Unit 3

Lecture 1: Dimensions of Project Monitoring & Control, Earned Value Analysis,

- Monitoring –collecting, recording, and reporting information concerning project performance that project manger and others wish to know
- Controlling –uses data from monitor activity to bring actual performance to planned performance
- Why do we monitor?
- What do we monitor?
- When to we monitor?
- How do we monitor?

Why do we monitor?

- Simply because we know that things don't always go according to plan (no matter how much we prepare)
- To detect and react appropriately to deviations and changes to plans

What do we monitor?

- Men (human resources)
- Machines
- Materials
- Money

- Space
- Time
- Tasks
- Quality/Technical Performance

Input:

- Time
- Money
- Resources
- Material Usage
- Tasks
- Quality/Technical Performance

Output:

- Progress
- Costs
- Job starts
- Job completion
- Engineering / Design changes
- Variation order (VO)

When do we monitor?

- End of the project
- Continuously
- Regularly
- Logically
- While there is still time to react
- As soon as possible
- At task completion
- At pre-planned decision points (milestones

Where do we monitor?

- At head office?
- At the site office?
- On the spot?
- Depends on situation and the 'whats'

How do we monitor

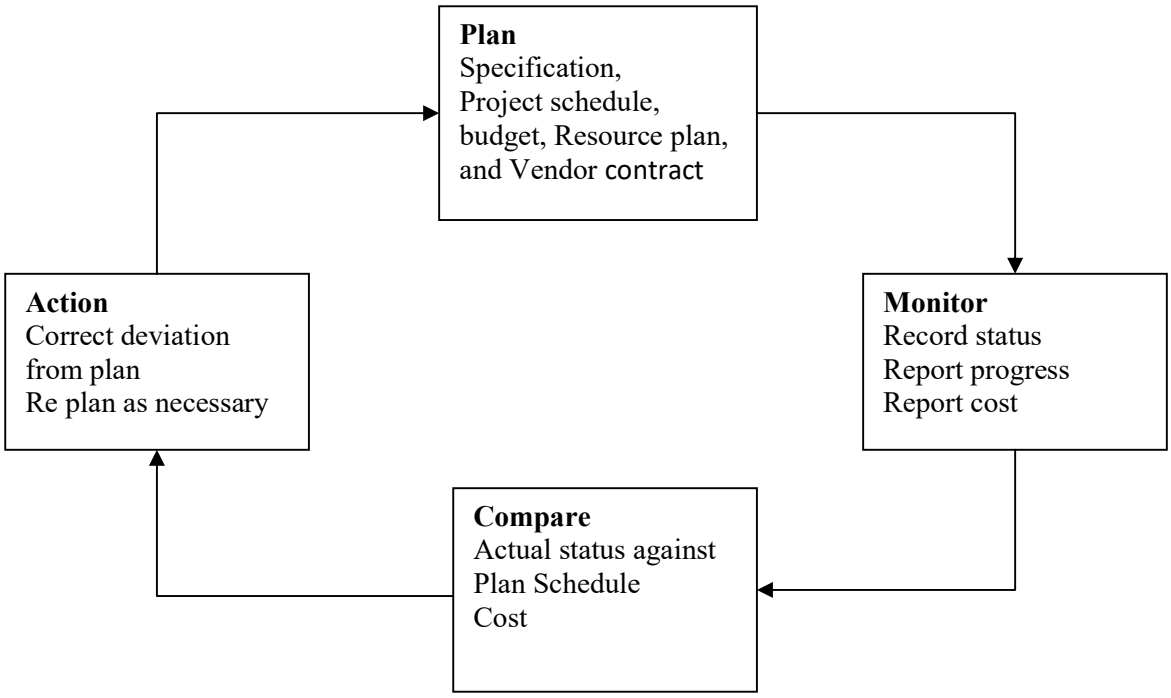
- Through meetings with clients, parties involved in project (Contractor, supplier,etc.)
- For schedule –Update CPA, PERT Charts, Update Gantt Charts
- Using Earned Value Analysis
- Calculate Critical Ratios
- Milestones
- Reports
- Tests and inspections
- Delivery or staggered delivery
- PMIS (Project Management Info Sys) Updating

Meetings –Some monitoring issues

- What problems do you have and what is being done to correct them?
- What problems do you anticipate in the future?
- Do you need any resources you do not yet have?
- Do you need information you do not have yet?
- Do you know anything that will give you schedule difficulties?
- Any possibility your task will finish early/late?

Will your task be completed under/over/on budget?

Project Control Cycle



Project Control

Control –process and activities needed to correct deviations from plan

Control the triple constraints

- time (schedule)
- cost (budget, expenses, etc)
- Performance (specifications, testing results, etc.)

Techniques for monitoring and control

- Earned Value Analysis
- Critical Ratio

Earned Value Analysis

- A way of measuring overall performance (not individual task) is using an aggregate performance measure -Earned Value
- Earned value of work performed (value completed) for those tasks in progress found by multiplying the estimated percent physical completion of work for each task by the planned cost for those tasks. The result is amount that should be spent on the task so far. This can be compared with actual amount spent.
- Methods for estimating percent completion

- The 50-50 estimate. 50% is assumed when task is begun, and remaining 50% when work completed.
- 0-100% rule. This rule allows no credit for work until task is complete, highly conservative rule, project always seem late until the very end of project when everything appears to suddenly catch up
- Critical input rule. This rule assigns progress according to amount of critical input that has been used. Labor or skilled dependent, machine critical input –buy machine complete task –may be misinformation
- Proportional rule. This rule divides planned (or actual) time-to-date by total scheduled time(or budgeted (or actual) cost-to-date by total budgeted cast] to calculate percent complete. This is commonly used rule.
- Refer to earned value chart –basis for evaluating cost and performance to date
- If total value of the work accomplished is in balance with the planned (baseline) cost, and actual cost then top mgmt has no particular need for a detailed analysis of individual tasks
- Earned value concept –combines cost reporting & aggregate performance reporting into one comprehensive chart
- Baseline cost to completion –referred to as budget at completion (BAC)
- Actual cost to date –referred to as estimated cost at completion (EAC)
- Identify several variances according to two guidelines
  1. A negative variance is 'bad'

  2. Cost and schedule variances are calculated as earned value minus some other measure

Earned Value Chart –basis for evaluating cost & performance to date:

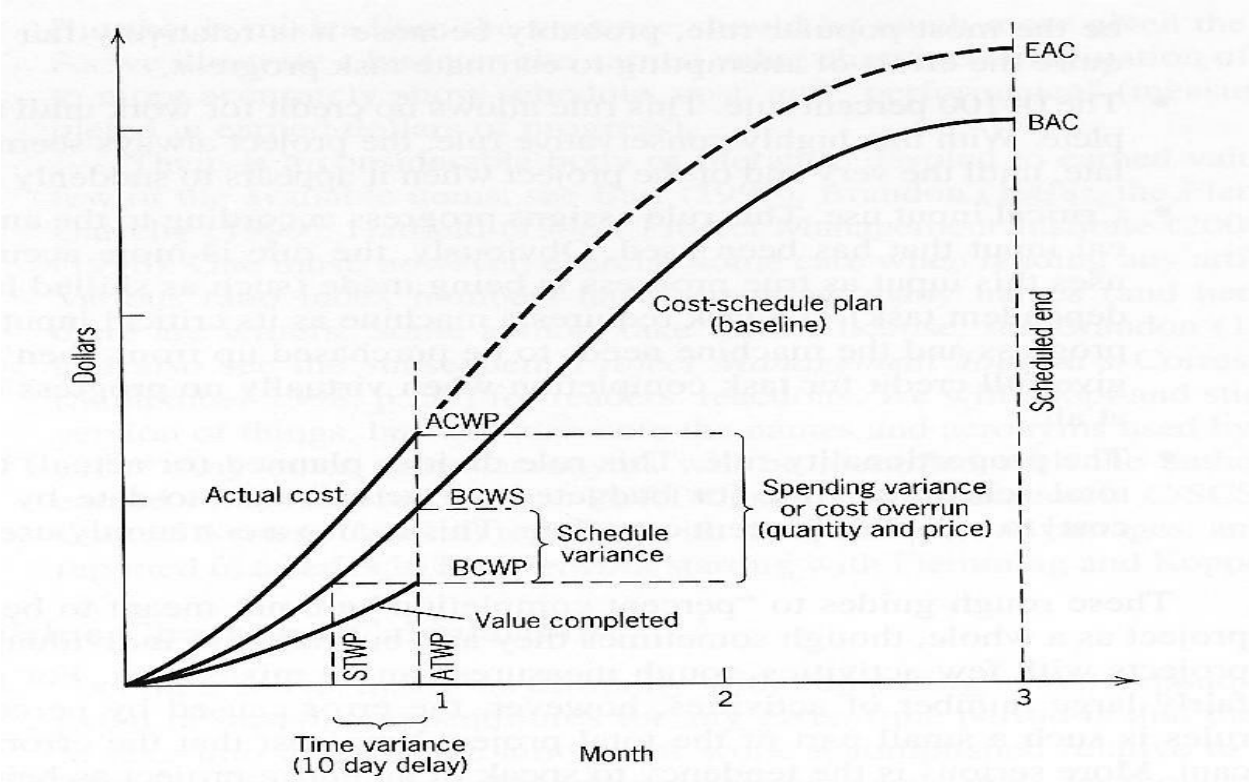


Figure 10-7 Earned value chart.

Earned Value Analysis –Variances:

4 types of variances:

- **Cost (spending)variance (CV)**–difference between budgeted cost of work performed (earned value) (BCWP) and actual cost of that work (ACWP)
- **Schedulevariance (SV)**–difference between earned value (BCWP) and cost of work we scheduled to perform to date (BCWS)
- **Timevariance (TV)**–difference between time scheduled for work performed (STWP) and actual time to perform it (ATWP)

Earned Value Variance –Formula

CV = BCWP –ACWP (negative value -cost overrun)
SV = BCWP –BCWS (negative value -behind schedule)
TV = STWP –ATWP (negative value -delay)

Index (Ratios)

Cost Performance Index (CPI) = BCWP/ACWP
Schedule Performance Index (SPI) = BCWP/BCWS
Time Performance Index (TPI) = STWP/ATWP

EXAMPLE

Assume that operations on a Work Package cost RM 1,500 to complete. They were originally scheduled to finish today. At this point, we actually spent RM1,350. And we estimate that we have completed two thirds (2/3) of the work. What are the cost and schedule variances?
CV = BCWP –ACWP = 1500 (2/3) –1350 = **-350**
SV = BCWP –BCWS = 1500 (2/3) –1500 = **-500**
CPI = BCWP/ACWP = 1500(2/3)/1350 = **0.74**
SPI = BCWP/BCWS = 1500(2/3)/1500 = **0.67**
Spending higher than budget, and given what we have spent, we are not as far along as we should be (have **not completed as much work as we should have**)

Note: Possible to have one of indicators to be favorable while the other unfavorable

- Might be ahead of schedule and behind costs
- Six possibilities (see figure next slide)

6 Possibilities Earned Value Analysis

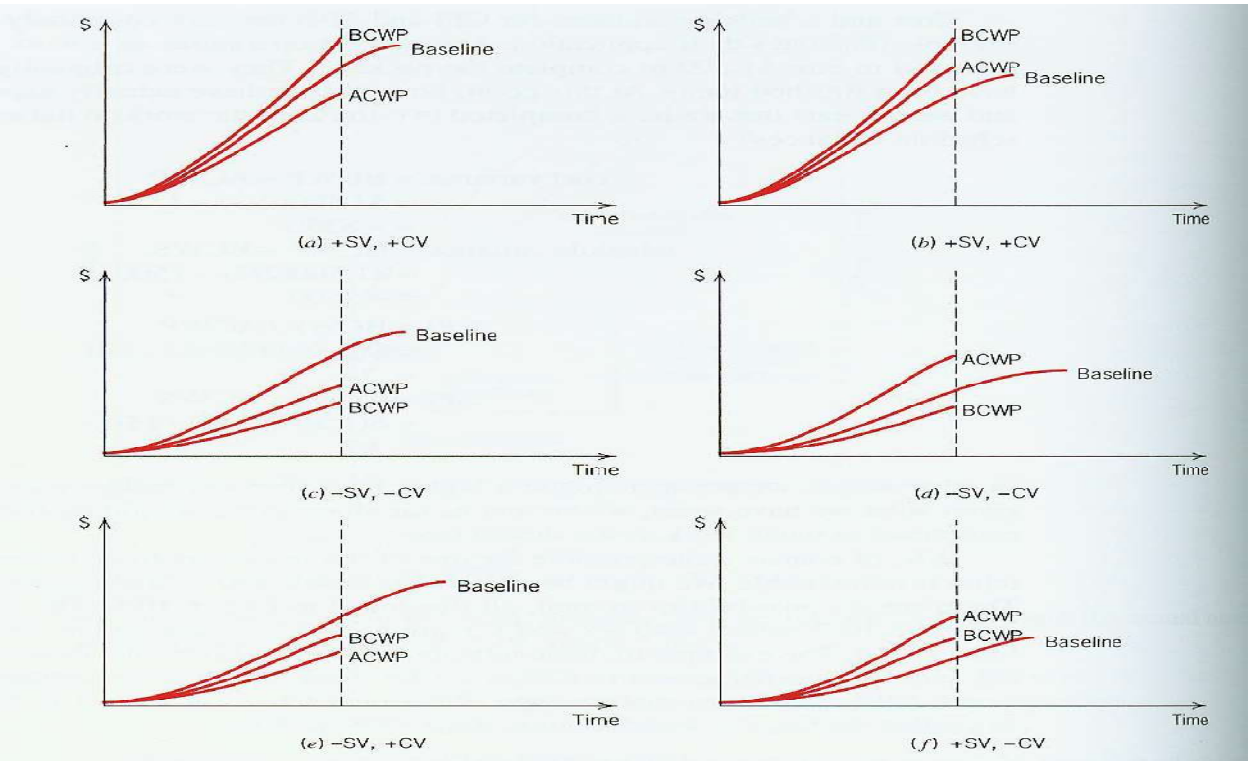
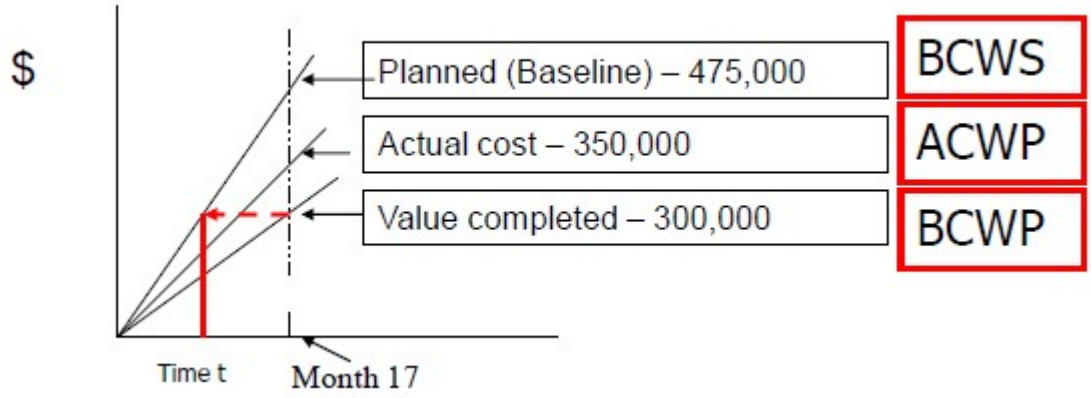


Figure 10-8 Six possible arrangements of ACWP, BCWP, and Baseline resulting in four combinations of positive and negative schedule variance (SV) and cost variance (CV). (Figure 10-7 is arrangement *d*.)

EXERCISE

A project to develop a country park has an actual cost in month 17 of $350,000, a planned cost of $475,000, and a value completed of $300,000. Find the cost and schedule variances and the three indexes.



Solution

BCWS = 475,000
BCWP = 300,000
ACWP = 350,000
CV = 300,000 –350,000 = -50,000 (negative value -cost overrun)
SV = 300,000 –475,000 = -175,000 (negative value -behind schedule)
Cost Performance Index (CPI) = BCWP/ACWP = 300/350 = 0.86
Schedule Performance Index (SPI) = BCWP/BCWS = 300/475 = 0.63
Time Performance Index (TPI) = STWP/ATWP
Scheduled Time Work Performed (STWP) can be estimated
Time t = Schedule Variance/Slope of Planned costs = -175,000/ (475,000/17) = -6.26 months
Time Difference= 17-6.26 = 10.74
TV = 10.74/17 = 0.63
CV = BCWP –ACWP
SV = BCWP –BCWS

Critical ratio

- Sometimes, especially large projects, it may be worthwhile calculating a set of critical ratios for all project activities
- The critical ratio is
- actual progress budgeted cost
- scheduled progress actual cost
- If ratio is 1 everything is probably on target
- The further away form 1 the ratio is, the more we may need to investigate

Critical ratio example

Calculate the critical ratios for the following activities and indicate which are probably on target and need to be investigated.

Activity	Actual progress	Scheduled Progress	Budgeted Cost	Actual cost	Critical ratio (CR)
A	4 days	4 days	60	40	
B	3 days	2 days	50	50	
C	2 days	3 days	30	20	
D	1 day	1 day	20	30	
E	2 days	4 days	25	25	

Critical ratio example

- Can be on schedule and below budget (Act A) Why so good? Cutting corners?
- Can be behind schedule but below budget (Act C)
- Can be on budget but physical progress lagging (Act E)
- Can be on schedule but cost running higher than budget (Act D)
- On budget ahead of schedule (Act B)

Summary

- Need proper project monitoring and control mechanisms
- Tools available to help in monitoring and controlling activities
- There are human control and management aspects not covered here

Lecture 2: **Error Tracking, Software Reviews,**

Error tracking can also be used to estimate the progress of the project. In this case we track errors in work products (requirement specifications, design documents, source code etc) to assess the status of a project.

The process works as follows:
We collect error related metrics over many projects and determine our defect removal efficiency in the following manner:
Defect removal efficiency, DRE = E / (E+D), where
- E – errors found before shipment
- D – errors found during operation

It provides a strong indication of the effectiveness of the quality assurance activities.
Now let us assume that we have collected the following errors and defect data over the last 24 months:
- Errors per requirement specification page – E_{req}
- Error per component – design level – E_{design}
- Errors per component – code level – E_{code}
- DRE – requirement analysis
- DRE – architectural design
- DRE – coding

We now record the number of errors found during each SE step and calculate current values for E_{req}, E_{design}, and E_{code}. These values are then compared to averages of past projects. If the current results vary more than 20% from average, there may be cause for concern and there is certainly cause for investigation.

Example
- E_{req} for the current project = 2.1
- Organizational average = 3.6

– Two possibilities
- The team has done an outstanding job
- The team has been lax in its review approach

– If the second scenario appears likely
- Build additional design time

This can also be used to better target review and/or testing resources in the following manner:
– 120 components
– 32 exhibit E_{design} > 1.2 average
– Adjust code review resources accordingly

Time Boxing
Time-boxing is used in severe deadline pressure. It is a use incremental strategy where tasks associated with each increment are time-boxed in the following manner:

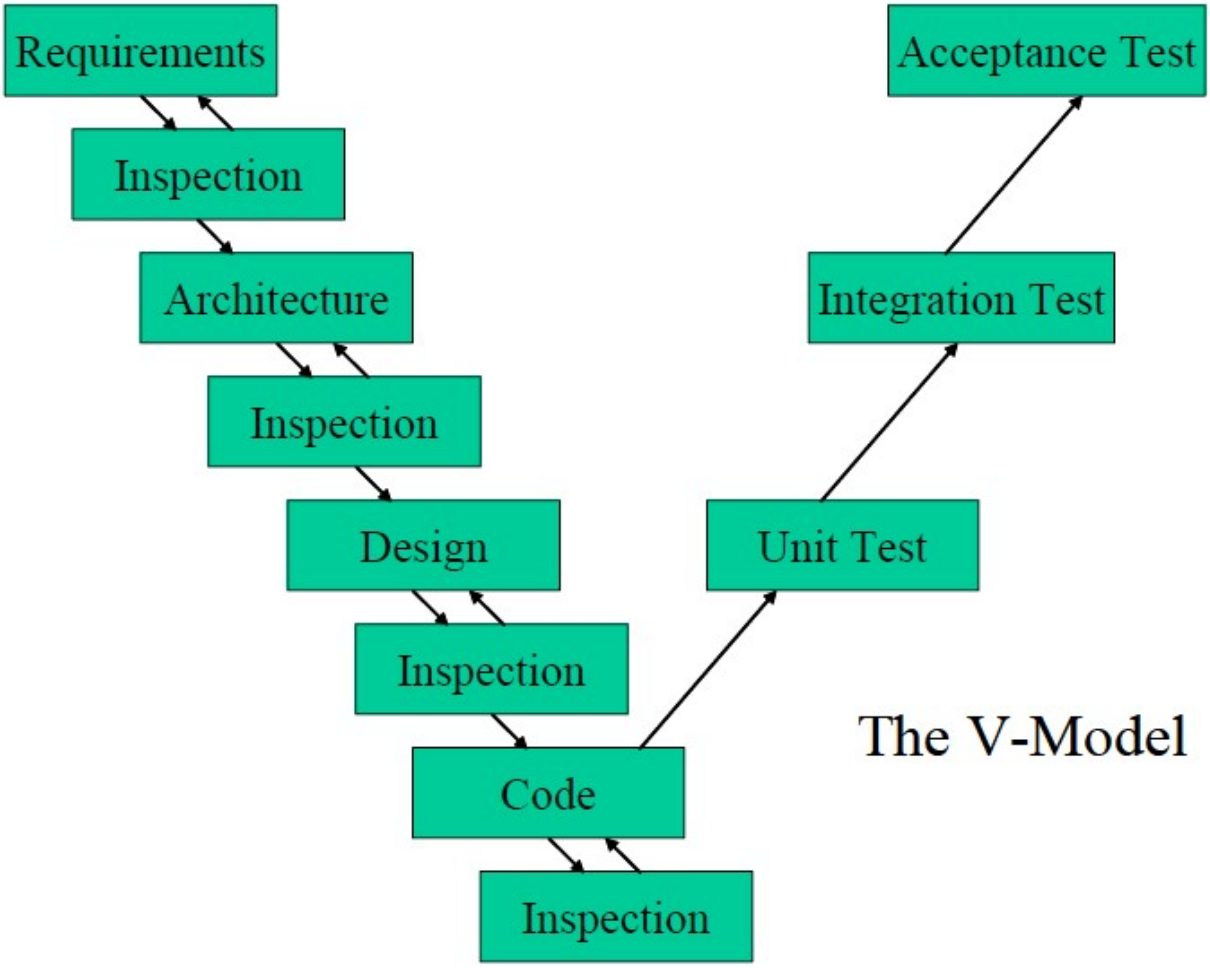
- Schedule for each task is adjusted by working backward from the delivery date.
- A box is put around each task
- When a task hits the boundary of the box, work stops and next task begins

The principle behind time-boxing is the 90-10 rule (similar to Pareto Principle) – rather than becoming stuck on the 10% of a task, the product proceeds towards the delivery date in 90% of the cases.

Lecture 3: Types of Review: Inspections, Desk checks, Walkthroughs, Code Reviews, Pair Programming

Software Reviews

Software reviews are the filter for the software engineering process. They are applied at various different points and serve to uncover errors that can be removed and help to purify the software engineering activities.
In this context it is useful to look at the "V-model" of software development. This model emphasizes that SQA is a function performed at all stages of software development life cycle. At the initial stages (requirement, architecture, design, code), it is achieved through activities known as Formal Technical Reviews or FTR. At the later stages (integration and acceptance), testing comes into picture.



Importance of reviews

Technical work needs reviewing for the same reason that pencils needs erasers: To errors is human. The second reason that we need technical reviews is although that people are good at catching errors, large class of errors escape the originator more easily than they escape anyone else.
Freedman defines a review – any review – as a way of using the diversity of a group of people to:

- Point out needed improvements in the product of a single person or team
- Confirm those parts of a product in which improvement is either not desired or no needed
- Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

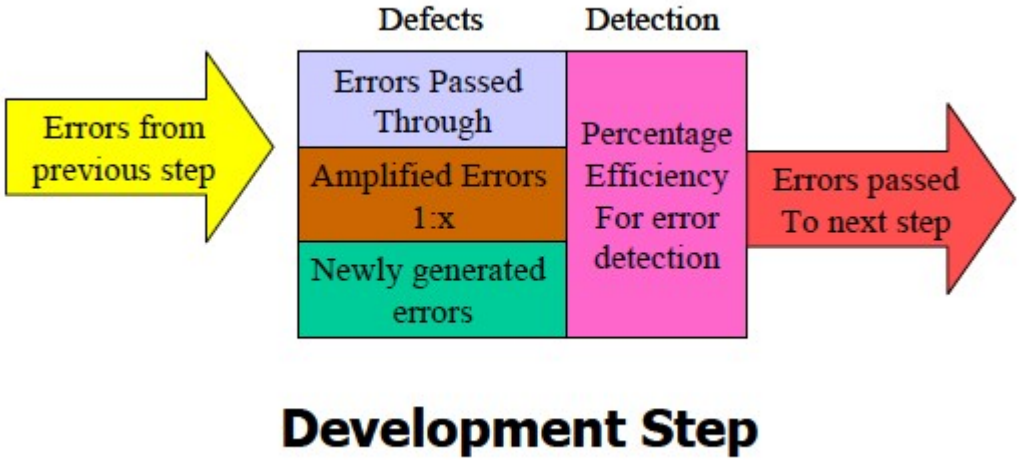
Reviews help the development team in improving the defect removal efficiency and hence play an important role in the development of a high-quality product.

Types of Reviews

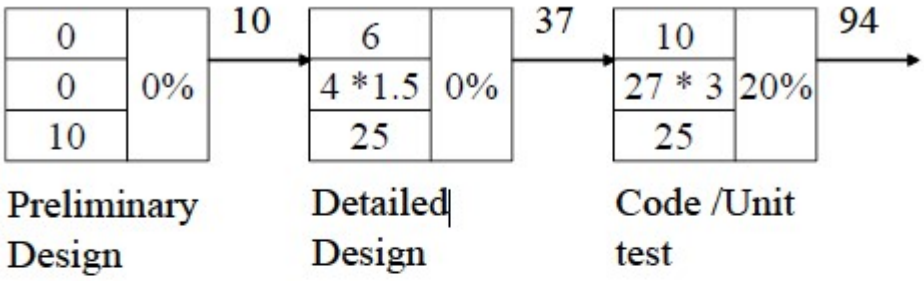
Inspections, Desk checks, Walkthroughs, Code Reviews, Pair Programming

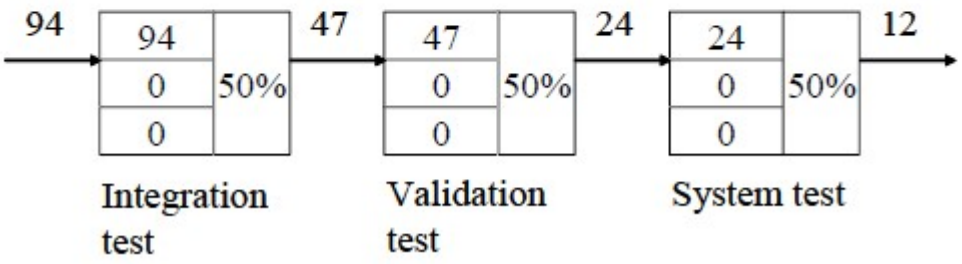
There are many types of reviews. In general they can be categorized into two main categories namely **informal and formal technical reviews**.
Formal Technical reviews are sometimes called as walkthroughs or inspections. They are the most effective filter from QA standpoint. To understand the significance of these reviews, let us look at the defect amplification model shown below.

This model depicts that each development step inherits certain errors from the previous step. Some of these errors are just passed through to the next step while some are worked on and hence are amplified with a ratio of 1:x. In addition, each step may also generate some new errors. If each step has some mechanism for error detection, some of these errors may be detected and removed and the rest are passed on to the next step.

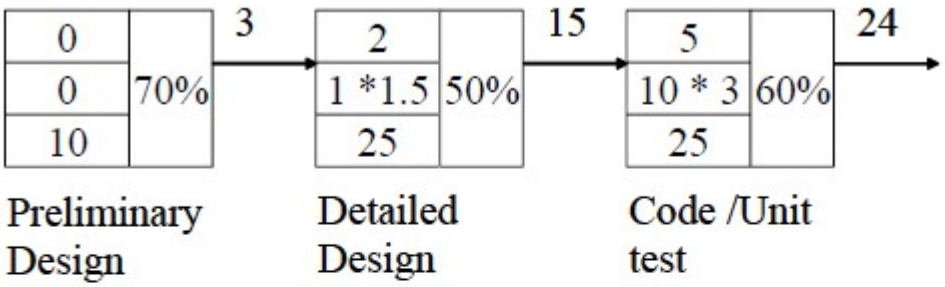


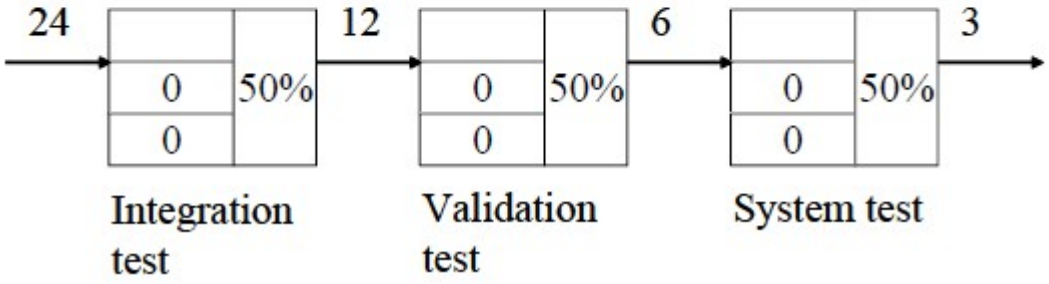
Let us now assume that we do not have any SQA related activities for the first two stages and we are only using testing for detection of any defects. Let us assume that the Preliminary design generated 10 defects which were passed on to detailed design. At that phase, 6 defects were passed on to the next stages and 4 were amplified at a ration of 1:1.5. In addition, there were 25 new defects introduced at this stage. Therefore, a total of 37 defects were passed on to the next stage as shown in the diagram. In the Code and Unite test stage, we start to test our system and assuming 20% defect removal efficiency of this stage, 94 defects (80% of (10 + 27 * 3 + 25)) are passed on to the next stage. This process continues and the system is delivered with 12 defects remaining in the product.





If FTR are used in the earlier stages, the quality of the end-product is much better as shown in the following diagram. Note that in this case we have code inspection in addition to unit testing at the third stage and the defect removal efficiency of that stage is 60%.





Formal Technical Reviews

Formal Technical Reviews are conducted by software engineers. The primary objective is to find errors during the process so that they do not become defects after release of software as they uncover errors in function, logic design, or implementation. The idea is to have early discovery of errors so they do not propagate to the next step in the process. They also ensure that the software has been represented according to predefined standards and it is developed in a uniform manner. They make projects more manageable and help groom new resources as well as provide backup and continuity. FTRs include walkthroughs, inspections, and other small group technical assessments of software.

1.Walkthrough

- A walkthrough is characterized by the author of the document under review **guiding the participants through the document and his or her thought processes**, to achieve a common understanding and to gather feedback.
- This is especially useful if people from outside the software discipline are present, who are not used to, or cannot easily understand software development documents.
- The content of the document is explained step by step by the author, to reach consensus on changes or to gather information.
- Within a walkthrough the author does most of the preparation.
- The participants, who are selected from different departments and backgrounds, are not required to do a detailed study of the documents in advance.
- Because of the way the meeting is structured, a large number of people can participate and this larger audience can bring a great number of diverse viewpoints regarding the contents of the document being reviewed as well as serving an educational purpose.
- If the audience represents a broad cross-section of skills and disciplines, it can give assurance that no major defects are 'missed' in the walk-through.
- A walkthrough is especially useful for higher-level documents, such as requirement specifications and architectural documents.
- The specific **goals of a walkthrough** depend on its role in the creation of the document. In general the following goals can be applicable:
    - to present the document to stakeholders both within and outside the soft ware discipline, in order to gather information regarding the topic under documentation;
    - to explain (knowledge transfer) and evaluate the contents of the document;
    - to establish a common understanding of the document;
    - to examine and discuss the validity of proposed solutions and the viability of alternatives, establishing consensus.
- **Key characteristics of walkthroughs** are:
    - The meeting is led by the authors; often a separate scribe is present.
    - Scenarios and dry runs may be used to validate the content.
    - Separate pre-meeting preparation for reviewers is optional.

Guidelines for walkthroughs

FTRs are usually conducted in a meeting that is successful only if it is properly planned, controlled, and attended. The producer informs the PM that the WP is ready and the review is needed. The review meeting consists of 3-5 people and advanced preparation is required. It is important that this preparation should not require more than 2 hours of work per person. It should focus on specific (and small) part of the overall software. For example, instead of the entire design, walkthroughs are

conducted for each component, or small group of components. By narrowing focus, FTR has a high probability of uncovering errors.
It is important to remember that the focus is on a work product for which the producer of the WP asks the project leader for review. Project leader informs the review leader. The review leader evaluates the WP for readiness and if satisfied generates copies of review material and distributes to reviewers for advanced preparation. The agenda is also prepared by the review leader.

Review Meetings

Review meeting is attended by the review leader, all reviewers, and the producer. One of the reviewers takes the roles of recorder. Producer walks through the product, explaining the material while other reviewers raise issues based upon their advanced preparation. When valid problems or errors are recorded, the recorder notes each one of them. At the end of the RM, all attendees of the meeting must decide whether to:

- Accept the product without further modification
- Reject the product due to severe errors
  - Major errors identified
  - Must review again after fixing
- Accept the product provisionally
  - Minor errors to be fixed
  - No further review

Review Reporting and Record keeping

During the FTR the recorder notes all the issues. They are summarized at the end and a review issue list is prepared. A summary report is produced that includes:

- What is reviewed?
- Who reviewed it
- What were the findings and conclusions?

It then becomes part of project historical record.

The review issue list

It is sometimes very useful to have a proper review issue list. It has two objectives.

- Identify problem areas within the WP
- Action item checklist

It is important to establish a follow-up procedure to ensure that items on the issue list have been properly addressed.

Review Guidelines

It is essential to note that an uncontrolled review can be worse than no review. The basis principle is that the review should focus on the product and not the producer so that it does not become personal.

Remember to be sensitive to personal egos. Errors should be pointed out gently and the tone should be loose and constructive.
This can be achieved by setting an agenda and maintaining it. In order to do so, the review team should:
 Avoid drift
- Limit debate and rebuttal
- Enunciate problem areas but don't try to solve all problems
- Take written notes
- Limit the number of participants and insist upon advanced preparation
- Develop a checklist for each product that is likely to be reviewed
- Allocate resources and schedule time for FTRs
- Conduct meaningful training for all reviewers
- Review your early reviews
- Determine what approach works best for you

2.Pair-Programming

Most people associate pair-programming with XP5 and agile development in general, but it's also a development process that incorporates continuous code review. Pair-programming is two developers writing code at a single workstation with only one developer typing at a time and continuous free-form discussion and review.
Studies of pair-programming have shown it to be very effective at both finding bugs and promoting knowledge transfer. And some developers really enjoy doing it. There's a controversial issue about whether pair-programming reviews are better, worse, or complementary to more standard reviews. The reviewing developer is deeply involved in the code, giving great thought to the issues and consequences arising from different implementations. On the one hand this gives the reviewer lots of inspection time and a deep insight into the problem at hand, so perhaps this means the review is more effective. On the other hand, this closeness is exactly what you don't want in a reviewer; just as no author can see all typos in his own writing, a reviewer too close to the code cannot step back and critique it from a fresh and unbiased position. Some people suggest using both techniques – pair-programming for the deep
review and a follow-up standard review for fresh eyes. Although this takes a lot of developer time to implement, it would seem that this technique would find the greatest number of defects. We've never seen anyone do this in practice. The single biggest complaint about pair-programming is that it takes too much time. Rather than having a reviewer spend 15-30 minutes reviewing a change that took one developer a few days to make, in pair-programming you have two developers on the task the entire time. Some developers just don't like pair-programming; it depends on the disposition of the developers and who is partnered with whom. Pair-programming also does not address the issue of remote developers.

A full discussion of the pros and cons of pair-programming in general is beyond our scope.

3.Inspection

- Inspection is the **most formal review type**.

- The document under inspection is prepared and checked thoroughly by the reviewers before the meeting, comparing the work product with its sources and other referenced documents, and using rules and checklists.
- In the inspection meeting the defects found are logged and any discussion is postponed until the discussion phase. This makes the inspection meeting a very efficient meeting.
- Many engineering organizations have established independent test groups that specialize in finding defects.
- Similar principles have led to the introduction of inspections and reviews in general.
- Depending on the organization and the objectives of a project, inspections can be balanced to serve a number of goals.

The generally accepted **goals of inspection** are to:

  - help the author to improve the quality of the document under inspection
  - remove defects efficiently, as early as possible
  - improve product quality, by producing documents with a higher level of quality
  - create a common understanding by exchanging information among the inspection participants
  - train new employees in the organization's development process
  - learn from defects found and improve processes in order to prevent recurrence of similar defects
  - Sample a few pages or sections from a larger document in order to measure the typical quality of the document, leading to improved work by individuals in the future, and to process improvements.
- Key **characteristics of an inspection** are:
  - It is usually led by a trained moderator (certainly not by the author).
  - It uses defined roles during the process.
  - It involves peers to examine the product.
  - Rules and checklists are used during the preparation phase.
  - A separate preparation is carried out during which the product is examined and the defects are found.
  - The defects found are documented in a logging list or issue log.
  - A formal follow-up is carried out by the moderator applying exit criteria.
  - Optionally, a causal analysis step is introduced to address process improve ment issues and learn from the defects found.
  - Metrics are gathered and analyzed to optimize the process.

Formal inspections

For historical reasons, “formal” reviews are usually called “inspections.” This is a hold-over from Michael Fagan’s seminal 1976 study at IBM regarding the efficacy of peer reviews. He tried many combinations of variables and came up with a procedure for reviewing up to 250 lines of prose or source code. After 800 iterations he came up with a formalized inspection strategy and whom to this day you can pay to tell you about it (company name: Fagan Associates). His methods were further studied and ex-pended upon by others, most nota ably Tom Gilb and Karl Wiegers. In general, a “formal” review refers to a heavy-process review with three to six participants meeting together in one room with print-outs

and/or a projector. Someone is the “moderator” or “controller” and acts as the organizer, keeps everyone on task, controls the pace of the review, and acts as arbiter of disputes. Everyone reads through the materials beforehand to properly prepare for the meeting. Each participant will be assigned a specific “role.” A “re-viewer” might be tasked with critical analysis while an “observer” might be called in for domain-specific advice or to learn how to do reviews properly. In a Fagan Inspection, a “reader” looks at source code only for comprehension – not for critique – and presents this to the group. This separates what the author in-tended from what is actually presented; often the author himself is able to pick out defects given this third-party description. When defects are discovered in a formal review, they are usually recorded in great detail. Besides the general location of the error in the code, they include details such as severity (e.g. major, minor), type (e.g. algorithm, documentation, data-usage, error-handling), and phase-injection (e.g. developer error, design oversight, requirements mistake). Typically this information is kept in a database so defect metrics can be analyzed from many angles and possibly compared to similar metrics from QA.

Formal inspections also typically record other metrics such as individual time spent during pre-meeting reading and during the meeting itself, lines-of-code inspection rates, and problems encountered with the process it self. These numbers and comments are examined periodically in process-improvement meetings; Fagan Inspections go one step further and requires a process-rating questionnaire after each meeting to help with the improvement step.

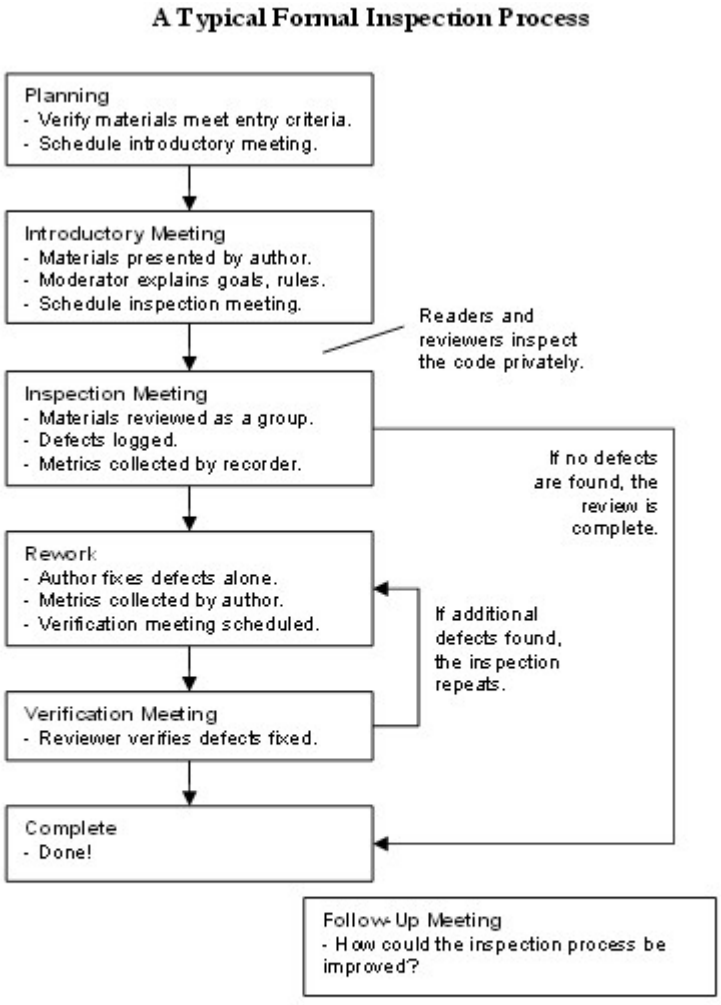


Figure 1: Typical workflow for a "formal" inspection. Not shown are the artifacts created by the review: The defect log, meeting notes, and metrics log. Some inspections also have a closing questionnaire used in the follow-up meeting.

Formal inspections' greatest asset is also its biggest drawback: When you have many people spending lots of time reading code and discussing its consequences, you are going to identify a lot of defects. And there are plenty of studies that show formal inspections can identify a large number of problems in source code. But most organizations cannot afford to tie up that many people for that long. You also have to schedule the meetings – a daunting task in it self and one that ends up consuming extra developer time1. Finally, most formal methods require training to be effective, and this is an additional time and expense that is difficult to accept, especially when you aren't already used to doing code reviews. Many studies in the past 15 years have come out demonstrating that other forms of review uncover just as many defects as do formal reviews but with much less time and training2. This result –anticipated by those who have tried many types of review – has put formal inspections out of favor in the industry. After all, if you can get all the proven benefits of formal inspections but occupy 1/3 the developer time, that's clearly better.

4. Desk Checks

A desk check is the rest line of defense against defects. You can speed up formal inspections by taking care of simple defects in desk checks rest. For many work products, desk checks sauce, and you might not need to go to a formal inspection. However, desk checks are only e active if taken seriously. It's easy to just say \LGTM" (Looks Good to Me) without actually checking the product. It's important to spend enough time on desk checks, and managers must allocate time for them

5. Code Reviews

1. A *code review* is a special kind of inspection in which the team examines a sample of code and fixes any defects in it.
   - ▷ In a code review, a defect is a block of code which does not properly implement its requirements, which does not function as the programmer intended, or which is not incorrect but could be improved
     - For example, it could be made more readable or its performance could be improved
2. It's important to review the code which is most likely to have defects. This will generally be the most complex, tricky or involved code.
   - ▷ Good candidates for code review include:
     - A portion of the software that only one person has the expertise to maintain
     - Code that implements a highly abstract or tricky algorithm
     - An object, library or API that is particularly difficult to work with
     - Code written by someone who is inexperienced or has not written that kind of code before, or written in an unfamiliar language
   - Code which employs a new programming technique
   - An area of the code that will be especially catastrophic if there are defects

Code Review Checklist

- ▷ Clarity
  - Is the code clear and easy to understand?
  - Did the programmer unnecessarily obfuscate any part of it?
  - Can the code be re-factored to make it clearer?
- ▷ Maintainability
  - Will other programmers be able to maintain this code?
  - Is it well commented and documented properly?
- ▷ Accuracy
  - Does the code accomplish what it is meant to do?
  - If an algorithm is being implemented, is it implemented correctly?
- ▷ Readability and Robustness
  - Is the code fault-tolerant? Is the code error-tolerant?
  - Will it handle abnormal conditions or malformed input?
  - Does it fail gracefully if it encounters an unexpended condition?
- ▷ Security
  - Is the code vulnerable to unauthorized access, malicious use, or modification?
- ▷ Scalability
  - Could the code be a bottleneck that prevents the system from growing to accommodate increase load, data, users, or input?

- Reusability
  - Could this code be reused in other applications?
  - Can it be made more general?
- Efficiency
  - Does the code make efficient use if memory, CPU cycles, bandwidth, or other system resources?
  - Can it be optimized?