COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

CHAPTER 8

SOFTWARE PROJECT MANAGEMENT

M. MAHMUDUL HASAN
ASSISTANT PROFESSOR, CS, AIUB
Web: http://cs.aiub.edu/profile/m.hasan
WordPress: https://wordpress.com/view/mhasansuman.wordpress.com

FUNCTION-BASED METRICS

- The function point metric (FP), first proposed by Albrecht, can be used effectively as a means for measuring the functionality delivered by a system (e.g., size)
- Function points are derived using an empirical relationship based on countable measures of software's information domain and assessments of software complexity
  - Number of external inputs (EIs): input transactions that update internal computer files
  - Number of external outputs (EOs): transactions where data is output to the user, e.g., printed reports
  - Number of internal logical files (ILFs): group of data that is usually accessed together, e.g., purchase order file (tables in the database)
  - Number of external interface files (EIFs): file sharing among different applications to achieve a common goal (connect external database through interoperability)
  - Number of external inquiries (EQs): transactions that provide information but do not update internal file (queries in the database)

FUNCTION POINTS METRICS

Information Domain Value	**Count**		**Simple**	**Average**	**Complex**		
(FPunadjusted)							
Number of external inputs (EIs)		X	3	4	6	=	
Number of external outputs (EOs)		X	4	5	7	=	
Number of external inquiries (EQs)		X	3	4	6	=	
Number of internal logical files (ILFs)		X	7	10	15	=	
Number of external interface files (EIFs)		X	5	7	10	=	
					Count Total		

METRICS FOR OO DESIGN

Whitmire describes nine distinct and measurable characteristics of an OO design:

- **Size:** size is defined in terms of volume, length, and functionality
- **Complexity:** how classes of an OO design are interrelated to one another
- **Coupling:** the physical connections between elements of the OO design
- **Cohesion :** the degree to which all operations working together to achieve a single, well-defined purpose

METRICS FOR OO DESIGN

- **Sufficiency:** the degree to which an abstraction possesses the features required of it, or the degree to which a design component possesses features in its abstraction, from the point of view of the current application. (e.g., deals with interface and hide internals to the users)
- **Completeness:** an indirect implication about the degree to which the abstraction or design component can be reused
- **Primitiveness:** applied to both operations and classes, the degree to which an operation is atomic
- **Similarity:** the degree to which two or more classes are similar in terms of their structure, function, behavior, or purpose
- **Volatility:** measures the likelihood that a change will occur

CLASS ORIENTED METRICS

Proposed by Chidamber and Kemerer:

- Weighted methods per class - number of functions in class (WMC)
- Depth of the inheritance tree (DIT), Multilevel Inheritance
- Number of children (NOC), Multiple Inheritance
- Coupling between object classes (CBC)
- Lack of cohesion in methods (LCOM)

Proposed by Lorenz and Kidd:

- Class size (LOC)
- Number of operations overridden by a subclass
- Number of operations added by a subclass

OPERATION-ORIENTED AND CODE METRICS

❑ CODE METRICS

Proposed by Lorenz and Kidd:

- Average operation size (LOC)
- Operation complexity
- Average number of parameters per operation

❑ CODE METRICS

- Halstead's Software Science: a comprehensive collection of metrics all predicated on the number (count and occurrence) of operators and operands within a component or program
- A = C + B

METRICS FOR TESTING

- Testing effort can also be estimated using metrics
- Binder suggests a broad array of design metrics that have a direct influence on the "testability" of an OO system.
  - Lack of cohesion in methods (LCOM).
  - Percent public and protected (PAP).
  - Public access to data members (PAD).
  - Number of root classes (NOR).
  - Number of children (NOC) and depth of the inheritance tree (DIT).

MAINTENANCE METRICS

- IEEE Std. 982.1-1988 suggests a **software maturity index (SMI)** that provides an indication of the stability of a software product (based on changes that occur for each release of the product). The following information is determined:
  - M_T = the number of modules in the current release
  - F_c = the number of modules in the current release that have been changed
  - F_a = the number of modules in the current release that have been added
  - F_d = the number of modules from the preceding release that were deleted in the current release

- The software maturity index is computed in the following manner:

  SMI = $[MT - (F_a + F_c + F_d)]/MT$

- As SMI approaches 1.0, the product begins to stabilize.

BASELINE

- A *baseline* is a software configuration management concept that helps you to control change without seriously impeding justifiable change.
- The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as:

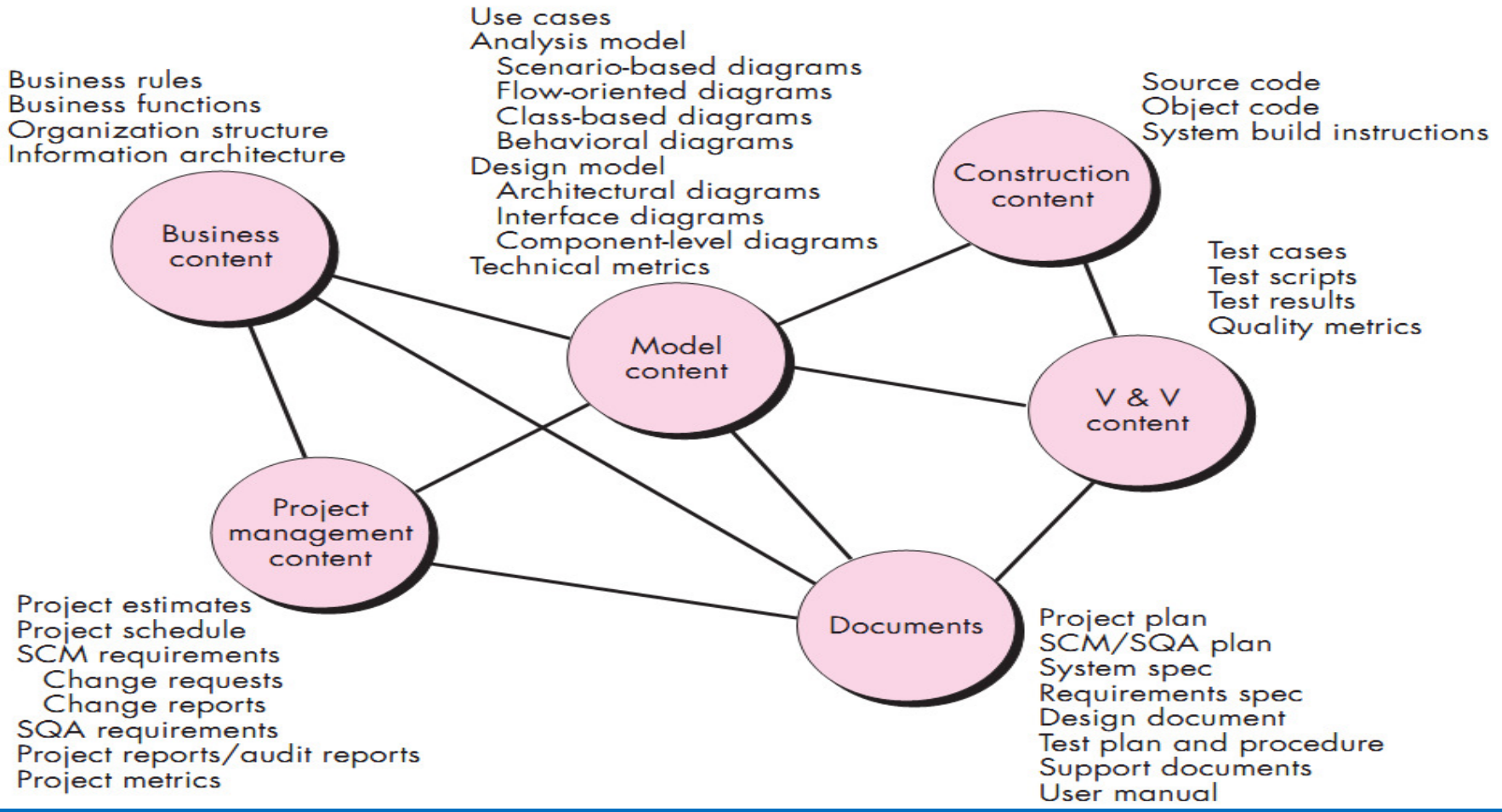
  A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

- For example, the elements of a design model have been documented and reviewed. Errors are found and corrected. Once all parts of the model have been reviewed, corrected, and then approved, the design model becomes a baseline.

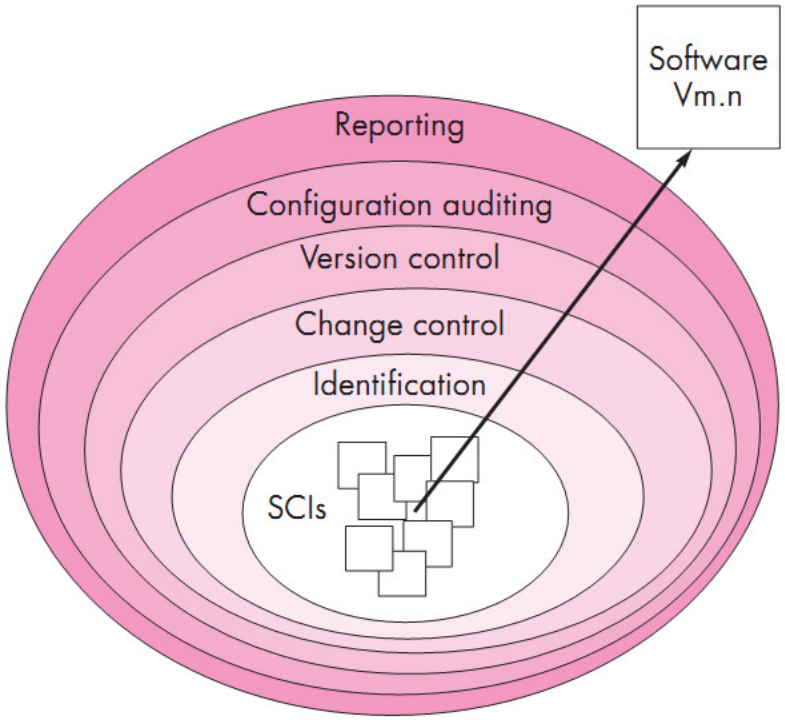
SCM REPOSITORY

- The items that comprise all information produced as part of the software development process are collectively called a *software configuration*.
- A database that acts as the center for both accumulation and storage of software engineering information
- The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner

CONTENT OF SCM REPOSITORY



SCM LAYERS



- Identification: S/w configuration items (SCIs) flow outward through these layers throughout their lifetime
- Change Control: When a new SCI is created, it must be identified, however, if no changes are requested for the SCI, the changer control layer does not apply.
- Version Control: Each SCI created is assigned to a specific version of s/w. A version control captures all changes to all files in the configuration along with the reason for changes and details of who made the changes and when
- Configuration Auditing: Regression testing
- Reporting: The record of all these SCI's (i.e., name, creation date, version, etc.) is maintained for configuration auditing purpose.

SOFTWARE PROJECT PLANNING

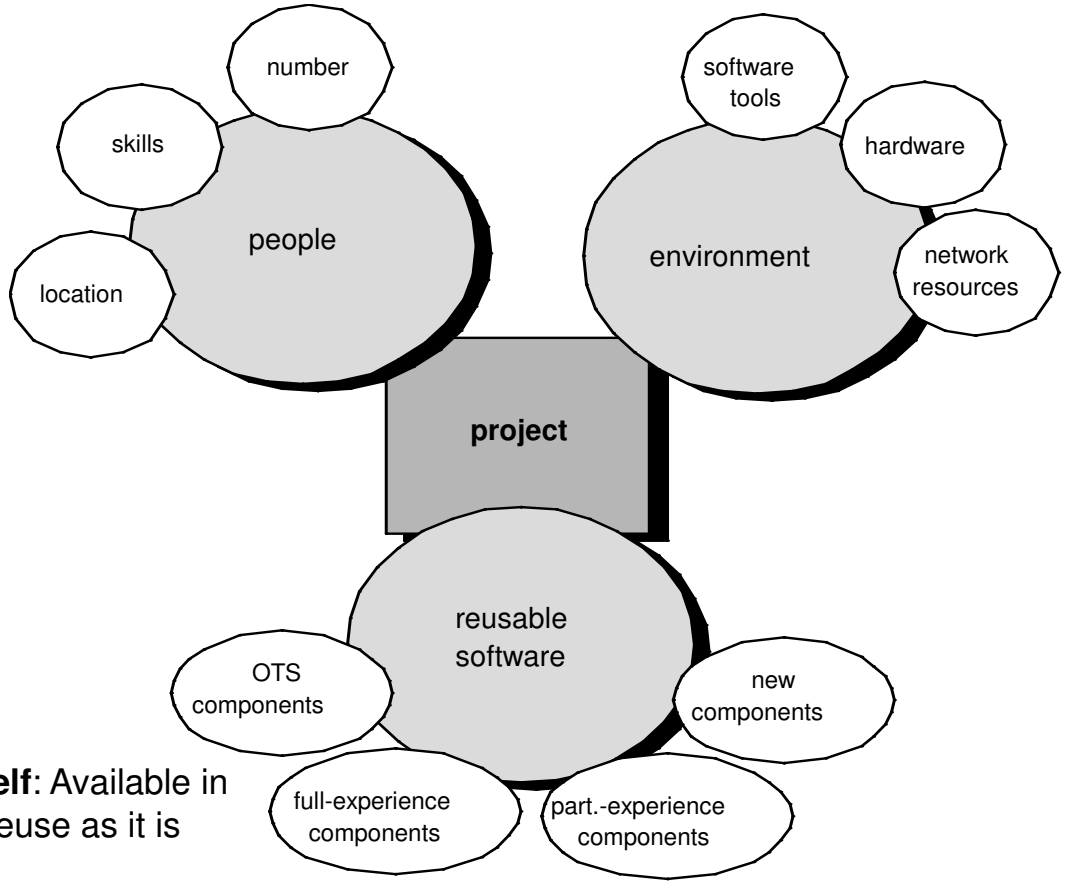
- The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

WHY?

- So, the end result gets done on time, on budget, and with quality

PROJECT PLANNING TASK SET-1

- Establish project scope
- Determine feasibility
- Analyze risks
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources



Off-the-shelf: Available in the stock, reuse as it is

COCOMO (CONSTRUCTIVE COST MODEL)

Based on SLOC characteristic, and operates according to the following equations:

- **Effort = PM = Coefficient$_{\text{<Effort Factor>}}$*(SLOC/1000)^P** [100,000 SLOC/1000 = 100k SLOC]
- **Development time = DM = 2.50*(PM)^T**
- **Required number of people = ST = PM/DM**

PM : person-months needed for project (labor working hours)
SLOC : source lines of code
P : project complexity (1.04-1.24)
DM : duration time in months for project (weekdays)
T : SLOC-dependent coefficient (0.32-0.38)
ST : average staffing necessary

Software Project Type	Coefficient <Effort Factor>	P	T
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

COCOMO (CONSTRUCTIVE COST MODEL)

Organic:

- relatively small size (2-50 KLoc), simple, less innovative software projects in which a small teams with good application experience work to a project with less tightly schedule
- Example: showing VUES information to webpage, payroll system

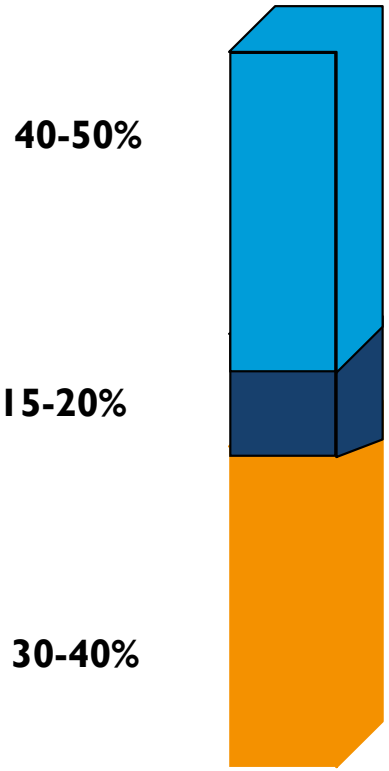
Semidetached:

- Medium size (50-300 KLoc) and innovation, complexity in software project in which teams with mixed experience levels works in a mix of h/w and s/w application with relative tight schedule
- Example: biometric log-in time saved in VUES database

Embedded:

- A Large size (>300 KLoc), more innovative, complex software project that must be developed within a strongly coupled to hardware environment (e.g., auto pilot)
- Projects has the characteristics that the product being developed had to operate within very tight constraints and changes to the system are very costly

EFFORT ALLOCATION (40-20-40 RULE)

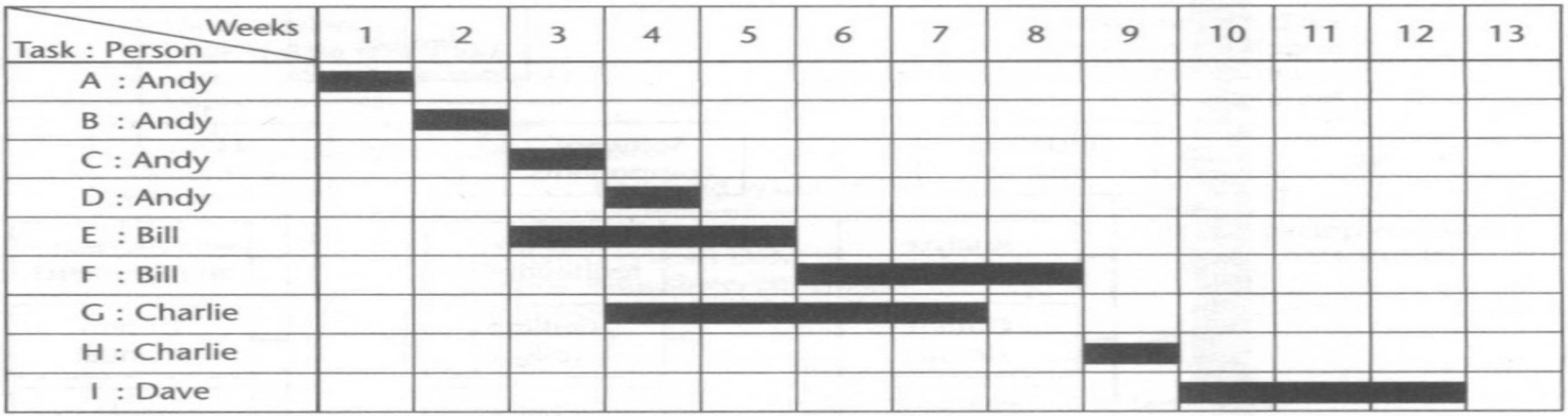


- "front end" activities
  - customer communication
  - analysis
  - design
  - review and modification
- construction activities
  - coding or code generation
- testing and installation
  - unit, integration
  - white-box, black box
  - regression

WHY ARE PROJECTS LATE?

- an unrealistic deadline established by someone outside the software development group
- changing customer requirements that are not reflected in schedule changes
- an honest underestimate of the amount of effort and/or the number of resources that will be required to do the job
- predictable and/or unpredictable risks that were not considered when the project commenced
- technical difficulties that could not have been foreseen in advance
- human difficulties that could not have been foreseen in advance
- miscommunication among project staff that results in delays
- a failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem

TIMELINE CHARTS

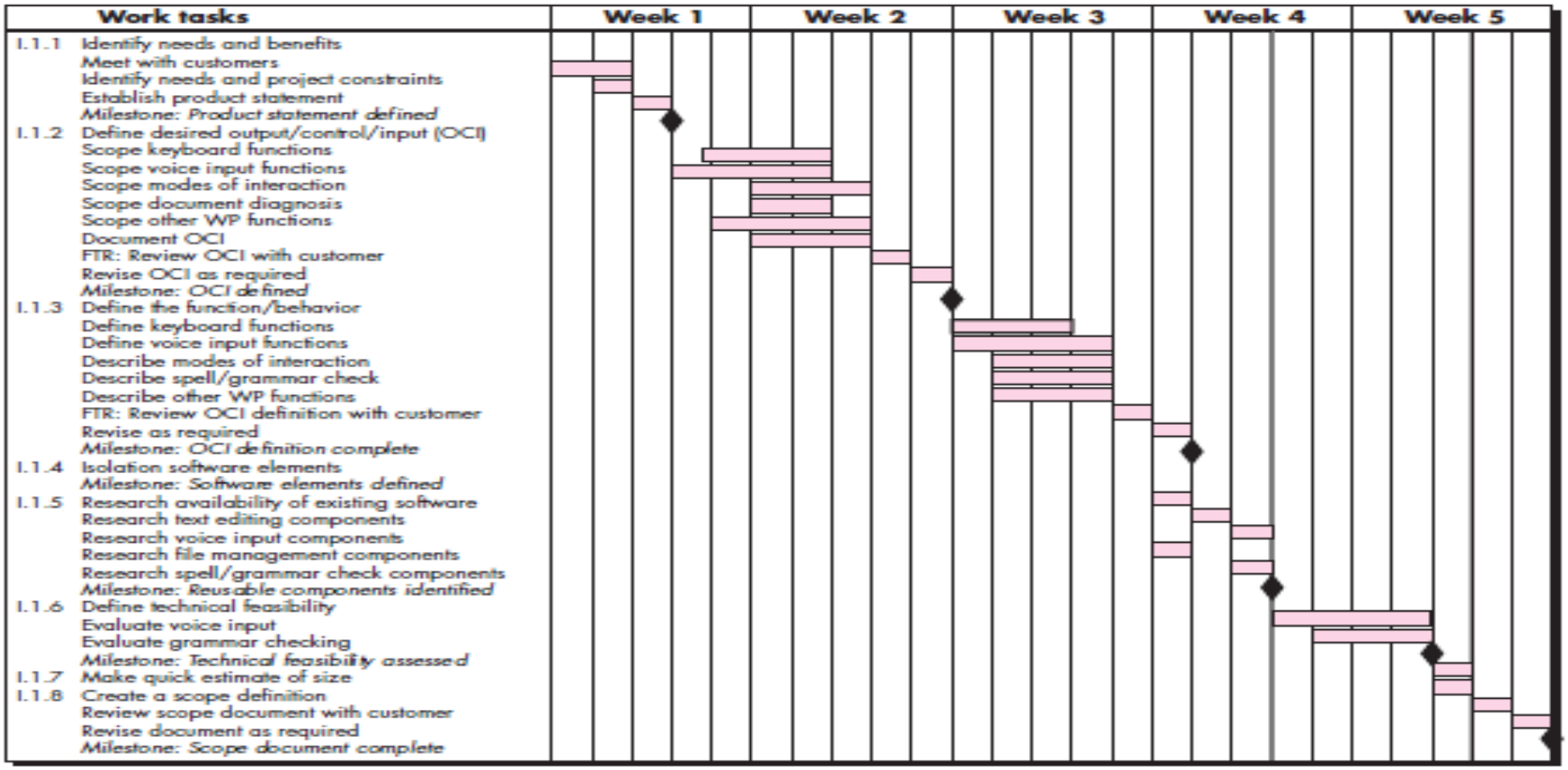


Activity key:

A : Overall design
B : Specify module 1
C : Specify module 2
D : Specify module 3
E : Code module 1
F : Code module 3
G : Code module 2
H : Integration testing
I : System testing

Figure 6.6 A project plan as a bar chart

TIMELINE CHARTS (MILESTONE)



SCHEDULE TRACKING: EARNED VALUE ANALYSIS (EVA)

- Earned value is a measure of progress. It enables us to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling. And "provides accurate and reliable readings of performance from as early as 15 percent into the project."
- EVA Exercise
  - Assume you are a software project manager and you've been asked to compute earned value statistics for a small software project.
  - The project has 56 planned work tasks that are estimated to require 582 person-days to complete.
  - At the time that you've been asked to do the earned value analysis, 12 tasks have been completed.
  - However, the project schedule indicates that 15 tasks should have been completed.
  - The following scheduling data (in person-days) are avalable:

EVA EXERCISE

Task	Planned Effort	Actual Effort
1	12.0	12.5
2	15.0	11.0
3	13.0	17.0
4	8.0	9.5
5	9.5	9.0
6	18.0	19.0
7	10.0	10.0
8	4.0	4.5
9	12.0	10.0
10	6.0	6.5
11	5.0	4.0
12	14.0	14.5
13	16.0	—
14	6.0	—
15	8.0	—

BCWP= 126.50 | BCWS= 156.50 | ACWP= 127.50

Given Total Task = 56; Effort Estimated= 582 Person Day

- BAC = 582.00 person-day
- SPI = BCWP/ BCWS = 126.5/ 156.5 = 0.808307
- SV = BCWP - BCWS = 126.5 - 156.5 = -30 person-day
- CPI = BCWP/ ACWP = 126.5/ 127.5 = 0.99
- CV = BCWP – ACWP = 126.5 - 127.5 = -1 person-day
- % schedule for completion = BCWS/ BAC
  = (156.5/ 582.00) x 100 = 26.89%
  [% of work scheduled to be done at this time]
- % complete = BCWP/ BAC
  = (126.5/ 582.00) x 100 = 21.74%
  [% of work completed at this time]

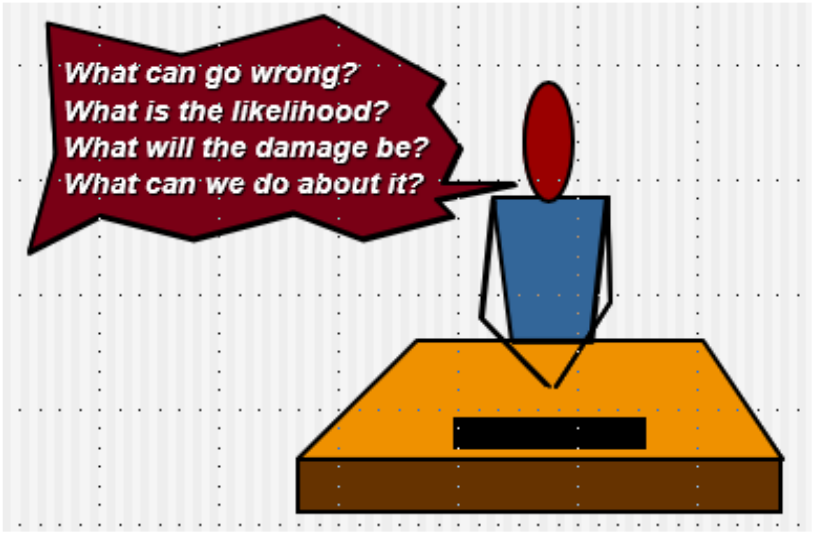
EVA EXERCISE

Measure	Equation	Comment
Earned Value (EV)	EV=% work completed × baseline cost	BCWP, Budgeted cost of work performed
Planned Value (PV)	PV=% work scheduled × baseline cost	BCWS, Budgeted cost of work scheduled
Actual Cost (AC)		ACWP, Actual cost of work performed
Schedule Variance (SV)	SV=EV-PV	
Cost Variance (CV)	CV=EV-AC	
Schedule Performance Index (SPI)	SPI = EV/PV	SPI < 1: Project is behind schedule SPI > 1: Project is ahead of schedule
Cost Performance Index (CPI)	CPI = EV/AC	CPI < 1: Project is over budget CPI > 1: Project is under budget
Cost-Schedule Index (CSI)	CSI = CPI × SPI	CSI < 1: Project is not healthy CSI > 1: Project is healthy

RISK OVERVIEW

The chance of exposure to (introduce) the adverse (opposing) consequences of future events'

- Project plans have to be based on *assumptions*. *Risk* is the possibility that an assumption is wrong. When the risk happens, it becomes a *problem* or an *issue*
- Risks are potential problems that might affect the successful completion of a software project
- Risks involve uncertainty and potential losses
- Risk analysis and management are intended to help a software team understand and manage uncertainty during the development process
- The important thing is to remember that things can go wrong and to make plans to minimize their impact when they do



RISK MANAGEMENT

Reactive

- project team reacts to risks when they occur
- mitigation—plan for additional resources to reduce the severity of damages
- fix on failure—resources are found and applied when the risk strikes

Proactive

- formal risk analysis is performed
- organization corrects the root causes of risk
  - examining risk sources that lie beyond the bounds of the software (C=A/B)
  - developing the skill to manage change

RISK PROJECTION & BUILDING A RISK TABLE

- Risk projection, also called risk estimation, attempts to rate each risk in two ways
  - Probability: the likelihood or probability that the risk is real (between 1 to 99)
  - Consequences: the consequences of the problems associated with the risk, should it occur
- The project planner, along with other managers and technical staff, performs four risk projection activities:
  - Probability: establish a scale that reflects the perceived likelihood of a risk,
  - Consequences: define the consequences of the risk,
  - Risk Exposure (Impact Score): estimate the impact of the risk on the project and the product. RE = Risk Probability x Risk Consequence
  - Accuracy: note the overall accuracy of the risk projection so that there will be no misunderstandings.

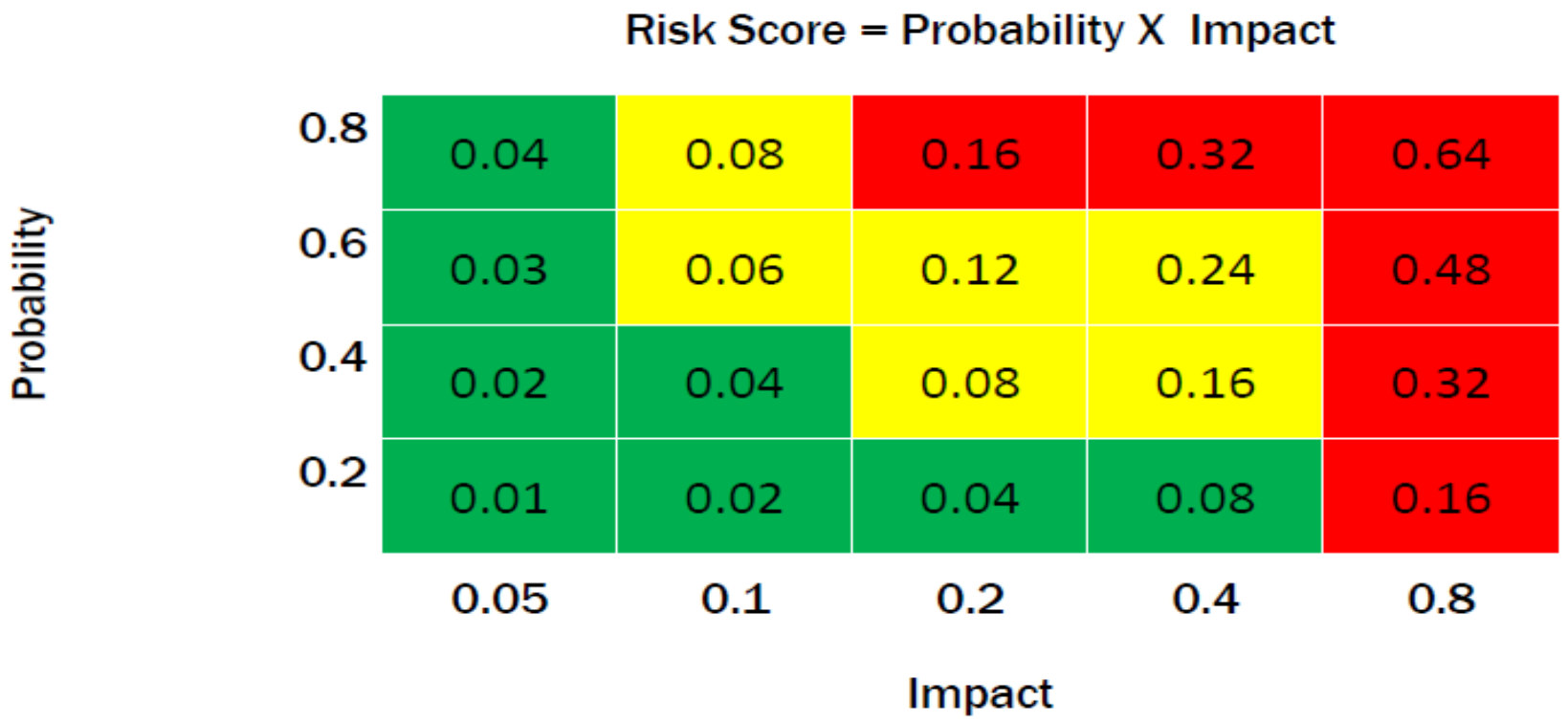
RISK COMPONENT & DRIVERS

- The major risk components (risk categories) are defined in the following manner:
  - **Performance risk:** the degree of uncertainty that the product will meet its requirements and be fit for its intended use
  - **Cost risk:** the degree of uncertainty that the project budget will be maintained
  - **Support risk:** the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance
  - **Schedule risk:** the degree of uncertainty that the project schedule will be maintained, and that the product will be delivered on time
- The impact of each risk driver on the risk component is divided into one of four impact categories— *negligible, marginal, critical,* or *catastrophic*

PROBABILITY-IMPACT MATRIX (RISK PRIORITY)

Probability	Impact: Trivial	Minor	Moderate	Major	Extreme
Rare	Low	Low	Low	Medium	Medium
Unlikely	Low	Low	Medium	Medium	Medium
Moderate	Low	Medium	Medium	Medium	High
Likely	Medium	Medium	Medium	High	High
Very likely	Medium	Medium	High	High	High

PROBABILITY-IMPACT MATRIX



RISK CHECK LIST

- Product size (PS) — risks associated with the overall size of the software to be built or modified
- Business impact (BU) — risks associated with constraints imposed by management or the marketplace
- Customer characteristics (CU) — risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner
- Process definition (PR) — risks associated with the degree to which the software process has been defined and is followed by the development organization [autopilot performance fixing with XP]
- Development environment (DE) — risks associated with the availability and quality of the tools to be used to build the product [resource allocation plan]
- Technology to be built (TE) — risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system
- Staff size and experience (ST) — risks associated with the overall technical and project experience of the software engineers who will do the work

BUILDING RISK TABLE - 2

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
•				
•				
•				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

The work product is called a Risk Mitigation, Monitoring, and Management Plan (RMMM)

CASE 1: ASSESSING RISK IMPACT

Assume that the software team defines a project risk in the following manner:

Risk identification. Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Risk probability. 80% (likely).

Risk impact. 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $14.00, the overall cost (impact) to develop the components would be 18 x 100 x 14 = $25,200.

Risk exposure. *RE* = 0.80 x 25,200 ~ $20,200.

A FRAMEWORK FOR DEALING WITH RISK - RISK MANAGEMENT

- **Risk identification** – what risks might there be?
- **Risk analysis and prioritization** – which are the most serious risks?
- **Risk planning** – what are we going to do about them?
- **Risk monitoring** – what is the current state of the risk? Must be an ongoing activity, as the importance and likelihood of particular risks can change as project proceeds.

RISK IDENTIFICATION

- Approaches of identify risks include:

  - **Use of checklists** – usually based on the experience of past projects. Some risk are generic risk, they are relevant to all software projects. A standard checklist can be used to identify the risks (e.g., changing technology).
  - **Brainstorming** – getting knowledgeable stakeholders together to pool concerns
  - **Causal mapping** – identifying possible chains of cause and effect. For example, illness of a team member is a risk that might put the project completion date at risk and result in the late delivery of the product (C=A/B)

BOEHM'S TOP 10 DEVELOPMENT RISKS

Risk	*Risk reduction techniques*
Personnel shortfalls	Staffing with top talent; job matching; teambuilding; training and career development; early scheduling of key personnel
Unrealistic time and cost estimates	Multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods
Developing the wrong software functions	Improved software evaluation; formal specification methods; user surveys; prototyping; early user manuals
Developing the wrong user interface	Prototyping; task analysis; user involvement

BOEHM'S TOP 10 DEVELOPMENT RISKS

Risk	*Risk reduction techniques*
Gold plating	Requirements scrubbing (cleaning), prototyping, design to cost
Late changes to requirements	Change control, incremental development
Shortfalls in externally supplied components	Benchmarking (evaluate by comparison with standard), inspections, formal specifications, contractual agreements, quality controls
Shortfalls in externally performed tasks	Quality assurance procedures, competitive design
Real time performance problems	Simulation, prototyping, tuning
Development technically too difficult	Technical analysis, cost-benefit analysis, prototyping , training

RISK PLANNING

Risks can be dealt with by:

- Risk prevention/avoidance – a project can, for example, be protected from the risk of overrunning the schedule by increasing duration estimates or reducing functionality.
- Risk reduction – some risk, while they cannot be prevented, can have their likelihoods reduced by prior planning. The risk of late changes to a requirements specification can, for example, be reduced by prototyping but will not eliminate the risk of late changes.
- Risk transfer – the impact of some risk can be transferred away from the project, by, for example, contracting out or taking out insurance

CASE 2: RISK REDUCTION LEVERAGE (RRL)

- Risk Reduction Leverage is another Quantitative means of assessing how Risks are being managed

$$\text{Risk Reduction Leverage} = \frac{(\text{Risk Exposure Before} - \text{Risk Exposure After})}{\text{Cost of Risk Reduction}}$$

- RE_{before} is risk exposure before risk reduction e.g., 20% chance of a fire causing $20,000 damage
- RE_{after} is risk exposure after risk reduction e.g., fire alarm costing $1500 reduces probability of fire damage to 5%

- RRL = (0.2x20,000) – (0.05x20,000) /1500
  = 4,000 – 1,000 / 1500 = 2
- RRL > 1.00 therefore worth doing

REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach.*
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, *33*(1), 182-214.