meetings, hot spots that must be taken care of, and other activates that will keep them from devoting 100% of time to the project.**
4. **After all participant testing is completed; the products are subjected to a standard acceptance test.**
5. **Participants work in their normal work areas using the same hardware, software, and other tools they would normally use.**
6. **All individual results are kept confidential from management. Management gets a summary. Individuals are allowed to see how they stacked up against the competition.**
7. Discuss prototyping in great detail. Your discussion should include the circumstances under which prototyping is most likely to be the best software development paradigm. Include the circumstances under which prototyping is not particularly useful and what should be used. Compare and contrast prototyping, the water fall model, and the spiral model in detail. Explain in detail the circumstances under which prototyping is most likely to be the best software development paradigm. Conversely, when is it not needed and what should be used?
8. Circle the letter preceding all correct statements pertaining to “major goals of OOD and OOP.”
9. Minimize the expense in time, cost, and other resource requirements during original implementation.
10. Maximize re-use of the software components.
11. Minimize the cost maintenance, fixing bugs.
12. Minimize the cost of modification, responding to changes in the environment.
13. Maximize the efficiency of resource utilization.
14. Maximize the ability to optimize software with respect to the hardware/software environment while minimizing the cost of the change.
15. Allow for extension of the language by a user adding abstract data type (ADT’s).
16. Circle the letter preceding all correct statements pertaining to OOD and OOP.”
17. Modify the Body Only: If you modify only the body then in principle it should be possible to recompile the body and link it to the specification. This would automatically update all users without further effort.
18. Modify the Specification: The specification, body, and all user code must be recompiled and linked when data structures or functionality is added to the specification. It will not be necessary to modify any user code however unless the user code wishes to take advantage of new functionality defined in the interface. If functionality is removed from the specification, or the interface for invoking a method is modified, then the user code that exercises these features will also have to be modified.
19. Modification of User Code: Modification of user code should not affect the way objects are utilized by the rest of the system.
20. OOP Modification: Any time the specification or body of a class is modified, the specification, body, and all code utilizing the class must be recompiled and linked to prevent errors.

Briefly explain each of the following with respect to OO specification, design, and programming.

***ABSTRACTION:***

**The essence of abstraction is to extract essential properties while omitting inessential details.**

This consists of focusing on the essential, inherent aspects of an entity and ignoring its accidental properties. In system development, this means focusing on what an object is and does, before deciding how it should be implemented. **Use of abstraction preserves the freedom to make decisions as long as possible by avoiding premature commitments to details. Most modern languages provide data abstraction, but the ability to use inheritance and polymorphism provides additional power.** Use of abstraction during analysis means dealing only with application-domain concepts, not making design and implementation dictions before the problem is understood. Proper use of abstraction allows the same model to be used for analysis, high-level design, program structure, database structure, and documentation. A language-independent style of designs defers programming details until the final, relatively mechanical stage of development.

***MODULARITY:***

**Modularity applies to the physical architecture of the system. It is purposeful structuring to help manage complexity.**

The term modularity is widely used in studies of technological and organizational systems. Product systems are deemed “modular”, for example, when they can be decomposed into a number of components that may be mixed and matched in a variety of configurations. The components are able to connect, interact, or exchange resources (such as energy or data) in some way, by adhering to a standardized interface. Unlike a tightly integrated product whereby each component is designed to work specifically (and often exclusively) with other particular components in a tightly coupled system, modular products are systems of components that are “loosely coupled.” Organizational systems are said to become increasingly modular when they begin to substitute loosely coupled forms for tightly integrated, hierarchical structures. For instance, when the firm utilizes contract manufacturing rather than in-house manufacturing, it is using an organizational component that is more independent than building such capabilities in-house: the firm can switch between contract manufacturers that perform different functions, and the contract manufacturer can similarly work for different firms. As firms in a given industry begin to substitute loose coupling with organizational components that lie outside of firm boundaries for activities that were once conducted in-house, the entire production system (which may encompass many firms) becomes increasingly modular. The firms themselves become more specialized components. Using loosely coupled structures enables firms to achieve greater flexibility in both scope and scale. The firm can switch easily between different providers of these activities (e.g., between different contract manufacturers or alliance partners) compared to building the capabilities for all activities in house, thus responding to different market needs more quickly. However, these flexibility gains come with a price. Therefore, the organization must assess the flexibility gains achievable, and any accompanying loss of performance, with each of these forms.

***LOCALIZATION:***

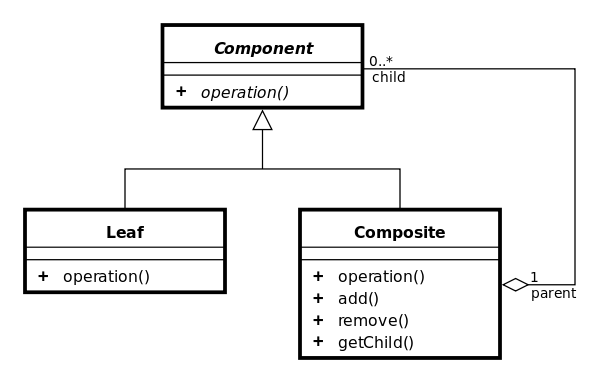
**Localization is primarily concerned with physical proximity, a collection of related data structures and operations on these data structures are collected into a single highly cohesive unit with minimal coupling to the rest of the system.**

Localization In object oriented design approach localization is based on objects. In a design, if there are some changes in the localization approach, the total plan will be violated, because one function may involve several objects, and one object may provide many functions. “Localization is the process of gathering and placing things in close physical proximity to each other”.Metrics should apply to the class as a complete entity. Even the relationship between functions and classes is not necessarily one-to-one. For that reason, metrics that reflect the manner in which classes collaborate must be capable of accommodating one-to-many and many-to-one relationships

***UNIFORMITY:***

**Uniformity simply means that the modules are free from any necessary differences, it usually results from a consistent notation, consistent control structures, and calling sequences.**

Design for uniformity: Child-related operations are defined in the Component interface. This enables clients to treat Leaf and Composite objects uniformly. But type safety is lost because clients can perform child-related operations on Leaf objects. The Composite design pattern emphasizes uniformity over type safety.



***COMPLETENESS:***

**Completeness ensures that all important elements (data and functions) are present.**

It does everything you expect

Does the class capture all of the useful behavior of the thing being modeled to be re-usable? What about future users (reusers) of the class? How much more time does it take to provide completeness? Is it worth it?

***CONFIRMABILITY:***

**”Confirmability implies that we must decompose our system so that it can be readily tested thus helping to make systems more easily modified and maintained.**

a principle

* 1. **Select any 4 of the above and explain in detail how they are related to metrics we have developed previously including Miller’s Law, Graicunas Law, coupling, cohesion, scope-of-effect/scope-of-control, highly factored, etcetera.**

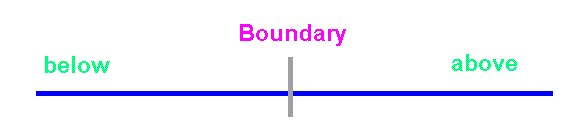
1. Glen Meyers established a set of five goals in "The Art of Software Testing," (Wiley 1979) to be achieved when developing software test cases. Please state and explain the goals. Your discussion should include McCabe’s “Cyclomatic Complexity Measure” and “Basis Path Testing” in detail. How would McCabe’s techniques be employed in industry.

**Common View of Testing:**

**A successful test in one in which no errors are found.**

**Glen Meyers, "The Art of Software Testing," Wiley 1979.**

1. **Testing is a process of executing a program with the intent of finding an error.**
2. **A good test case is one that has a high probability of finding an as yet undiscovered error.**
3. **A successful test is one that uncovers an as yet undiscovered error.**
4. ***A good test reduces by a count greater than one, the number of additional tests cases that must be designed to achieve reasonable testing.***
5. ***A good test tells us something about the presence or absence of a class of errors rather than an error associated only with the specific test at hand.***

******

***Data should include one datum below the boundary, one datum above the boundary, , crossing the boundary, and the boundary itself. Example FICA tax with tax rate and limit.***

***Our objective is to design tests that systematically uncover different classes of errors with a minimum amount of time and effort.***

**McCabe’s Cyclomatic Complexity Measure and Basis Path Testing**

Flow graphs depict logical control flow. Any procedural design representation can be translated into a flow graph.





Cyclomatic Complexity, V(G), is the number of regions in a planar graph. Count all bounded regions and the unbounded area outside the graph. Empirical evidence suggests that modules where V(G) >= 10 are highly prone to errors and extremely difficult to adequately test. There appears to be a distinct relationship between McCabe’s metric, the number of errors existing in the code, and the time required to find and correct such errors.

Quantitative Measure of Difficulty: based on decisions – control flow!

**Basis Path Testing**



Procedural (flow chart)

Design showing program control structure.

A node may contain one or more procedural statements. A set of procedural statements and a decision may map into the same node. Nodes are connected by edges. A node containing a condition is termed a **predicate node**.

Corresponding flow graph assuming no compound conditions are contained in the decision diamonds.



Cyclomatic Complexity is the number of independent paths in the basis set and provides an upper bound on the number of independent tests that must be conducted to ensure every instruction is executed at least once. To be independent, a path must introduce at least one new set of processing statements or conditions.

Paths (there are only four):

**1) 1 - 11**

**2) 1 - 2 - 3 - 4 - 5 - 10 - 1 - 11**

**3) 1 - 2 - 3 - 6 - 8 - 9 - 10 - 1 - 11**

**4) 1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 – 11**

**Cyclomatic Complexity**

Cyclomatic Complexity may be computed in one of three ways to determine the ***basis set***.



1) The number of regions of the flow graph correspond to the Cyclomatic Complexity.

2) V(G) for a flow graph, G, is defined as **V(G) = E - N + 2** where E is the number of flow graph edges and N is the number of flow graph nodes.

3) Cyclomatic Complexity, V(G), for a flow graph G is also defined as **V(G) = P + 1** where P is the number of predicate nodes contained in the flow graph G.

Example:

1) The flow graph has 4 regions.

2) V(G) = 11 edges - 9 nodes + 2 = 4.

3) V(G) = 3 predicate nodes + 1 = 4.

V(G) provides an upper bound on the number of independent paths that comprise the basis set, and by implication, an upper bound on the number of tests that must be designed and executed to guarantee coverage of all program statements.



PDL

**Procedure SORT**

**1: do while records remain**

**read record;**

**2: if record field 1 = 0**

**3: then process record;**

**store in buffer;**

**increment counter;**

**4: elsif record field 2 = 0**

**5: then reset counter;**

**6: else process record;**

**store in file;**

**7a: end if**

**endif**

**7b: end do**

**8: end procedure**



1. Compare and contrast “White Box” and “Black Box” testing in detail. State their goals

**Maximize the amount of delivered function (weighted by years of useful system life) per dollar of total system lifetime cost.**

**The default goal for projects that are not monitored for all development and quality aspects is “Completion in the shortest possible time.” Time is the only variable directly observable by management that does not require special effort!**

**Control of a software project requires measurement of project activities and deliverables!**

***The major goal for most software managers should not be goal attainment, but rather “Goal Alignment.”***

**The proper function of management is not to make people work, but to make it possible for them to work!**

**White/Glass Box Test:**

**Knowing the internal workings of a product, tests can be conducted to be sure that all paths are exercised. Exhaustive testing is seldom feasible (compiler – infinite number of legal/illegal; license plates – large upper bound).**

**White box testing typically discovers from *50% to 70%* of software errors.**

**Most popular form of traditional testing.**

**Black Box Testing:**

**Knowing the specified function that a product was designed to perform, tests are conducted to demonstrate that all functions (individually and as a group) are fully operational.**

**In black box testing there is little or no interest in the internal structure of the software. Black box testing finds the "blind spots" in a design. It also locates areas that work well under normal circumstances yet result in errors whenever data happens to be somewhat unusual.**

**White Box Testing**

**Ideally all datum should be tested exhaustively but this is seldom feasible. For example, assume a license plate consisting of letters and digits in the form LLL-DDD. The number of combinations of data is 26\*26\*26\*10\*10\*10 = 17,576,000 possibilities.**

**A language translator must accept an infinite number of correct programs and reject an infinite number bad programs.**

**We compromise by:**

**guarantee that all independent paths within a module have been exercised at least once**

**exercise all logical decisions on their TRUE and FALSE sides**

**execute all loops at their boundaries and within their operational bounds**

**exercise all internal data structures to assure their validity**

**A) Logic errors and incorrect assumptions are inversely proportional to the probability that a program path will be executed.**

**B) We often believe that a logical path is not likely to be executed when, in fact, it may be executed on a regular basis.**

**C) Typographical errors are random.**

**D) Errors are seldom uniformly distributed.**

**"Bugs lurk in corners and congregate at boundaries." White box testing is far more likely to uncover them.**

**Black Box Testing**

**Most popular forms include:**

***Stress tests* to determine capacity and response limitations.**

**If a minimum amount of data is required for accurate results, *test with the exact amount as well as too little*.**

***Normal Path Test* - verify valid data leads to valid results.**

***Error Path Test* - verify invalid data leads to proper error processing. Normal Path Testing combined with Error Path Testing is also know as *"Equivalence Partitioning."***

**Black box testing should be heuristically directed at all command interfaces:**

**Specify commands with incorrect syntax; use subtle and blatant variations of the correct form.**

**Provide syntactically correct input that is out of sequence or specified at the wrong time. Enter data above, below, and not in the valid range.**

**Type a partially correct command then terminate the command entry.**

**Omit all commands, just hit return.**

**Generate a system interrupt immediately after a command is entered.**

**Check for incorrect or missing functions, functional validity.**

**Look for interface errors.**

**Check for errors in data structures or external database access.**

**Check for initialization / termination errors.**

**Is the systems sensitive to certain inputs or ranges**

**Since errors cluster at boundaries rather than at the center of an input domain, tests should be developed:**

**1) For an input with a range bounded by values *A* and *B*, test cases should be designed with values equal to *A* and *B*, just above and just below *A* and *B* respectively.**

**2) If an input specifies a number of discrete values, test cases should be developed that exercise the minimum and maximum numbers. Test cases should also be tried that are just above and below the extremes.**

**3) Guidelines 1 and 2 should be applied to *output conditions.* For example if a table of shuttle re-entry parameters is to be generated, test cases should be generated to produce the minimum and maximum size table.**

**4) Be sure to test all internal data structures at their boundaries. Programmers do not like to do this.**

***Fact:***

**The cost of finding and repairing defects *increases exponentially the latter they are found in the process.***

**As a result, *preventing defects is generally less expensive than finding errors*, even early in the process.**

***Testing can never be expected to locate all the problems* in software.**

**Most organizations do not generate uniformly distributed random errors, rather the *errors they generate fall into repetitive classes*. It doesn't make sense to uniformly spend time and money attempting to improve quality, rather resources should be concentrated in the error prone areas.**

**Programming *changes are highly error-prone*, all changes should be view as potential sources of error injection.**

**As a result World Class Software Organizations collect defect statistics on both the Upstream (specification and design) as well as Down-stream process (coding and testing).**

***Statistical Quality Assurance implies:***

**1) Information about software defects is collected and categorized.**

**2) An attempt is made to trace each defect to its underlying cause (e.g., specification, design, violation of standards, poor communications with the customer).**

**3) The Pareto principle (80% of the defects can be traced to 20% of all possible causes) is used to isolate the 20% of major error types.**

**4) Once the vital few categories have been identified, the majority of available resources are committed to correct the problems that have caused these defects.**

**Practitioners should spend their time and money on things that really matter. The purpose of Statistical Quality Assurance (SQA) is to determine what really matters!**

1. Discuss Parkinson’s Law, the survey by Lawrence and Jeffery, and Metric premise by Weinberg-Schulman in detail.
   1. **Parkinson’s Law: *Work expands to fill the allocated time*. Without deadlines the worker bees will not work at maximum capacity.**

**Parkinson’s Law**

***“Work expands to fill the time allotted for the task.”***

**Northcote Parkinson (1954) was a British humorist like Will Rogers in the United States. He contrived the law attempting to describe the British Post Office System. In his view it was a bureaucracy where you could not get ahead by merit or fired for lack of merit. The only way to the top was via death and whom you know. There is nothing scientific about the law.** (unionized universities)

**Parkinson’s law may be true on an assembly line or in a bureaucracy. It is seldom true for computer professional who derive satisfaction form their jobs.**

**A variation of Parkinson’s law is likely true.**

**Organizational busy work will expand to fill the working day! (public school teachers)**

A good manager attempts to shield their people from as much of this “busy” work as possible.

**Your people could have found jobs that pay just as much for less effort in college. In fact they may not have even had to go to college. They have sacrificed a portion of their lives and monetary commitment to reach their current level of performance and job position. Usually because they genuinely enjoy what they are doing! A manager’s job is to allow them their enjoyment.**

**Results of a 1985 Survey by Michael Lawrence and Ross Jeffery, University of New South Wales, Australia.**

**The goal was to determine the effect on productivity by various influences on estimating. These include:**

1. **Managers are the best estimators for projects. Developers should not be given more time than is actually required or the additional time will be squandered. In fact many believe the deadline should be a little shorter than what is actually required to encourage high productivity and some voluntary overtime to meet the attainable goal.**
2. **Development staff is the best estimators of time to complete a project. They know what they can accomplish better than anyone else. Hence, deadlines for project deliverables should be established by development staff (an almost folkloric belief in our industry).**
3. **Deadlines should be set by a systems analyst who will have absolutely no involvement in the project other than to estimate the most realistic delivery schedule.**

**All measurements of productivity were made using Barry Boehm’s CoCoMo methodology. CoCoMo is a unit less number. The higher the CoCoMo number the higher the productivity.**

**Actual industrial projects were used to collect the data.**

Results of 1985 Survey by Lawrence and Jeffrey

|  |  |  |
| --- | --- | --- |
| **Effort Estimate Prepared by:** | **Average Measured Productivity** | **Number of Projects** |
| Developer Alone | **8.0** | **19** |
| **Supervisor alone** | **6.6** | **23** |
| **Developer and Supervisor Compromise** | **7.8** | **16** |
| **Systems Analyst (not involved with project).** | **9.5** | **21** |
| **Control Group**  **(No deadline, wake me when through)** | **12.0** | **24** |

**Projects where time pressure was not used as an inducement to increase productivity experienced the highest productivity.**

**The best management strategy might be “Just wake me up when you are done.”**

**The best estimate of actual performance was by a systems analyst that did not participate in the project except to project the most likely time to completion with a fixed set of resources.**

**Developers tend to under estimate the time they require to complete a project. Apparently their egos get involved.**

Managers tend to be more interested in what is desired, or must occur, as opposed to what is possible with a fixed set of resources.

**The proper function of management is not to make people work, but to make it possible for them to work!**

**Weinberg-Schulman Experiment**

**Metric Premise**

**Rational, competent men and women can work to effectively maximize any single observed indication of success.**

**Assign five project teams to maximize one of five project goals.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Goal** | **Shortest Completion Time** | **Minimize Program Size** | **Minimize Data Space** | **Maximize Program Clarity** | **User Friendly Interface** |
| **Shortest Completion Time** | **1** | **4** | **4** | **5** | **3** |
| **Minimize Program Size** | **2 – 3** | **1** | **2** | **3** | **5** |
| **Minimize Data Space Used** | **5** | **2** | **1** | **4** | **4** |
| **Maximize Program Clarity** | **4** | **3** | **3** | **1 – 2** | **2** |
| **User Friendly Interface** | **2 – 3** | **5** | **5** | **1 – 2** | **1** |

**Highest rating is 1, the lowest rating is 5.**

Teams generally achieved their assigned goal.

**To achieve the assigned goal they typically sacrificed other goals.**

**The team assigned to “shortest competition time” sacrificed the most with respect to other goals.**

1. Discuss subliminal suggestion, and measures of software quality in detail.

**Subliminal Suggestion**

**“Subliminal Suggestion” is the use of background music, color schemes, etceteras, too influence productivity.**

#### Production Line versus Professionals

1. **Music**
2. **Wall Colors (dark versus warm, pastel versus harsh)**
3. **Films (Jaws) {fall off seats, record popcorn & drink sales}**

**1960’s Cornell University research indicates that workers in noisy – interrupted environments are measurably less creative.**

The use of white noise/mood music can significantly influence productivity in production line environments. It frequently reduces productivity in white-collar professional environments.

Much of the repetitive daily work activity of the professional is done as serial processing using the symbolic manipulative abilities of the left brain.

The creative leap requires right-brain function. If it is listening to 1001 strings of Muzak or noise, the creative leap is lost.

**Information Technology Workers (IT) have been shown to be 30% more likely to produce zero defect work when provided with a quality environment.**

**Typical Quality Measures**

1. **Mean-Time-Between Failure (MTBF). How long on the average does a product function properly before breaking? This should probably be weighted by the number of units of the product in use, product size and cumulative actual time of use.**
2. **Mean-Time to Repair (MTTR). When a product breaks, how long is required on average to fix it.**
3. **Systems Availability = MTBF / ( MTBF + MTTR )**
4. **User Change Request Rate measured per volume of code or delivered functionality.**
   1. **New functionality (missed specification, skipped in design or implementation: looking for patterns).**
   2. **Modification of existing functionality (missed in specification, not understood correctly at the time of specification, changing environment, poor communication).**
   3. **Functionality not utilized.**

**Quality versus Excellence**

**As designers and implementers, we tend to tie our self-esteem strongly to the quality of the product we produce – not the quantity of the product.**

**For some reason there is little personnel satisfaction in turning out large amounts of mediocre products even though that may be exactly what is required for a given situation.**

**For example, producing the best user interface at higher cost at a latter date for a select market of connoisseurs may not be nearly as important economically as producing the first user interface cheaply and mass marketing it to a less demanding group of connoisseurs. But it will be more satisfying?**

**If you suggest constructive criticism with respect to the quality of the product you risk the developer interpreting your comments as a vicious personnel attack.**

**A major arouser of emotion in the work place is a perceived attack on self-esteem.**

**Any action by management that is perceived by the development staff as affecting quality is likely to invoke a negative emotional response!**

**What is acceptable quality?**

**Builder: Since their self-esteem is tied to the product, the minimum standard is typically the highest quality perceived to have ever been attained in the past.**

**Managers: Managers tend to think of quality as just something that may be supplied in varying degrees according to the pocketbook of the market place. It’s like toppings on a baked potato. The more you are willing to spend the more topping that will be supplied.**

**The Market: “The market does not give a damn about that much quality. If the features you depend on work, who cares about the rest.” People may talk in glowing terms about quality or complain bitterly about its absence. The acid test however is what they are willing to pay for quality.**

**Fact of life for developers: Quality is accomplished during specification, design, and implementation. It may not be economically added as an afterthought at a later date, if at all.**

**Standard Western Quality Quandary: Managers / development staff must determine quality standards based on what the markets wants balanced by what they are willing to pay (or our impression of this tolerance).**

**The Flight from Excellence**

**Allowing the standard for quality to be set by our perception of what the buyer/market place is willing to live with and what they are willing to spend as opposed to higher quality goals represents a flight from excellence.**

**Managers typically associate high quality with excessive time spent in development and testing.**

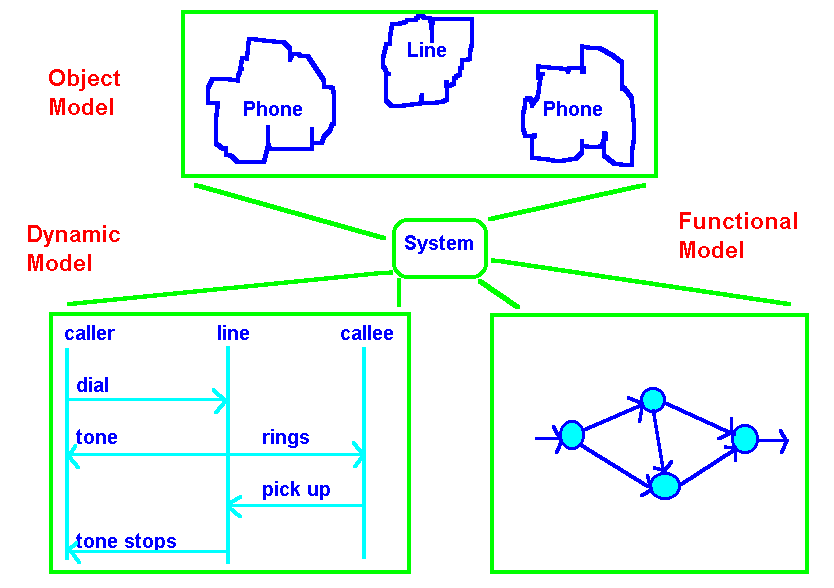
**A typical manager might be quoted as saying “Some of my folks would tinker forever with a task all in the name of quality!”**

**For example, assume we can extrapolate from past empirical data that the mean time between failures of 3.4 weeks could be extended to 2 months if we spend 3 more weeks in development.**

**Expect to see some squirming (Olympic class hemming and hawing) from potential customers who want the product now as well as management desiring to sell the product. Typically they will state they are as quality conscious as the next person but three weeks represents real money to them.**

**Note they seldom consider the lost revenues that are required in the future to correct mistakes caused by using software not yet ready for the market (,e.g. DBase II, III, and IV - same control key saved file in version II deleted file in IV)!**

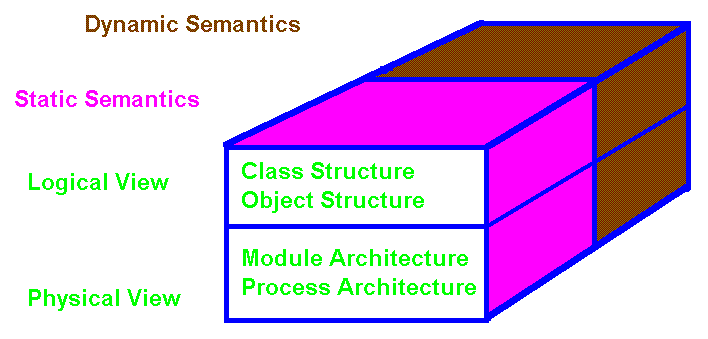
1. Discuss Object Oriented Design (OOD) in great detail. Include the 3 models most design will require and what is included in great detail. You must clearly explain how the models are related.

****

***OBJECT MODEL:* The object model describes a system in terms of the objects of which it is constructed - it shows their identity, their relationship to other objects, their attributes and operations. The object model provides an essential framework for understanding the dynamic and functional models.**

***DYNAMIC MODEL:* The dynamic model describes those aspects of a system related to time and the sequencing of operations - events that mark changes, states that define the context for events, and the organization of events and states.**

***FUNCTIONAL MODEL:* The functional model describes those aspects of a system concerned with transformations of values function mappings, constraints, and functional dependencies. This model captures what a system does without regard for how or when it is done.**



## Booch Models for Object Oriented Design

***LOGICAL VERSUS PHYSICAL MODELS:***

**What classes exist and how are those classes related? Class Diagram.**

**What mechanisms are used to regulate how objects collaborate? Object Diagrams.**

**Where should each class and object be declared? Module Diagrams.**

**To what processor should a process be allocated, and for a given processor, how should its multiple processes be scheduled? Process Diagram.**

**STATIC VERSUS DYNAMIC SEMANTICS:**

**It is difficult to describe a dynamic event in a static medium. In object oriented solutions we attempt to express dynamic semantics through: state transition diagrams, and timing diagrams.**

# OBJECT MODELS

**OBJECT MODEL: The object model describes a system in terms of the objects of which it is constructed - it shows their identity, their relationship to other objects, their attributes and operations. The object model provides an essential framework for understanding the dynamic and functional models.**

**DYNAMIC MODEL: The dynamic model describes those aspects of a system related to time and the sequencing of operations - events that mark changes, sequences of events, states that define the context for events, and the organization of events and states.**

**FUNCTIONAL MODEL: The functional model describes those aspects of a system concerned with transformations of values, functions mappings, constraints, and functional dependencies. This model captures what a system does without regard for how or when it is done.**

***Ojbects***

***Objects have:***

1. **Attributes**
2. **Operations and Methods**
3. **Links and Associations**
4. **Exhibit Multiplicity**
5. **Link Attributes**

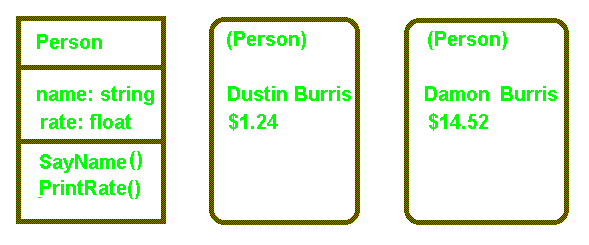
***Object Modeling Technique Notation ( OMT )***

**Notation for Object Classes:**

|  |
| --- |
| **Class Name** |
| **attribute-name-1: data-type-1 = default-value-1**  **attribute-name-2: data-type-2 = default-value-2**  **\* \* \*** |
| **operation-name-1(argument-list): result-type-1**  **operation-name-2(argument-list): result-type-2**  **\* \* \*** |

**A *class diagram* is a pattern or template that describes all possible instances of data and methods that an object of the class might have.**

**An *object or instance diagram* describes a particular object.**

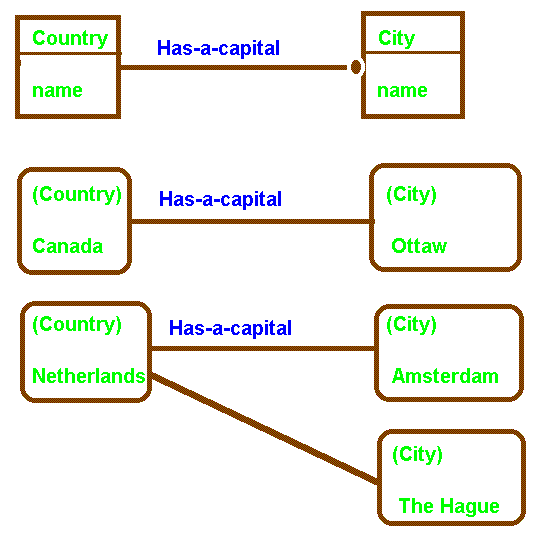


**Class Definition Objects with values**

**Explicit object identifiers are not needed. Each object has its own unique identity. Two "people" (objects) with the same name and pay rate would still be unique and the systems could differentiate between them. You may *invoke methods* on the objects Dustin and Damon to *Pay* them or to ask them to *say their name* (SayName).**

***LINKS AND ASSOCIATIONS***

**A *link* is a physical or conceptual association between objects. A link is an instance of an association. A group of links with common structure and semantics is called an *association*. Associations are *inherently bi-directional* and frequently *appear as verbs* in a problem description.**

****

**Cultural/Political capitals**

**One-to-Many Association and Links (1 : M)**



**One-to-One (1 : 1)**

1. Explain in great detail the meaning of public, protected, and private modifiers with respect to inheritance with examples. You may use our Object Oriented Design (OOD) notation (OMT/UML used in question 6) to implement examples as opposed to code if desired.