Software Project Management

Durga Prasad Mohapatra
Professor
CSE Deptt.
NIT Rourkela

Software quality cont…

ISO/IEC 15504 IT process assessment

- ISO/IEC 15504 is a standard for process assessment that shares many concepts with CMMI. The two standards (i.e. ISO/IEC 15504 and CMMI) should be compatible.
- It is designed to provide guidance on the assessment of software development processes.
- To do this, there must be some benchmark or process reference model which represents the ideal development life cycle against which the actual processes can be compared.

Process Reference Model

- A defined standard approach to development.
- Needs a defined set of processes that represent good practice to be the benchmark, against which the processes to be assessed can be judged.
- **ISO 12207** is the default reference model.
- Could use other reference models in specific environments.
- The processes are assessed on the basis of 9 process attributes, as shown in next table.

ISO 15504 performance attributes

CMMI level	ISO 15504
	0. incomplete
initial	1.1.process performance – achieves defined outcome
repeatable	2.1 process management – it is planned and monitored
	2.2 work product management – control of work products

ISO 15504 performance attributes – cont…

CMMI	ISO 15504
Defined	3.1. Process definition
	3.2. Process deployment
Managed	4.1. Process measurement
	4.2. Process control
Optimizing	5.1. Process innovation
	5.2. Process optimization

ISO 15504 Process Assessment

For each process in the relevant Process Reference Model

For each set of attribute level criteria

Assess whether:

N: not achieved	0-15%
P: partially achieved	15%-50%
L: largely achieved	50%-85%
F: fully achieved	85% or more

ISO 15504 performance indicators

This is just an example of how indicators for each level *might* be identified

1. Performance

Descriptions of maximum and minimum expected input values exist

2.1 Performance management

A plan of how expected input variable ranges are to be obtained which is up to date

ISO 15504 performance indicators 2

2.2 Work product management

There are minutes of a meeting where the input requirements document was reviewed and corrections were mandated

3.1 Process definition

A written procedure for input requirements gathering exists

3.2 Process deployment

A control document exists that is signed as each part of the procedure is completed

ISO 15504 performance indicators 2

4.1. Process measurement

Measurement data can be collected e.g. number of changes resulting from review

4.2. Process control

Memos relating to management actions taken in the light of the above

ISO 15504 performance indicators 3

5.1 Process innovation

Existence of some kind of 'lessons learnt' report at the end of project

5.2. Process optimization

Existence of documents assessing the feasibility of suggested process improvements and which show consultation with relevant stakeholders

Quality Systems for Small Organizations

- Small organizations tend to believe:
  - We are all competent people hired to do a job, we can't afford training.
  - We all communicate with one another.
    - Osmosis works because we are so close.
  - We are all heroes:
    - We do what needs to be done.
    - Therefore rules do not apply to us.

Quality Systems for Small Organizations

- Often have problems:
  - Undocumented requirements
  - Inexperienced managers
  - Documenting the product
  - Resource allocation
  - Training
  - Peer reviews

Quality Systems for Small Organizations

- A two week CMM-based appraisal is probably excessive:
- Small organizations need to operate more efficiently at lower levels of maturity
  - Must first flourish if eventually they are to mature

Personal Software Process (PSP)

- Based on the work of Humphrey.
- PSP is a scaled down version of industrial software process:
  - Suitable for individual use.
- Even CMM assumes that engineers use effective personal practices.

Personal Software Process (PSP)

- A process is the set of steps for doing a job.
- The quality and productivity of an engineer are
  - Largely determined by his process
- PSP framework:
  - Helps software engineers to measure and improve the way they work.

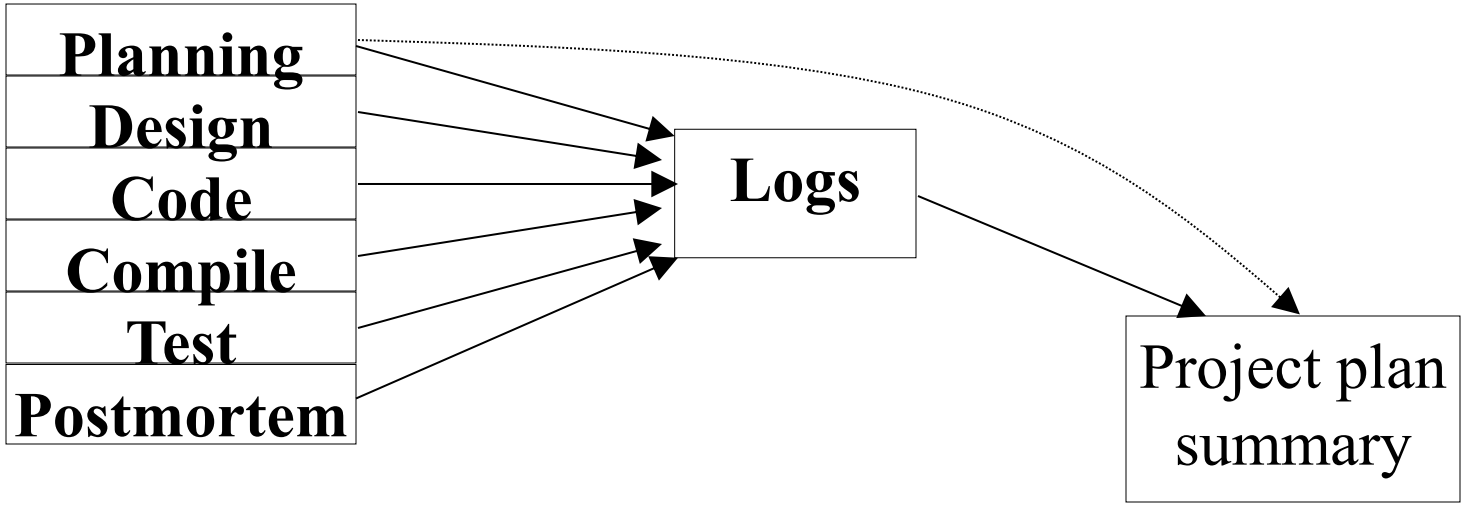
Personal Software Process (PSP)

- Helps developing personal skills and methods.
  - Estimating and planning method.
  - Shows how to track performance against plans.
  - Provides a defined process;
    - Can be fine tuned by individuals.
    - Recognizes that a process for individual use is different from that necessary for a team project.

Time Management

- Track the way you spend time:
  - Boring activities seem longer than actual.
  - Interesting activities seem short.
- Record time for:
  - Designing
  - Writing code
  - Compiling
  - Testing

Personal Software Process (PSP)



PSP-Planning

- Problem definition
- Estimate max, min, and total LOC
- Determine minutes/LOC
- Calculate max, min, and total development times
- Enter the plan data in project plan summary form
- Record the planned time in Log

PSP-Design

- Design the program.
- Record the design in specified format.
- Record the Design time in time recording log.

PSP-Code

- Implement the design.
- Use a standard format for code text.
- Use coding standards and guidelines.
- Record the coding time in time recording log.

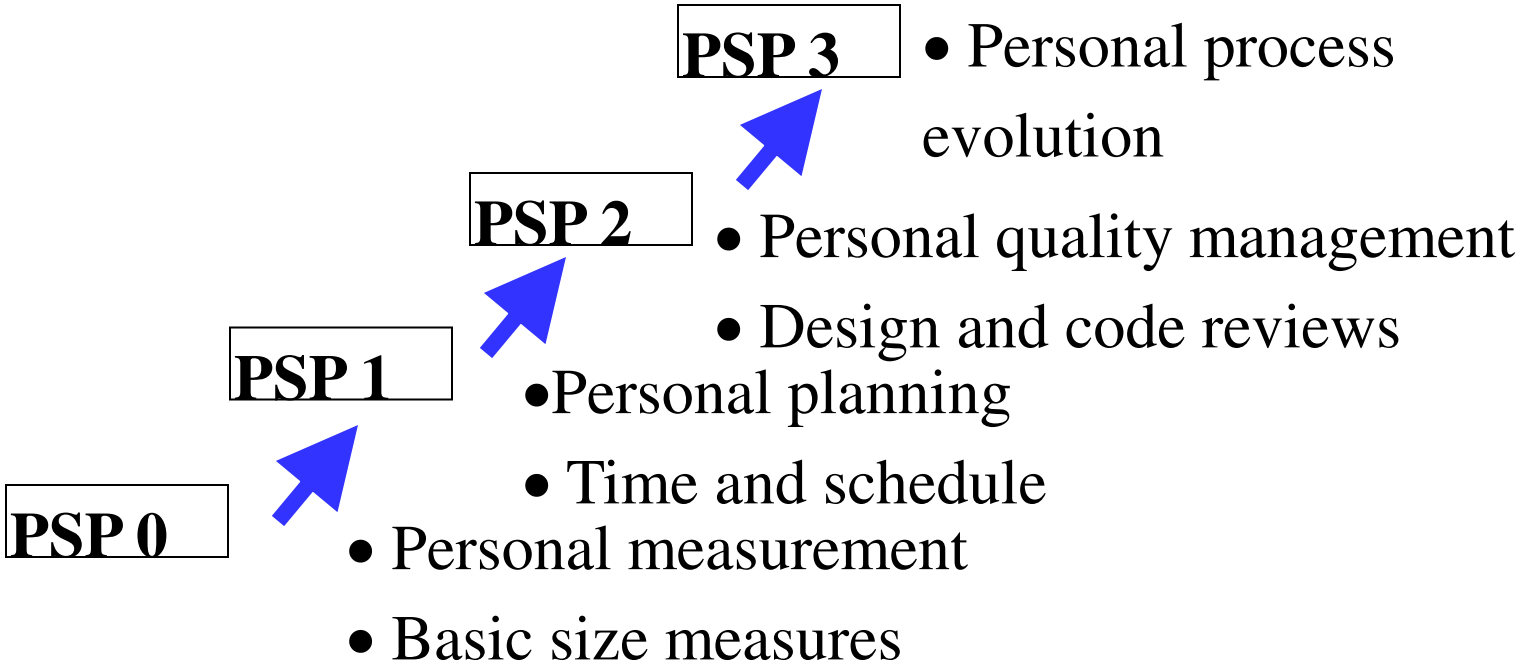
PSP-Compile

- Compile the program.
- Fix all the defects.
- Record compile time in time recording log.

PSP-Test/Postmortem

- Test:
  - Test the program.
  - Fix all the defects found.
  - Record testing time in time recording log.
- Postmortem:
  - Complete project plan summary form with actual time and size data.
  - Record postmortem time in time record.

Personal Software Process (PSP)



Six Sigma

- Six sigma is a quantitative approach to eliminate defects:
  - Applicable to all types of industry - from manufacturing, product development, to service.
- The statistical representation of Six Sigma quantitatively describes :
  - How a process is performing?

Six Sigma

- To achieve six sigma:
  - A process must not produce more than 3.4 defects per million opportunities.
  - 5 Sigma -> 230 defects per million.
  - 4 Sigma -> 6210 defects per million.
- Six sigma methodologies:
  - DMAIC (Define, Measure, Analyse, Improve, Control).
  - DMADV: (Define, Measure, Analyse, Design, Verify).

Six Sigma Methodologies

- The methodologies are implemented by Green belt and Black belt workers:
  - Supervised by Master black belt worker.
- Pareto Chart:
  - Simple bar chart to represent defect data
  - Emphasis on identifying the problems that occur with greatest frequency
    - or incur the highest cost

Techniques to improve quality

- Inspection
- Walkthrough
- Review

Already discussed

- Clean room software development
- Formal methods
- Testing
- Reliability

‘Clean-room’ software development

- Pioneered at IBM
- The term cleanroom was first coined at IBM by drawing analogy to the semi-conductor fabrication units where the defects are avoided by manufacturing in an ultra-clean atmosphere.
- Relies heavily on walkthroughs, inspection and formal verification for bug removal
- Programmers are not allowed to test any of their code by executing the code other than doing some syntax testing using a compiler

‘Clean-room’ software development

Ideas associated with Harlan Mills at IBM

- Three separate teams:
  1. Specification team – documents user requirements and usage profiles (how much use each function will have)
  2. Development team – develops code but does not test it. Uses mathematical verification techniques
  3. Certification team – tests code. Statistical model used to decide when to stop

Formal methods

- Mathematical notations such as VDM (Vienna Development Method) and Z can be used to produce unambiguous specifications.
- Can prove correctness of software mathematically (cf. geometric proofs of Pythagoras' theorem).
- Newer approaches use Object Constraint Language (OCL) to add details to UML models.
- Aspiration is to be able to generate applications directly from UML+OCL without manual coding – Model Driven Architectures (MDA).

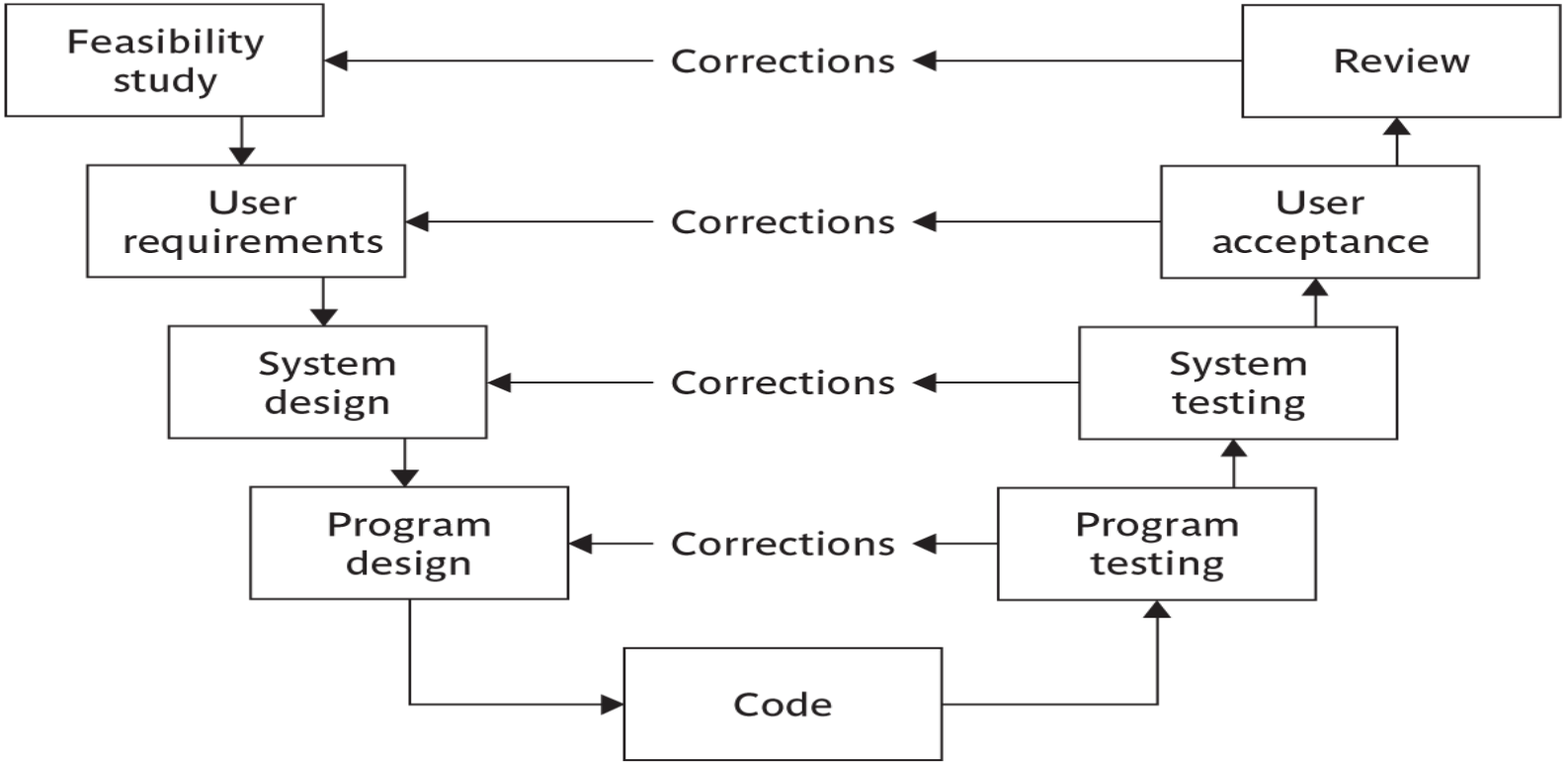
Verification versus Validation

- Verification is the process of determining whether the output of one phase of software development conforms to that of its previous phase;
  - whereas validation is the process of determining whether a fully developed software conforms to its requirements specification.
- Verification is carried out during the development process to check if the development activities are being carried out correctly,
  - whereas validation is carried out towards the end of the development process to check if the right product as required by the customer has been developed.

Testing: the V-process model

- This is shown diagrammatically on the next slide
- It is an extension of the waterfall approach
- For each development stage there is a testing stage
- The testing associated with different stages serves different purposes
  - e.g. system testing tests that components work together correctly, user acceptance testing tests that users can use system to carry out their work

Testing: the V-process model



Black box versus glass box testing

- Glass box testing
  - The tester is aware of the internal structure of the code; can test each path; can assess percentage test coverage of the tests e.g. proportion of code that has been executed
- Black box testing
  - The tester is not aware of internal structure; concerned with degree to which it meets user requirements

Levels of testing

- Unit testing
- Integration testing
- System testing

Testing activities

- *Test planning*
- *Test suite design*
- *Test case execution and result checking*
- *Test reporting:*
- *Debugging:*
- *Error correction:*
- *Defect retesting*
- *Regression testing*
- *Test closure:*

Test plans

- Specify test environment
  - In many cases, especially with software that controls equipment, a special test system will need to be set up
- Usage profile
  - failures in operational system more likely in the more heavily used components
  - Faults in less used parts can lie hidden for a long time
  - Testing heavily used components more thoroughly tends to reduce number of operational failures

Management of testing

The tester executes test cases and may as a result find discrepancies between actual results and expected results – **issues**

Issue resolution – could be:

- a mistake by tester
- a fault – needs correction
- a fault – may decide not to correct: **off-specification**
- a change – software works as specified, but specification wrong: submit to change control

Decision to stop testing

- The problem: impossible to know there are no more errors in code
- Need to estimate how many errors are likely to be left
- **Bug seeding –** insert (or leave) known bugs in code
- Estimate of bugs left =
  - (total errors found)/(seeded errors found) x (total seeded errors)

How many errors are still remaining?

- Seed the code with some known errors:
  - artificial errors are introduced into the program.
  - Check how many of the seeded errors are detected during testing.

Error Seeding

Let:

- N be the total number of errors in the system
- n of these errors be found by testing.
- S be the total number of seeded errors,
- s of the seeded errors be found during testing.

Error Seeding

$n/N = s/S$

$N = S * n/s$

remaining defects:

$$N - n = n * ((S - s)/ s)$$

Example

- 100 errors were introduced.
- 90 of these errors were found during testing
- 50 other errors were also found.
- Remaining errors= 50 * (100-90)/90 = 6

Error Seeding - issues

- The kind of seeded errors should match closely with existing errors:
  - However, it is difficult to predict the types of errors that exist.
- Categories of remaining errors:
  - can be estimated by analyzing historical data from similar projects.

Alternative method of error estimation

- Have two independent testers, A and B
- N_1 = valid errors found by A
- N_2 = valid errors found by B
- N_{12} = number of cases where same error found by A and B
- Estimate = $(N_1 \times N_2)/ N_{12}$
- Example: A finds 30 errors, B finds 20 errors. 15 are common to A and B. How many errors are there likely to be?
- (30 X 20)/15 = 40 errors

Test automation

- Other than reducing human effort and time,
  - Test automation also significantly improves the thoroughness of testing.
- A large number of tools are at present available both in the public domain as well as from commercial sources.

Types of Testing Tools

- *Capture and playback*
- *Automated test script*
- *Random input test*
- *Model-based test*

Summary

- Discussed ISO/IEC 15504 Standard
- Discussed PSP.
- Briefly introduced Six sigma.
- Explained some techniques to improve software quality.
  - Clean room software development
  - Formal methods
  - Testing

References

1. B. Hughes, M. Cotterell, R. Mall, *Software Project Management*, Sixth Edition, McGraw Hill Education (India) Pvt. Ltd., 2018.
2. R. Mall, *Fundamentals of Software Engineering*, Fifth Edition, PHI Learning Pvt. Ltd., 2018.

Thank you