What Makes a Good Project Manager?

Tuesday, July 12, 2022 11:01 AM

Problem: Start → ☁ → Finish

Core Skills

Flexible

- Compassion
- Lead vs. Manage
- Meetings are communication

% of projects:

31 - Cancelled

53 - Finished late, over budget, not all functionality

16 - Completed

Techniques

- CMM - Capability Maturity Model
- PSP - Personal Software Process

Common Practices

Reviews, inspections, and walkthroughs
Metrics (measurement data)
Quality gates (binary quality decision gates)
Milestones (requirements, specifications, design, code, test, manuals)
Visibility of plans and progress
Defect tracking
Clear management accountability
Technical performance related to value for the business
Testing early and often
Fewer, better people (project managers and technical people)
Use of specialists
Opposition of featuritis and creeping requirements
Documentation for everything
Design before implementing
Planning (and use of planning tools)
Cost estimation (using tools, realistic vice optimistic)
Quality control
Change management
Reusable items
Project tracking
Users—understand them
Buy in and ownership of the project by all participants
Executive sponsor
Requirements
Risk management
User manuals (as system specifications)

Misc Tips

- Avoid members working in isolation
- Stay with the team
- Concentrate on tasks, not tools
- Do your homework (Read and evaluate)

4 Project Managing Basics

Wednesday, July 13, 2022 9:01 AM

3 - P's to Balance

People

- Project feasibility
- Risk management
- Team structure
- Project schedule
- Project understandability
- Sense of accomplishment

Process

- Standardize training
- Repeatable
- Allows for cost estimation and scheduling
- Project understanding
- People integration is made easier
- Communication

Product

- Artifacts (metrics, information)
- Software system

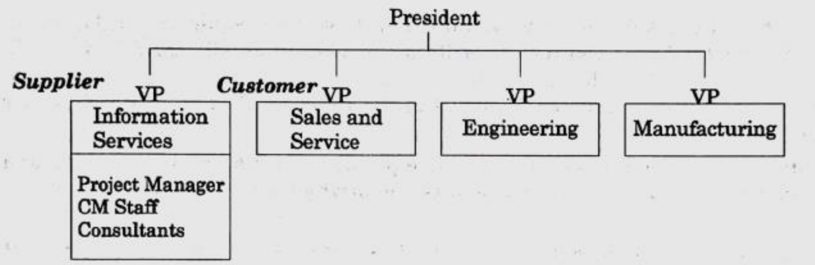
Visibility

Configuration Management

Baselines

Activities

- Identification (naming convention)
- Version Control
- Audits (Physical hardware inventory)
- Status accounting



Baseline	Contents	Configuration Control Board
Business Needs	What the sales and service department needs. What the other departments need to interface.	President, all VP's, PM, user groups from Sales and Service
Allocated	General design, technical capabilities, no name brands.	Sales and Service VP, PM, Systems Consultants, IS Department
Design	Detailed design, brand names, model numbers, custom software design.	PM, H/W Consultant, S/W Consultant, IS Department
Development	Systems being assembled, code being written.	PM, H/W Consultant, S/W Consultant, IS Department
Test	System being tested by "outsiders"	PM, Systems Consultant, Testers
Demonstration	A working system in a staged office.	PM, Systems Consultants, IS Department, Users
Operational	What the company is using today.	President, all VP's, PM, user groups from Sales and Service

Figure 2.5 Sample CM sketch.

Use standards

Managing a Project Day by Day

Thursday, July 14, 2022 8:59 AM

Similar to cultivating a garden. You cannot force plants to grow

Balance of

- The Manager's Emotional safety
- Team empowerment
- Lots of Personal Interaction
- Let people succeed frequently
- Recognise and Deal with causes of failure
- Reinforce healthy work principles

Control = plan + status + corrective action

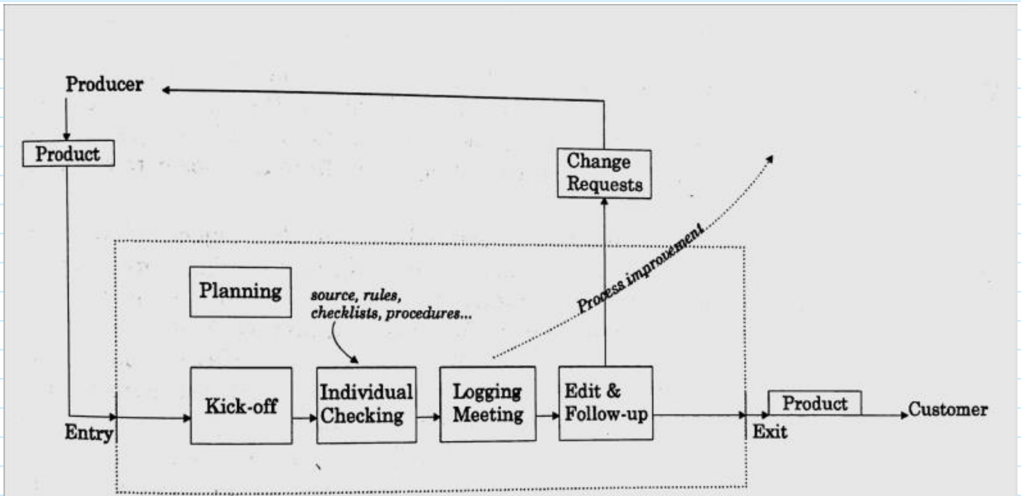


Figure 4.2 An inspection process.

Management Information center

Charts to display:

- Gantt
- PERT - Network chart
- Software size
- Cumulative cost
- Staffing status
- Earned value tracking (% of finished tasks in project)
- Requirements stability
- Slip = latest delivery date - previously announced date

Unit Development Folder (Physical)

Unit Development Folder (Physical)

Analyze Daily

- Make Decision table to evaluate critical points, and post in MIC

External supplies

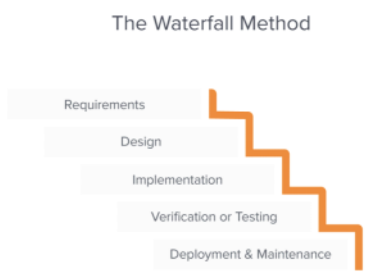
- Trust more, get status less
- Bring testers and users to meetings
- Use EVT's

Milestones promote visibility

Standards

The Waterfall Method

Monday, July 25, 2022 10:00 AM



Who uses the Waterfall model?

The Waterfall process is adopted by project managers who are faced with development projects that:

- Don't have ambiguous requirements.
- Offer a clear picture of how things will proceed from the outset.
- Have clients who seem unlikely to change the scope of the project once it is underway.

If a project manager prefers clearly defined processes, where cost, design, and time requirements are known upfront, then the Waterfall method is the way to go, as long as the project itself is conducive to those constraints.

Advantages

The Waterfall methodology is a straightforward, well-defined project management methodology with a proven track record. Since the requirements are clearly laid out from the beginning, each contributor knows what must be done when, and they can effectively plan their time for the duration of the project.

Other benefits of the Waterfall method include:

- Developers can catch design errors during the analysis and design stages, helping them to avoid writing faulty code during the implementation phase.
- The total cost of the project can be accurately estimated, as can the timeline, after the requirements have been defined.
- With the structured approach, it is easier to measure progress according to clearly defined milestones.
- Developers who join the project in progress can easily get up to speed because everything they need to know should be in the requirements document.
- Customers aren't always adding new requirements to the project, delaying production.

Weaknesses

Like any development process, the strengths in one area might mean weaknesses in the other. The Waterfall methodology's insistence on upfront project planning and commitment to a certain defined progress means that it is less flexible, or agile, later in the game. Changes that come further in the process can be time-consuming, painful, and costly.

Other reasons the Waterfall methodology may not work include:

- Projects can take longer to deliver with this chronological approach than with an iterative one, such as the Agile method.
- Clients often don't fully know what they want at the front end, opening the door to requests for changes and new features later in the process when they're harder to accommodate.
- Clients are not involved in the design and implementation stages.
- Deadline creep—when one phase in the process is delayed, all the other phases are delayed.

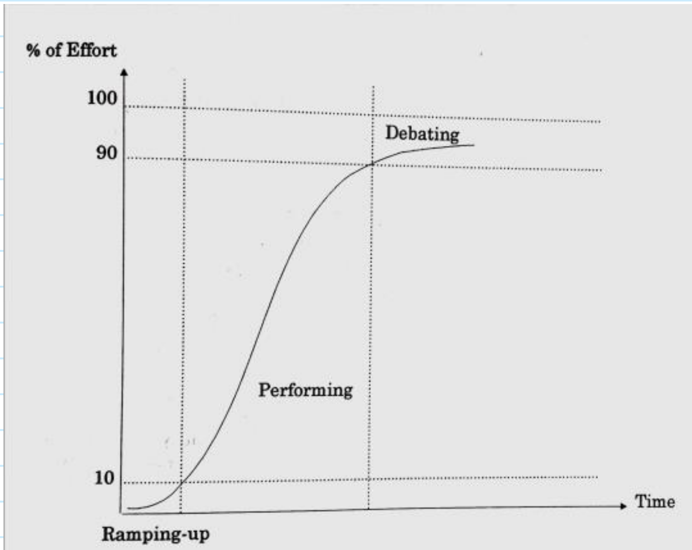
Requirements

Friday, July 15, 2022 10:58 AM

Visibility techniques

Joint Application Development workshop

Design by walking around



The 90/10 rule of work.

System storyboarding technique

Concept of operations (become the user)

Mind map

We want the system to be "easy to use"

Function: Time required for a user to learn to use the system given a tutorial manual.

Attribute	Scale	Test	Worst	Plan	Best	Now
Enter a patient record	Time (mins)	Give a user the manual and ask them to perform the attribute operation.	10	5	2	60
Create weekly report	Time (mins)	Give a user the manual and ask them to perform the attribute operation.	30	15	10	120
Perform system backup	Time (mins)	Give a user the manual and ask them to perform the attribute operation.	20	10	5	90

Figure 5.4 A Gilb chart to quantify "easy to use"

Rapid prototyping

Planning

Monday, July 18, 2022 10:08 AM

Task List (inputs, requirements)
Resources
Task network (task precedence)

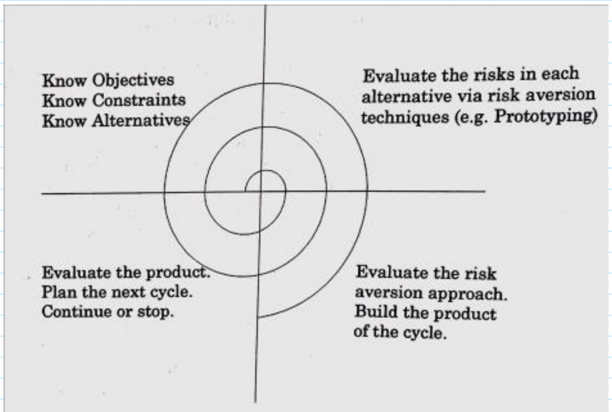
Software and hardware engineering management are the same

Rank user requirements

Microsoft process

- Long tail of the 80/20 rule
- Compile and test every day from day one
- Assign all new programmers to a mentor

Spiral



General spiral process.

Personal Software Process

- Record how long tasks take in minutes
- Track errors and fixes
- Metrics are crucial
- Lines of code to estimate size
- Check lists of risks and problems from past projects

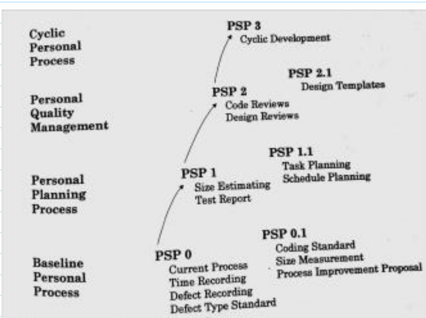
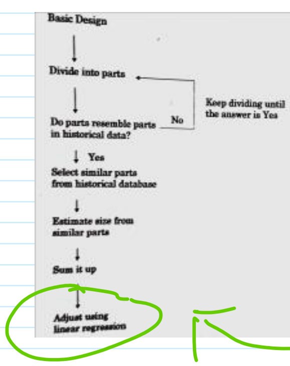


Figure 6.10 The Personal Software Process.



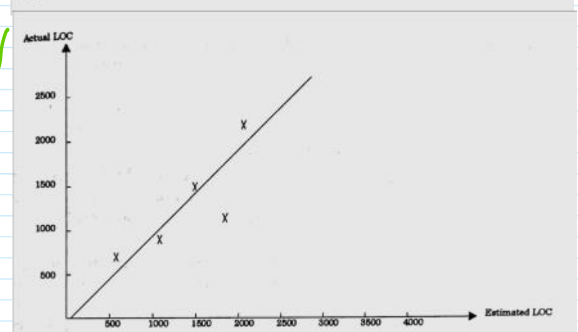
Final Estimated LOC = $\beta_0 + (\beta_1 * \text{Initial Estimated LOC})$

$$\beta_1 = \frac{\Sigma(\text{Estimated LOC}_i * \text{Actual LOC}_i) - n * \text{Estimated LOC}_{AVG} * \text{Actual LOC}_{AVG}}{\Sigma(\text{Estimated LOC}_i)^2 - n*(\text{Estimated LOC}_{AVG})^2}$$

for n prior projects

$$\beta_0 = \text{Actual LOC}_{AVG} - \beta_1 * \text{Estimated LOC}_{AVG}$$

Figure 6.29 Transforming the initial estimated size to the final estimated size using linear regression.



Data

Estimated	Actual
550	675
1100	795
1500	1450
1750	1200
2100	2315

$\beta_0 = 0.96$

Best Practices

Look at what the best people do and repeat it

Rayleigh Model

Look into this when needed...

Waterfall

Iterative

Evolutionary

Configuration Management

Evolutionary
Configuration Management

1100	795
1500	1450
1750	1200
2100	2315

$\beta_0 = 0.96$
$\beta_1 = -43$

Actual LOC = -43 + (0.95)*(Esimated LOC)
Calculated Values

Initial Estimate	Final Estimate
1000	917
2000	1877

5.30 A graph of linear regression for five projects.

Documenting the Plan

1. Introduction
   - 1.1 Project Overview
   - 1.2 Project Deliverables
   - 1.3 Evolution of the SPMP
   - 1.4 Reference Materials
   - 1.5 Definitions and Acronyms
2. Project Organization
   - 2.1 Process Model
   - 2.2 Organizational Structure
   - 2.3 Organizational Boundaries and Interfaces
   - 2.4 Project Responsibilities
3. Managerial Process
   - 3.1 Management Objectives and Priorities
   - 3.2 Assumptions, Dependencies, and Constraints
   - 3.3 Risk Management
   - 3.4 Monitoring and Controlling Mechanisms
   - 3.5 Staffing Plan
4. Technical Process
   - 4.1 Methods, Tools, and Techniques
   - 4.2 Software Documentation
   - 4.3 Project Support Functions
5. Work Packages, Schedule, and Budget
   - 5.1 Work Packages
   - 5.2 Dependencies
   - 5.3 Resource Requirements
   - 5.4 Budget and Resource Allocation
   - 5.5 Schedule

Format of the software project management plan according to IEEE-Std-1058.1.

Design

Tuesday, July 19, 2022 2:01 PM

Managing creativity

Designer experience is key

Iterate - "Plan to throw away, you will, anyhow" - Frederick Brooks

Abstraction (coupling, cohesion, information hiding, modularity)

Design for reuse

Questions to ask in examining a design.

- Is the overall organization of the program clear, including a good architectural overview and justification?
- Are modules well defined, including their functionality and their interfaces to other modules?
- Are all the functions listed in the requirements covered sensibly, by neither too many nor too few modules?
- Is the architecture designed to accommodate likely changes?
- Are the necessary buy vs build decisions included?
- Does the architecture describe how reused code will be made to conform to other architectural objectives?
- Are all the major data structures hidden behind access routines?
- Is the database organization and content specified?
- Are all key algorithms described and justified?
- Are all major objects described and justified?
- Is a strategy for handling user input described?
- Is a strategy for handling I/O described and justified?
- Are key aspects of the user interface defined?
- Is the user interface modularized so that change in it won't affect the rest of the program?
- Are memory-use estimates and a strategy for memory management described and justified?
- Does the architecture set space and speed budgets for each module?
- Is a strategy for handling strings described and are character string storage estimates provided?
- Is a coherent error-handling strategy provided?
- Are error messages managed as a set to present a clean user interface?
- Is a level of robustness specified?
- Is a part over or under architected? Are expectations in this area set out explicitly?
- Are the major system goals clearly stated?
- Does the whole architecture hang together conceptually?
- Is the top-level design independent of the machine and language that will be used to implement it?
- Are the motivations for all major decisions provided?
- Are you, as a programmer who will implement the system, comfortable with the architecture?

Software design Description

Software design Description

Each entity has:

Id
Type
Purpose
Subordinates
Dependencies
Interface
Resources
Processing (the algorithm)
Data

"None" is okay

Unified Software Development Process

Tuesday, July 12, 2022 11:43 AM

Use - case Driven

What is the system supposed to do for each user?

Also defines flow and testing

A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor.

Why?

- Capture valued requirements
- Drive the Process
- Devise Architecture

Use case model → Analysis model → Design model → Implementation Model

Use case model → Test model

Architecture - Centric

1. Rough outline
2. Write program for a specific use case
3. Mature / evolve cases
4. Repeat 1-3 until stable

Iterative and Incremental

- Deal with risks and extend usability
- Reduce cost
- Manages time
- Creates tempo
- Adaptive to user needs

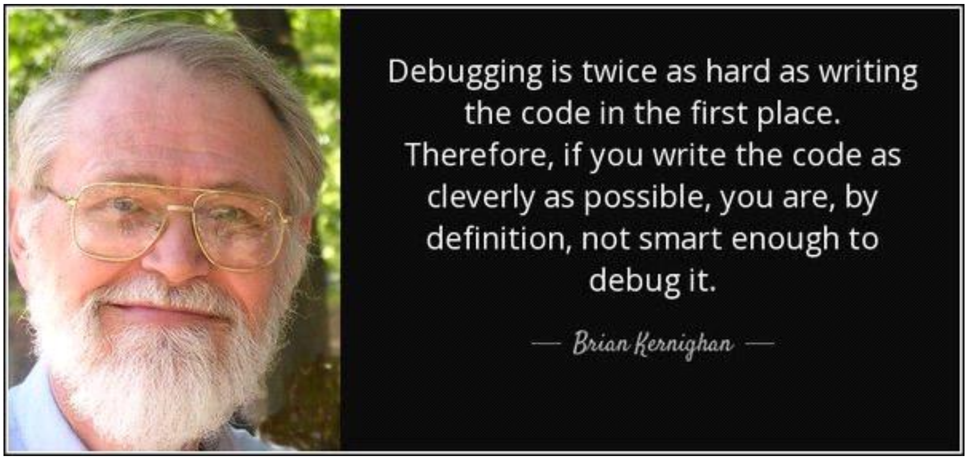
Inception → Elaboration → Construction → Transition

Integrated

- UML is a Framework

Comments

Wednesday, July 20, 2022 9:49 AM



Software Maintenance

Friday, July 22, 2022 9:37 AM

Types
- Adaptive (new functionality)
- Perfective (efficiency + readability)
- Corrective (bugs)
- Emergency (only on critical systems)

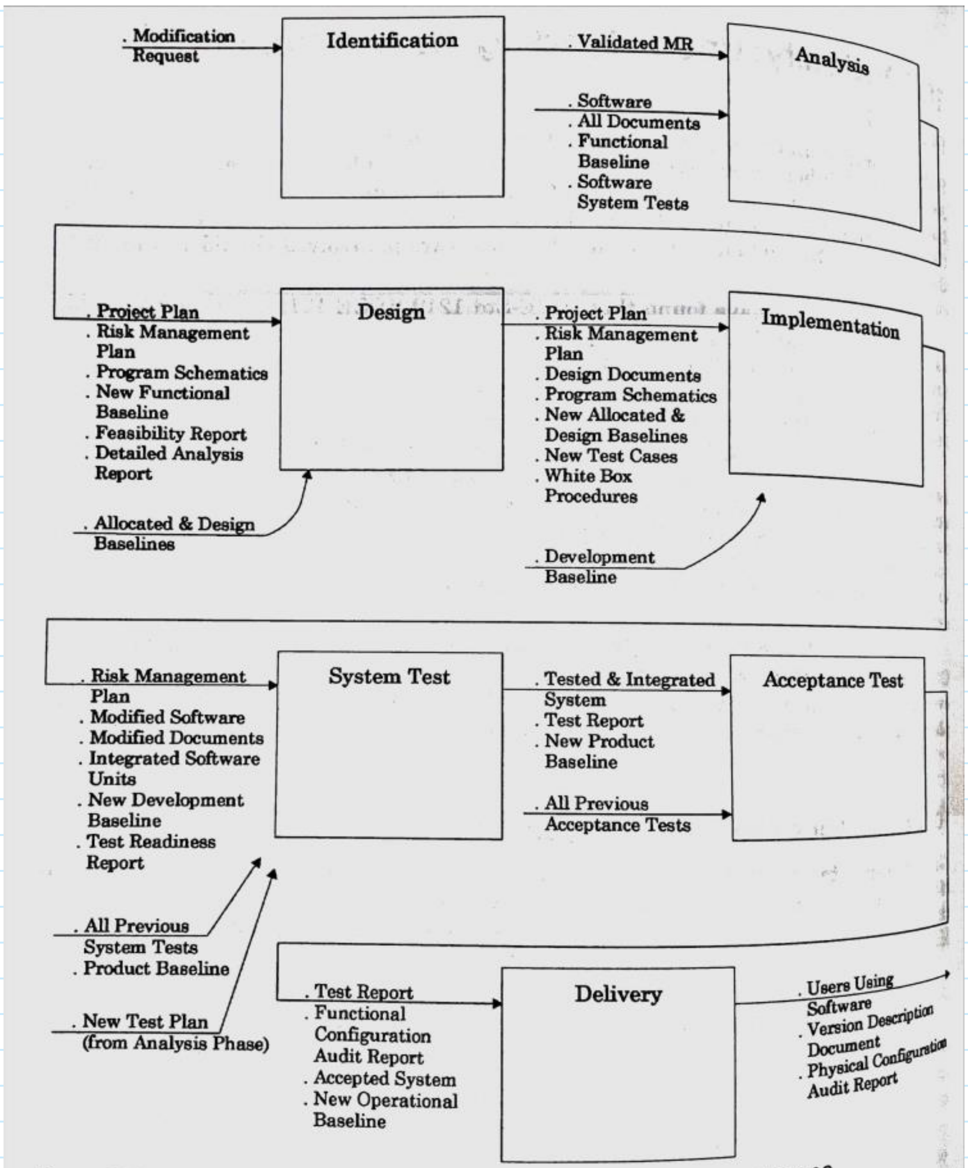


Figure 10.3 Summary of the maintenance process outlined by IEEE-Std-1219.

Code Review

Tuesday, March 25, 2025 6:36 PM

TABLE 13.1 CODE REVIEW SCRIPT

	Entry criteria	Check that the following are on hand: • The requirements statement • The program design • The program source code • The coding standards
1	Review procedure	First produce the finished program source code. Before compiling or testing the program, print out a source code listing. Next, do the code review. During the code review, carefully check every line of source code to find and fix as many of the defects as you can.
2	Fix the defects	Fix all defects found. Check the fixes to ensure they are correct. Record the defects in the Defect Recording Log.
3	Review for coverage	Verify that the program design fulfills all the functions described in the requirements. Verify that the source code implements all the design.
4	Review the program logic	Verify that the design logic is correct. Verify that the program correctly implements the design logic.
5	Check names and types	Verify that all names and types are correctly declared and used. Check for proper declaration of integer, long integer, and floating point data types.
6	Check all variables	Ensure that every variable is initialized. Check for overflow, underflow, or out-of-range problems.
7	Check program syntax	Verify that the source code properly follows the language specifications.
	Exit criteria	At completion you must have: • The completed and corrected source code • Completed Time Recording Log • Completed Defect Recording Log

Defect Log

Wednesday, August 3, 2022 2:32 PM

Defect Log - Spreadsheet

Date	Number	Type	Phase Introduced	Phase Removed	Fix Time(min)	Defects in Fixing
			Description			
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					

TABLE 12.1 DEFECT TYPE STANDARD

Defect Types		
Type Number	*Type Name*	**Description**
10	Documentation	comments, messages
20	Syntax	spelling, punctuation, typos, instruction formats
30	Build, package	change management, library, version control
40	Assignment	declaration, duplicate names, scope, limits
50	Interface	procedure calls and references, I/O, user formats
60	Checking	error messages, inadequate checks
70	Data	structure, content
80	Function	logic, pointers, loops, recursion, computation, function defects
90	System	configuration, timing, memory
100	Environment	design, compile, test, other support system problems