~~PART ONE : THE PRODUCT AND THE PROCESS~~

1. ~~THE PRODUCT~~INTRODUCTION TO SOFTWARE ENGINEERING

-1.1 Evolving role of s/w

Product and a vehicle to product -

As a product provide computing potential, S/w as Information Transformer – produce, manage, acquires, modify, display, and transmit

As a vehicle to deliver product - Controller, communication of info, creation & control of other programs

Important product – information, transforms personal data, manages business~~,~~ info to enhance competitiveness, provide gateway

Over the year improvement in h/w performance, change in computing~~Complex architecture~~ architecture, large memory~~,~~ & storage devices, variety of i/p, o/p devices, more sophisticated & complex computer-based system

~~Early years : S/w was an afterthought, unmanaged general purpose h/w, custom built s/w, limited distribution, developed & used by same organization, design & documentation nonexistent~~

~~Second era : multiprogramming & multi user, real time system, on line storage, s/w for distribution, larger programs, difficult to maintain,~~

~~Third era : distributed system, complex system, instant data access, and microprocessor~~

~~Fourth era : powerful desktop, sophisticated OS, n/w, client/server, information superhighway, prominent s/w industry, object oriented technology, artificial intelligence, multimedia & virtual reality~~

S/w related problems

1. Long development cycle
2. High cost
3. Error free s/w
4. Difficulty in measuring progress of s/w

1.2 SOFTWARE

—

—

~~3.~~ Definition of S/w : instructions, datastructure & documents

Formatted: Outline numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Bullets and Numbering

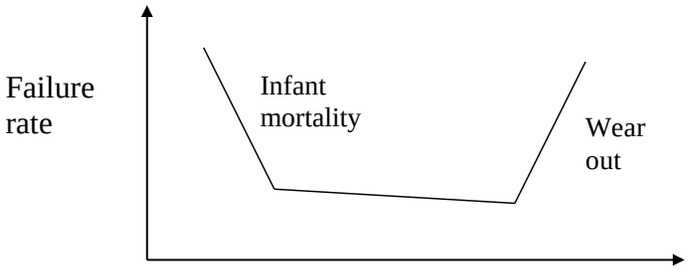
Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

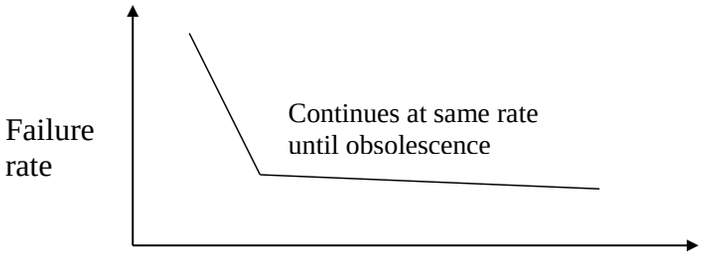
4. S/w Characteristic : s/w building is different, human creative process converted to physical form, s/w is a logical element

S/w & h/w manufactured, high quality, achieved through good design, h/w quality problem during manufactureing, relationship b/w people applied and work achieved is different, both construct a product but different approach, s/w cost concentrated on engineering so s/w projects can not be managed as manufacturing projects

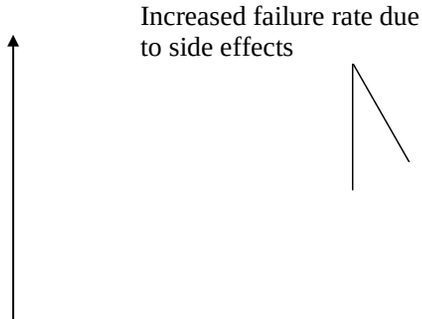
3.2. S/w does not wear out



Ideal failure rate of s/w



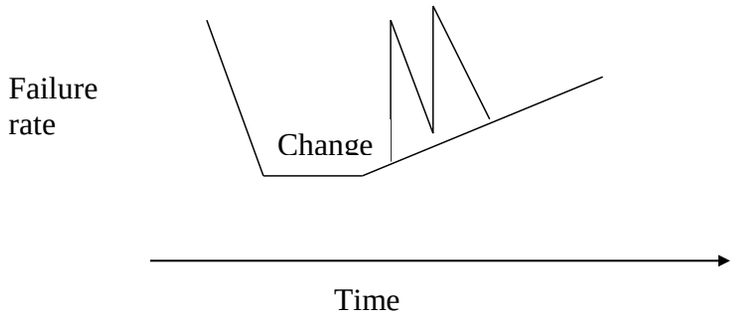
Actual failure rate



Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.52" + Indent at: 0.52"

Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.52" + Indent at: 0.52"



-1.3 THE CHANGING NATURE OF S/~~w~~W ~~applications~~ :
information content & information determinacy to determine s/w applications, Content refers to i/p & o/p data, Determinacy refers to predictability of order & timing of info.

~~1.~~1.System s/w : service other program, some os process complex & determinate info, some os process indeterminate data, heavy interaction with h/w, multiple user, concurrent operation require ~~,~~ scheduling, resource sharing, process management, complex data structure, multiple external i/f

2. ~~Real-time~~Application ~~–~~s/w : Stand alone programs that solve specific business needs, facilitate business operations & decision making, controls real-time business functions eg. Point of sale transaction processing etc
3. Engineering/scientific s/w : number crunching algos, range from astronomy, modern scientific s/w moving away from numerical algos, cad, system simulation take on characteristics of real-time or system s/w

—
features, limited functions

~~6.~~5. P~~ersonal~~roduct-line s/w: Provide specific capability for many different customers, eg. Tally, word processor

—Web-applications: set of linked hypertext files, nowadays webapps give sophisticated computation, can integrate corporate databases & business applications

7. Artificial intelligence s/w: uses nonnumerical algo to solve complex problems that can not solved by computation, robotics, expert system, artificial neural n/w

Formatted: Outline numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 3 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.75" + Indent at: 0.75"

Formatted: Bullets and Numbering

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

-Ubiquitous computing: growth of wireless n/w lead distributed computing, systems that allow small devices & computers to communicate across vast n/w

-~~netsourcing~~Net sourcing : WWW is a computing engine & content provider, applications with simple & sophisticated appli that provide benefit to end-users

-open source : distribution of source code for system appli, customer can modify locally, build self –descriptive source code

-The New Economy: the dot-com economy lead to new appli, s/w that facilitate mass communication, mass product distribution

1.4 LEGACY S/W

older programs referred as legacy s/w are of concern & continuous attention, costly to maintain, risky to evolve, characterized by poor quality, inextensible designs, poor or nonexistent documentation, poorly managed change, yet they support core business & indispensable, legacy system evolve because

- s/w must be adapted to meet new computing env or tech
- enhanced to implement new business requirement
- extended to interoperate with new system or database
- re-architect to make viable within n/w env

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

Formatted: Bullets and Numbering

Early years : S/w was an afterthought, unmanaged general purpose h/w, custom built s/w, limited distribution, developed & used by same organization, design & documentation nonexistent

Second era : multiprogramming & multi-user, real-time system, on-line storage, s/w for distribution, larger programs, difficult to maintain,

Third era : distributed system, complex system, instant data access, and microprocessor

Fourth era : powerful desktop, sophisticated OS, n/w, client/server, information superhighway, prominent s/w industry, object oriented technology, artificial intelligence, multimedia & virtual reality

~~S/w crisis~~

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

PART ONE: THE SOFTWARE PROCESS

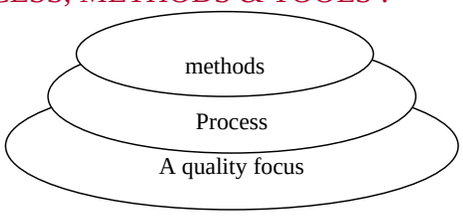
S/W process – to build a product, a series of predictable steps-a road map to create timely high quality result a framework for tasks that are required to build high quality s/w, an approach taken for engineering s/w

2.1 SOFTWARE ENGINEERING- A LAYERED TECHNOLOGY–:

SE:- Establishment & use of sound engineering principles in order to obtain economical s/w that is reliable & works efficiently on real machine

IEEE definition : SE (1) is the application of a systematic, disciplined & quantifiable approach to the development, operation & maintenance of s/w (2) the study of approaches as in (1)

~~PROCESS, METHODS & TOOLS :~~



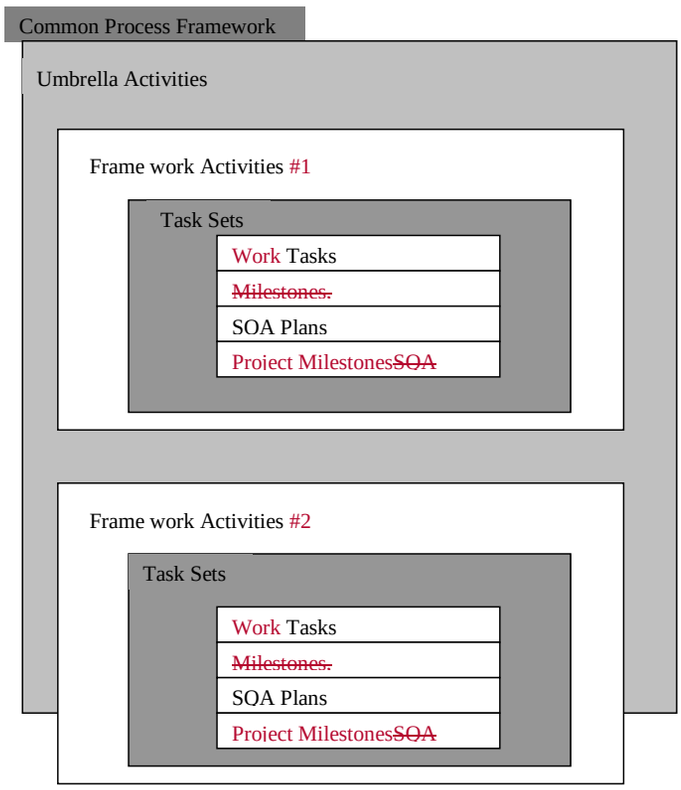
A layered technology, commitment to quality, quality management leads to improvement of process, quality focus is the bedrock

Process – foundation, holds technology layers together, enables timely & rational development, defines KPA for effective delivery of SE technology, KPA forms basis for management control of s/w projects, deals with methods applied, work product, quality & change

Methods – technical how tos for building s/w, encompasses requirement analysis, design, program construction, testing & maintenance, rely on basic principles of each technology

Tools – automated/semi-automated support for process & methods, CASE, combines s/w, h/w & SE database to create SE environment

A GENERIC VIEW IF S/W ENGINEERING :



A common process framework by defining small no of framework activities applicable to all s/w projects, a no of task sets – work tasks, project milestones, work products, QA – enables framework activities adapted to s/w projects. Umbrella activities independent of any framework & occur throughout process

Majority of the s/w follows generic process framework encompassing
Communication: Communication & collaboration with customer, requirements gathering
Planning: plan for SE, describes technical tasks to be conducted, risk identification, resource planning, work-product to be produced, schedule
Modeling: creation of a model for better understanding of s/w requirements, design of s/w
Construction: Coding & testing
Deployment: s/w delivery, evaluation by customer & feedback

Framework activities are surrounded by umbrella activities

- Software project tracking & control: assess progress against project plan take action to maintain schedule
- Risk management: assess risks that may affect project or the quality of product
- Software quality assurance: ensures s/w quality
- Formal technical reviews: assess SE work products, uncover & remove errors
- Measurement: defines & collects process, project & product measures, assist team
- Software configuration management: effects of changes
- Reusability management: work product reuse
- Document preparation & production:

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

Formatted: Bullets and Numbering

2.3 THE CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

~~Process maturity,~~ SEI has developed a model to find different levels of process capability & maturity, A Capability Maturity Model (CMM) has been defined which defined key activities required to achieve different levels of process maturity
CMMI process meta-model in 2 ways (1) continuous model (2) staged model, in continuous model process area assessed against goals & practice & rated to following capability levels

Level 0: Incomplete. Process either not performed or not achieve goals & objective defined in level 1

Level 1~~-~~: Performed. Specific Goals of process achieved, work task definition done~~Initial S/w process ad hoc/ occasionally, Chaotic~~

Level 2~~–~~: Managed. All level 1 criteria satisfied, organization has defined policy for process area, adequate resources, customer actively involved, work products monitored, controlled & reviewed~~Repeatable Basic project management process established to track cost, schedule & functionality~~.

Level 3~~–~~: Defined. Level 2 criteria satisfied. ~~Defined s/w~~ standard process organization's need, ~~for management & engineering, documented, standardized & integrated, use of documented & approved version of process for developing & supporting s/w~~work products, measures & process improvement info collected

Level 4~~-~~: Quantitatively managed. ~~Managed Detailed measure of~~ S~~s~~/w process controlled & improved~~&~~ using measurement ~~product quality are collected,~~& quantitative assessment~~ly understood and controlled~~

Level 5~~–~~: ~~–~~Optimized. ~~Continuous process improvement by quantitative feedback from p~~Process adapted & optimized to meet changes, ~~testing & technologies~~

E~~Each maturity level is associated with key process area. KPA describes those SE functions that must be present to satisfy good practice at a particular level, KPA has following characteristics~~ ach process area has Goals, Commitments, abilities, activities, Methods for monitoring implementation, methods for verifying implementation

Specific goals for project planning

1. establish estimates
   - 1.1 scope of project
   - 1.2 work product & task attributes
   - 1.3 project life cycle
   - 1.4 effort & cost
2. Develop a project plan
   - 2.1 establish budget & schedule
   - 2.2 project risks
   - 2.3 plan of data mgmt
   - 2.4 project resources
   - 2.5 knowledge & skill
   - 2.6 stakeholder involvement
   - 2.7 establish project plan

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Bullets and Numbering

Formatted: Outline numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.5" + Tab after: 1" + Indent at: 1"

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Outline numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.5" + Tab after: 1" + Indent at: 1"

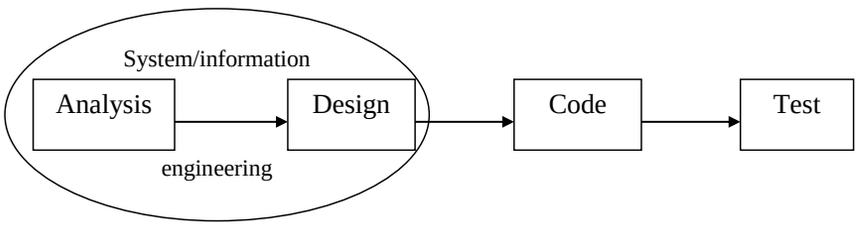
3. Obtain commitment to plan
   3.1 review plans
   3.2 reconcile work & resources
   3.3 plan commitment

Formatted: Outline numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.5" + Tab after: 1" + Indent at: 1"

3. PROCESS ~~SOFTWARE PROCESS MODELS~~

S/w development strategy, model based on nature of project, methods & tools, controls & deliverables required. Four stages :status quo, problem definition, technical development & solution integration

THE LINEAR SEQUENTIAL MODEL : classic life cycle/ water fall, systematic & sequential approach



7. System / Information engineering : part of larger system, requirement of all system elements, subset to s/w, essential for s/w interface
8. S/w requirement analysis : requirement gathering focused on s/w, info domain, required function, performance, interface, documentation
9. Design : data structure, s/w architecture, interface & procedural details, translation & assessed for quality
10. Code generation : machine readable form
11. Testing : logical internals & functional externals
12. Maintenance : changes due to error, adaptation & enhancement

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

Problems encountered

1. real project rarely follow sequential flow
2. difficult to state all the requirements explicitly
3. patience
4. delayed works

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

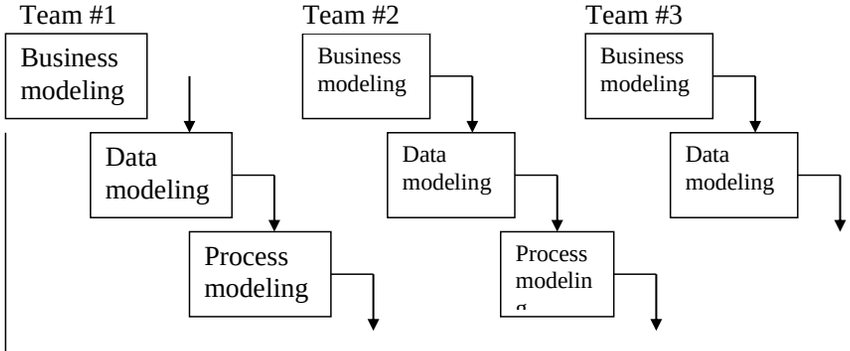
PROTOTYPING MODEL :

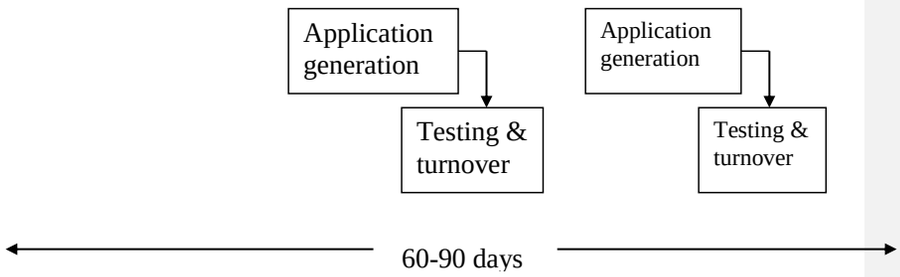
THE RAD MODEL :

A high speed linear sequential model, short development cycle, component based construction

13. Business modeling : info flow among business, info that drives business, generated, where & who
14. Data modeling : info refined into data object that supports business, characteristics & relationship
15. Process modeling : data objects to achieve info flow for implementing a business function, process description
16. Application generation : use of 4th GT & reuse of components
17. Testing & turnover : through testing & integration

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

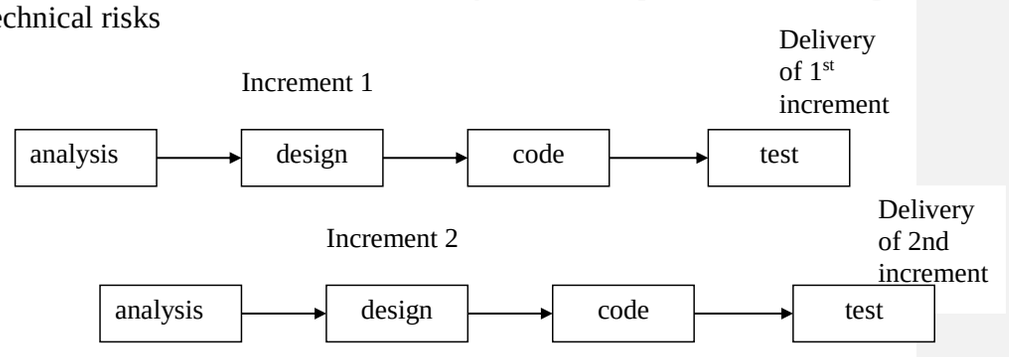




Applicable to modularized business applications, each separate major function by a separate RAD team, sufficient human resources, full commitment, only properly modularized systems, not appropriate for applications with high technical risk

~~EVOLUTIONARY S/W PROESS MODELS~~

THE INCREMENTAL MODEL : combines elements of linear model with iterative prototype, linear model in staggered fashion, each linear sequence produces deliverable in increments, first increment a core product with basic functionality, core product used/evaluated by customer, plan for next increment, modification & additional features & functionality, focus on delivery of operational product in each increment, useful when unavailability of staff, planned to manage technical risks

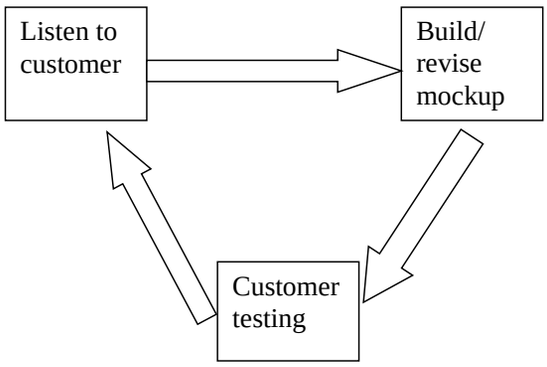


EVOLUTIONARY S/W PROESS MODELS

S/w evolves over time, business & product requirement change, tight market deadlines, difficult to produce comprehensive product, a limited version must be produce, a process model that accommodates evolution, iterative in nature, enables increasingly more complete version

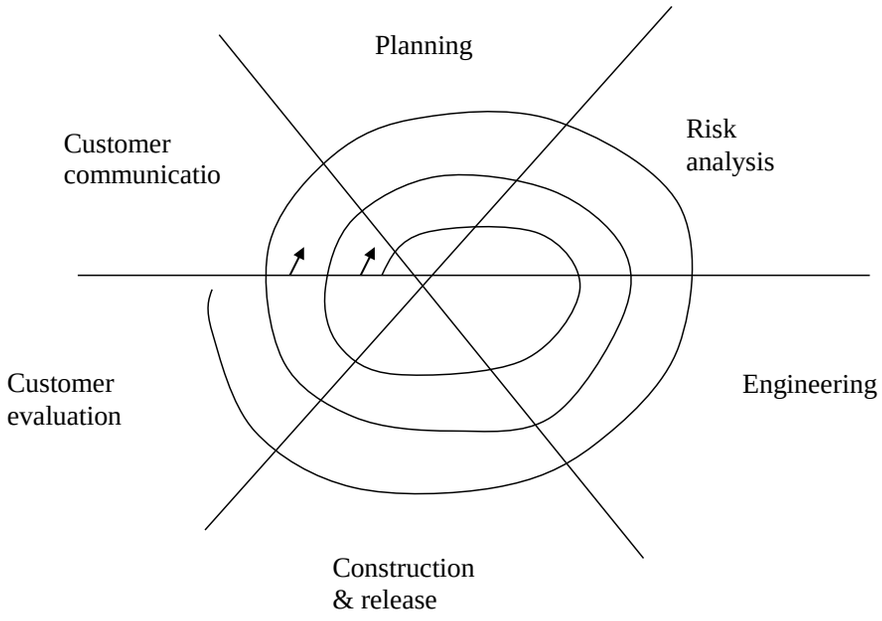
PROTOTYPING MODEL :

When difficult to define details, unsure of efficiency, begins with requirement gathering, objective of s/w, quick design, visible to customer like I/p o/p format, construction of prototype, evaluation, refine the requirements, iterative procedure



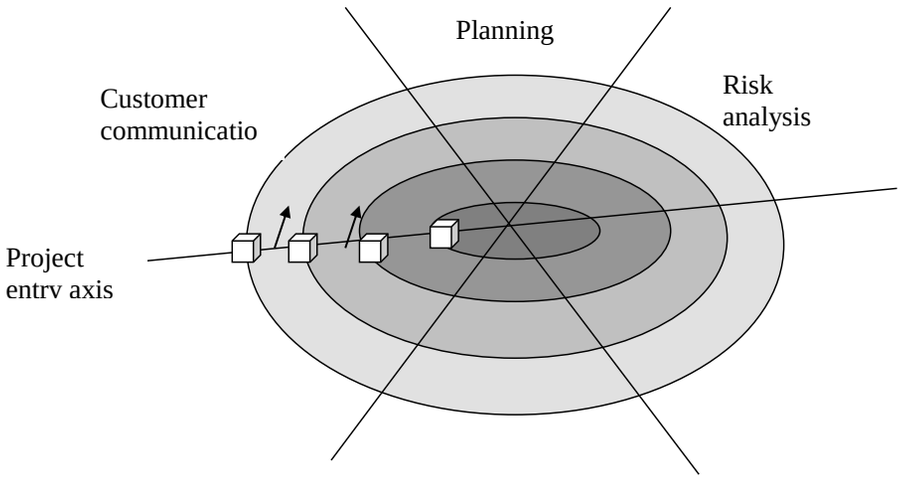
Problems : customers like to make it a working product, developers accept its inappropriateness

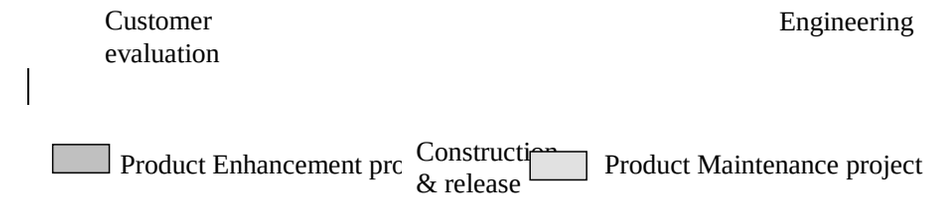
THE SPIRAL MODEL : combines iterative prototype with controlled & systematic linear model, series of incremental release, early iteratione may produce paper work/prototype, later more complete version, divided into no of framework activities/regions
Customer communication : effective commu b/w developer & customer
Planning : define resources, time
Risk analysis :assess technical & management risk
Engineering : representation of application
Construction & release : construct, test, install & user support
Customer evaluation : customer feedback



Each region has set of work tasks, s/w engineering team moves around spiral in clock wise direction starting at center, first circle as development of product specification, subsequent for develop a prototype & more sophisticated s/w, planning region adjusts project plan, cost & schedule adjusted on feedback, applicable throughout life of s/w

Project entry point axis





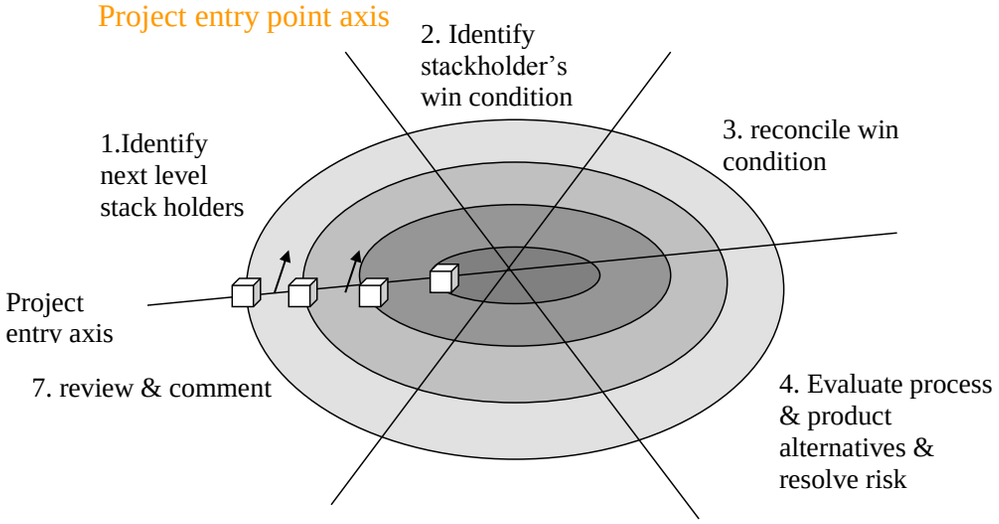
Each cube represents starting point for different type of project. Concept development project at the core & continue until complete, next new product development project, and operative throughout the life of s/w life.

A realistic approach to large scale systems, better understanding of risk, sometimes difficult to convince customer thast this process is controllable, require considerable risk assessment expertise, undiscovered risk causes problem, not used widely.

THE WINWIN SPIRAL MODEL :

Customer communication activity in spiral model to get requirements of customer, ideally developer asks customer about requirement & customer provide sufficient info to proceed, in reality customer & developer negotiates, customer to balance functionality, performance & other system characteristics against cost & timing, A win-win situation, customer wins by getting product thewt satisfy majority of needs, developer wins by realistic & achievable budget & deadline

Project entry point axis



6.Validate product & process Definitions

5.define next level of product & process

Developed by Boehm, defines negotiation activities at each pass, a single customer communication activity replaced by

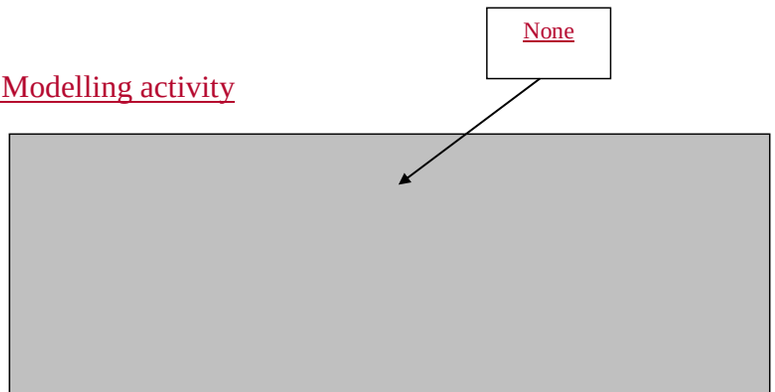
1. identification of system or subsystem's stack holders
2. determination of stackholder's win condition
3. Negotiations

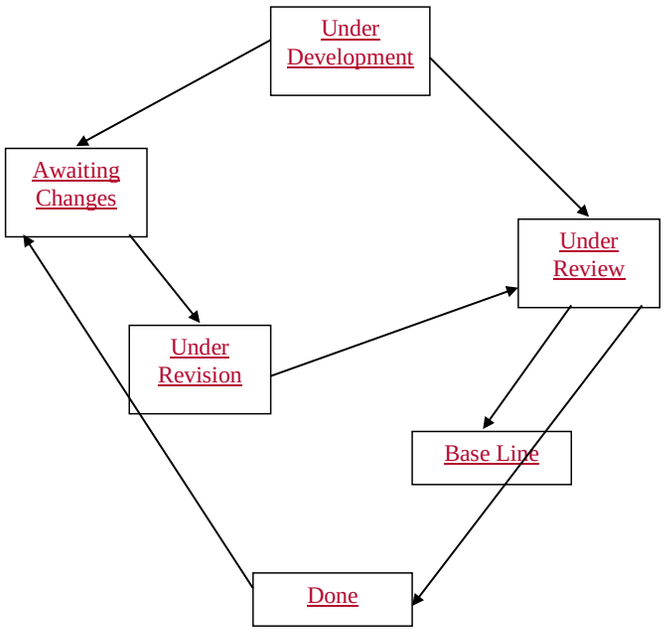
Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Successful completion achieves win-win result, introduction of three milestones, caleed anchor points, first is life cycle objectives, defines a set of objectives for each major se activity, second is life cycle architecture, defines objectives for system s/w architecture, third is intial operational capabilities, objectives associated with s/w installation/distribution, site preparation

THE CONCURRENT DEVELOPMENT MODEL:

Concurrency for activities during one phase, series of major technical tasks, activities or associated activities, concurrent but different stages, ex customer communication complete, awaiting changes state, analysis activity in under development state, if change is required, awaiting change





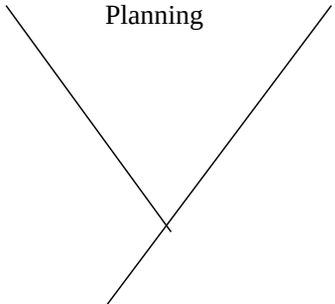
This process model applicable to all process model, provides picture of current state, defines network of activities rather than a sequence,

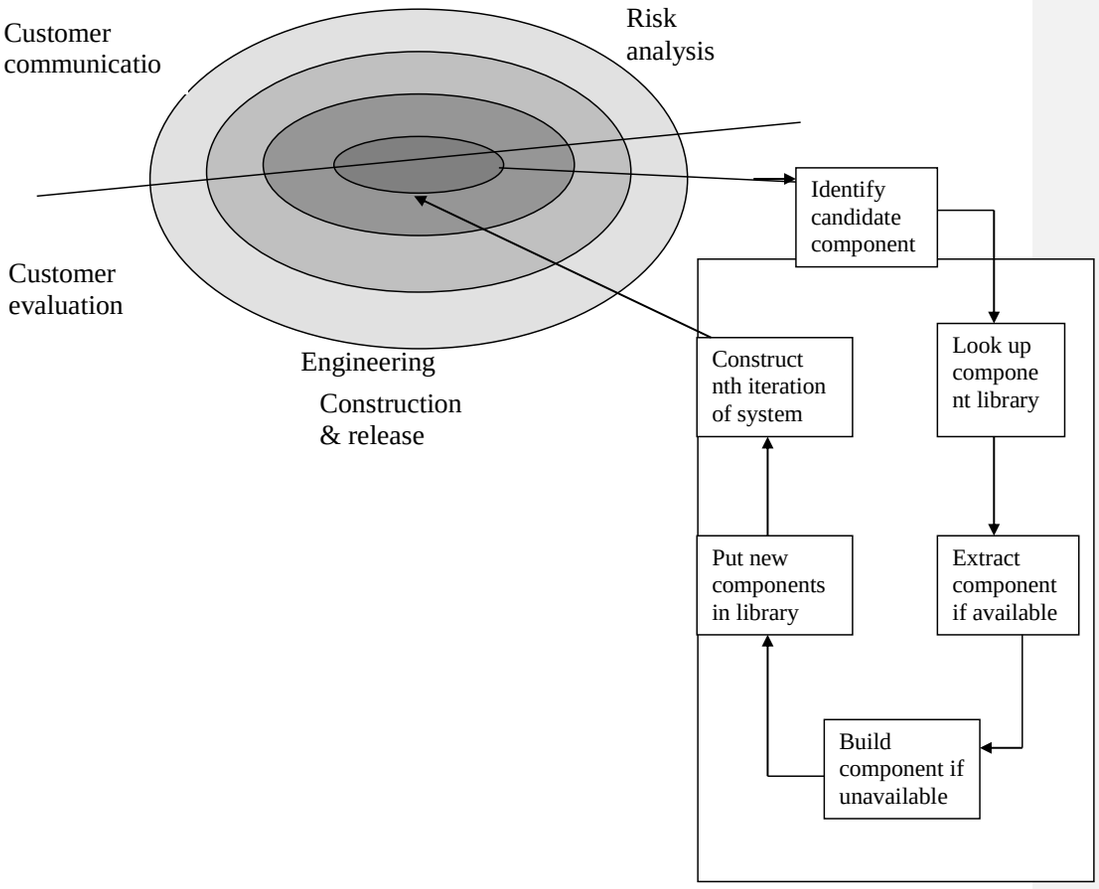
Intent of evolutionary models to develop high quality product, limitations do exist- prototype model poses planning problem because no of iterations, speed is slow, balance b/w extensibility & flexibility & the quality

3.5 Specialized process models

THE COMPONENT-BASED MODEL :

Based on object oriented technology, classes that encapsulates data & methods reused, similar characteristics as spiral model





Engineering activity starts with identification of candidate classes, class library, searching, reuse, engineered using object oriented method, flow return to spiral

70 % reduction in development cycle time, 84% in project cost, productivity index 26.2 against 16.9

THE FORMAL METHODS MODEL :

Set of activities that leads to mathematical specification of computer s/w, specify, develop & verify computer based system by applying mathematical notations, a variation "clean room se, provide mechanism to eliminates difficult problems, incompleteness & inconsistency

discovered & corrected, ~~in~~during design serves as program verification, thereby detect & correct errors
Time consuming & expensive, requires training, difficult to use as a communication method

ASPECT-ORIENTED SOFTWARE DEVELOPMENT

S/w engineer implement a set of localized features, functions & info, these localized features modeled as s/w component & constructed within s/w architecture, these featres are customer required properties or areas of technical interest, span entire archi, some are high leve properties of system ex security, fault tolerance, other affects functions ex business rules, others are systemic ex task synchronization, memory mgmt,

AOSD is new that defines, specifies, design & construct these aspects, may adopt characteristic of spiral & concurrent models, aspect identified & constructed in iteration & aspects are constructed independently of s/w component but direct impact on it.

FOURTH GENERATION TECHNIQUES :

S/w process mgmt tools, define a complete s/w process model ex GDPA – research process definition tool suit, SpeeDev – suite of tool for process definition, requirement mgmt, project planning & tracking, Step Gate Process – tools for workflow automation,

specify characteristics of s/w at high level, generates source code, ability to specify s/w in specialized language or graphics, nonprocedural queries, report generation, screen interaction, code generation

Starts with requirement gathering, translated into operational prototype, necessary to use design strategy, testing & integration, reduces s/w development time, improve productivity not easy to use, inefficient source code

The Unified Process

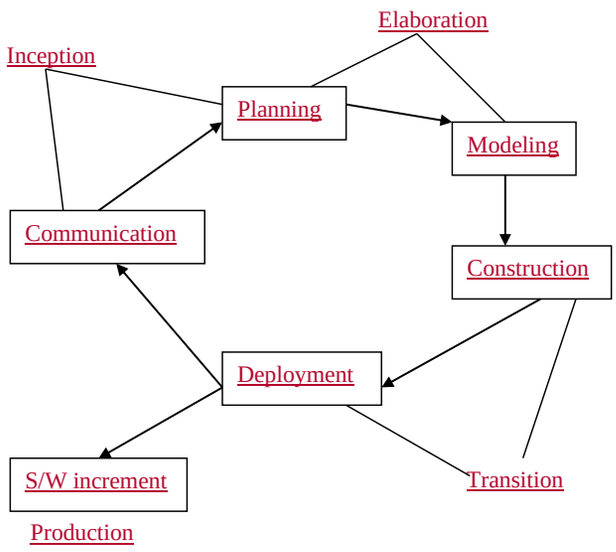
It's a use-case driven, archi-centric, iterative & incremental s/w process,

Use-case : text narrative or template that describe a system function or feature from user's point of view, written by user & serve as a basis for creation of comprehensive analysis model

A unified modeling language that contains notation for modeling & development of OO system, became std, Rational co. developed tools to support UML methods, UML provides techno to support OO SE but not the process framework, Unified process is a framework for OO SE using UML

There are phases of UP, *Inception phase* encompass customer communication & planning, business requirements for s/w identified, rough archi for system, plan for project , business requirements described through set of use-cases, defines desired features & functions by major class of users, use-case describe a sequence of actions that are performed by an actor as interacting with s/w, help identify scope of project & provide basis for project planning

A tentative outline of major subsystem & major functions & features, this archi later expanded to represent diff views of system,



Elaboration Phase encompass customer communication & modeling, elaboration refines & expands preliminary use-cases & expands archi to include 5 diff view of s/w 1) use-case model 2) analysis model 3) design model 4) implementation model 5) deployment model, elaboration gives first cut executable system, demonstrate viability of archi, not all the features & functions, review of plan to assess scope, risks & delivery dates

Construction phase- using archi Model as i/p develops s/w component making each use-case operational, design model turned in s/w as increment, necessary & required functions then implemented in source code, components build, unit test designed & executed, integration activities, from use-case a suite of acceptance tests

Transition phase – latter stage of construction, beta-testing of s/w, feedback on defects & necessary changes, creation of support info such as user manual, trouble shooting guide
Production phase – deployment of s/w, on-going use of s/w monitored, support for operating env, defect reports & change request

These 5 UP phases may occur concurrently, task sets identified as workflow, workflow identifies tasks required to accomplish imp SE actions, work products produced as successful completion of task

Unified Process Work products

Inception phase – vision document
- Initial use-case model
- Initial project glossary
- Initial business case
- Initial risk assessment
- Project plan – phases & iteration
- Business model
- Prototypes

To establish overall vision of project, identify business requirements, define business & project risks, collection f use-cases – describes interaction with the system, usage scenario that imply s/w features 7 functions, describe preconditions, flow of events, post-conditions

Formatted: Indent: Left: 1.5", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25", Tab stops: Not at 0.25"

Formatted: Bullets and Numbering

Elaboration phase – use-case model

- Supplementary requirements
- Analysis model
- S/w archi description
- Executable archi prototype
- Preliminary design model
- Revised risk list
- Project plan – iteration, workflows, milestones, tech work products
- Primary user manual

Produces set of work products that elaborate reuirements, archi description & primary design, s/w engineer begins OO analysis, defines classes that describe behavior of system, analysis model ids developed, representation of s/w archi, revisit risks & project plans

Construction phase- Design Model

- S/w components
- Integrated s/w increment
- Test plan & procedures
- Test cases
- Support documentations – user manual, installation manual, description

Design classes translated into s/w components to realize the system, mapping of components into physical computing env, tests to ensure use-cases are reflected in s/w

Transition phase – Delivered s/w increment

- Beta test reports
- User feedback

Delivers s/w increment & assess work products, feedback from beta testing

Se model by s/w project team, process technology tools help s/w developer to analyze current process, organize tasks, control & monitor progress, manage quality. Automated model of common processes & tasks, analyzed to determine workflow, examine alternatives, reduce time & cost

Formatted: Indent: Left: 1.63", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25", Tab stops: Not at 0.25"

Formatted: Bullets and Numbering

Formatted: Indent: Left: 1.63", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25", Tab stops: Not at 0.25"

Formatted: Bullets and Numbering

Formatted: Indent: Left: 1.63", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25", Tab stops: Not at 0.25"

Formatted: Bullets and Numbering

PRODUCT & PROCESS :

If the process is weak, the end product will undoubtedly suffer. But over reliance on process is also dangerous. Developer should derive as much satisfaction from the process as the end product.

—

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

In 2001, 17 s/w developers, writers & consultants signed manifesto that values

Individuals & interactions over process & tools
Working s/w over comprehensive documentations
Customer collaboration over contract negotiations
Responding to change over following a plan

Agile method to overcome weaknesses of conventional s/w engineering, can provide benefit, not applicable to all projects, can be applied as overriding philosophy, in changing & competitive business env – changing market conditions, evolution of end users, new competitive threats -s/w eng must be agile o respond to fluid business env

Agility is appropriate response to change, change in s/w being built, change in team members, new technology, impact on product
Agility encourage team structure & facilitates communication, emphasize rapid delivery of operational s/w, de-emphasize intermediate work products, customer as part of development team, flexible project plan

4.1 what is agility

Formatted: Indent: Left: 0", Hanging: 0.44", Outline numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.75" + Indent at: 0.75", Tab stops: Not at 0.75"

Formatted: Bullets and Numbering

12 principles

1. Highest priority is to satisfy customer through early & continuous delivery of valuable s/w
2. welcome changing requirements even late in development, harness change to customer's competitive advantage
3. deliver working s/w frequently, shorter timescale
4. business people & developer should work together daily throughout the project
5. Build project around motivated people, give env & support, trust
6. face-to-face conversation – a most effective method of conveying info within a development team
7. working s/w is primary measure of prograss
8. agile process support sustainable development, maintain a constant pace indefinitely
9. Continuous attention to tech excellence & good design enhances agility
10. simplicity is essential
11. best archi, requirements & design emerges from self-organizing teams
12. at regular intervals, team reflects on how to become more effectively, tunes & adjusts behavior

4.2 WHAT IS AN AGILE S/W?

1. Difficult to predict in advance which s/w requirement persists & which will change. Also difficult to predict about customer priorities as the project proceeds
2. For s/w, design & construction are interleaved. Both activities in tandem so design model proven as created, difficult to predict hoe mush design before construction to prove design
3. Analysis, design, construction & testing are not as predictable

To manage all these unpredictability- agile process, agile process must be adapted incrementally, requires customer feedback, for this purpose prototype is used

4.2.2 Human Factors: emphasize on people factor for successful agile s/w, agile development focuses on talent & skill of individual, process molds to the need of people & team, not other way round, team member should have

Competence. Encompass innate talent, s/w related skills, knowledge of process to be applied, skill & knowledge can be taught

Formatted: Outline numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.75" + Indent at: 0.75"

Formatted: Bullets and Numbering

Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.51" + Indent at: 0.51"

Formatted: Outline numbered + Level: 3 + Numbering Style: 1, 2, 3, … + Start at: 2 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.76" + Indent at: 0.76"

Formatted: Bullets and Numbering

Common Focus. May perform diff tasks & have diff skills, focused on one goal-deliver working s/w increment within the time, also focus on continual adaptation

Collaboration. SE assesses, analyzes & uses info communicated to SE team, creates info that will help customer to understand work, builds info to provide solution, to accomplish, team collaborates with customers

Decision-making ability. S/w team allowed the freedom of its own destiny, autonomy-decision making authority for tech & project issues

Fuzzy-problem solving. Agile team continuously encounters change, problem might change everyday, lessons from problem solving activities beneficial later.

Mutual trust & respect. Strongly knit that the whole is greater than sum of parts

Self Organization. Implies 1) agile team organizes itself for the work to be done 2) organizes process to accommodate local env 3) organizes work schedule to achieve the time limit. Many benefits- improves collaboration, boost team morale.

PART TWO: SOFTWARE ENGINEERING PRACTICE

~~160~~. SYSTEM ENGINEERING

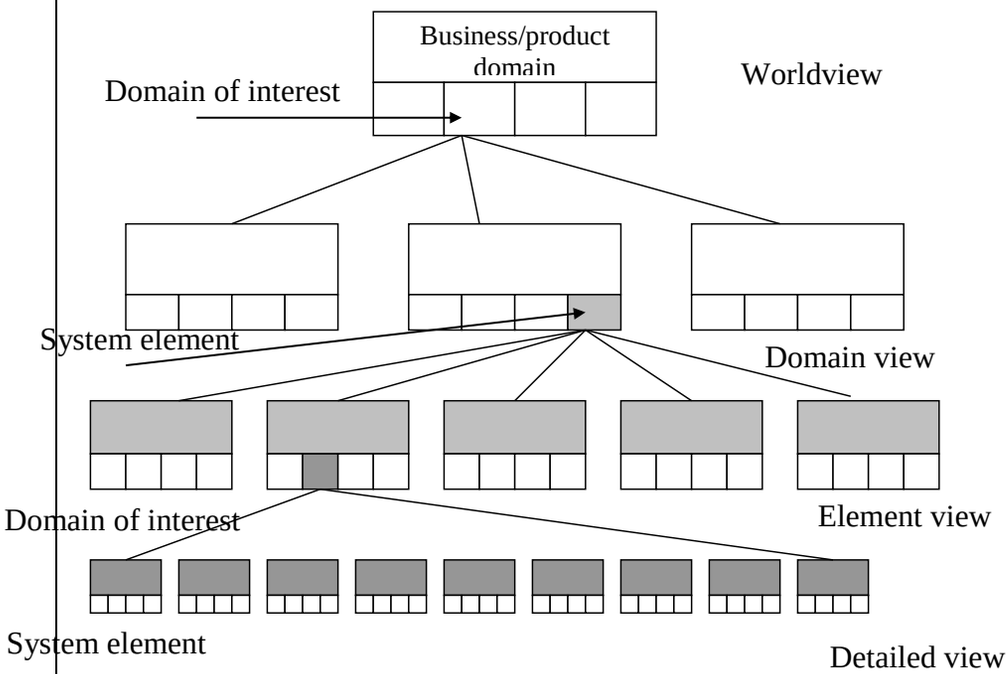
SE a part of system engineering, focus not only on s/w but a variety of elements of a system, analysis, design & organization of these elements to be a product/service – business process engineering if focus on business enterprise – product engineering when product is built, bring order to the development of computer-based system

COMPUTER BASED SYSTEMS :
Computer based system = A set / arrangements of elements that are organized to accomplish some predefined goal by processing info
Goal – support business function/ develop product
Computer-based system use no of system elements : (1) S/w – programs, data structure, documents, provides logical method, procedure or control (2) H/w - computing capability, interconnectivity device for data flow, electromechanical device for external world function (3) People – users & operators (4) Database – collection of info assessed by s/w (5) Documentation – info about use & operation of system (6) Procedures – steps defining use of each element
Combination of different elements for info transform, ex marketing dept transform raw sales data into profile of purchaser, robot transform command file into a set of control signals, creation of both require system engineering, One computer-based system as a macro element of larger computer-based

system, Ex. Factory Automation System – lowest level numerical control machine, data entry devices, each a computer-based system, elements of numerical control machines are h/w, s/w, database, documents etc, at next upper level manufacturing process a computer-based system, it has its own elements & also integrates numerical control machines, data entry device etc, each system has generic elements – s/w. h/w, people, database, procedures & documentation, some system shares generic elements, some has exclusive, System engineering defines elements for computer-based system n context of hierarchy of system

SYSTEM ENGINEERING HIERARCHY :

System engineering uses top-down & bottom-up methods to navigate hierarchy



Begins with world view, entire business/product domain examined, proper business/technology context established, world view refined to focus on domain of interest, need of targeted system elements analyzed, analysis, design & construction of system element initiated, at top broad context established, at bottom detailed technical activity conducted, thus world view composed of set of domains, each can be a system, Domain composed of elements, serve some role to accomplish goal of domain, element is

implemented by technical components, provide specific function, Focus narrows as moving downward

6.2.1 SYSTEM MODELING : System modeling an imp process of SE, Engineer creates model that

- Defines the processes for view under consideration
- Represent behavior of processes & their assumption
- Define exogenous & endogenous i/p
- Represent all linkage

Formatted: Indent: Left: -0.13", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

Restraining factors to be considered are

- Assumption : reduces no of permutations & variations, reasonable i/p, ex 3d animation- ability to specify human movement, range of allowable movement, limited
- Simplification : enables model created in timely manner, Ex a co. providing sales & services, service demand in two broad categories, internal & external- a simplified partition
- Limitations : bound the system, Ex. Aircraft avionics system, always two engines, propulsion monitoring model to accommodate it
- Constraints : Guide the creation & implementation of model, ex. Technological infrastructure
- Preferences : preferred architecture for data, function & technology, customer satisfaction predicated on degree to which preferred approach realized

Formatted: Indent: Left: -0.13", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

6.2.2 SYSTEM SIMULATION :

Many system interact in reactive manner, h/w & s/w that monitors real world events, based on event they impose control on machines, processes etc, real-time & embedded s/w, such systems difficult to predict their performance & behavior, if fails cause human 7 economic loss, system modeling & simulation help to eliminate surprises, applied during system engineering process, enables engineer to test drive before implementation

6.3 BUSINESS PROCESS ENGINEERING : AN OVERVIEW

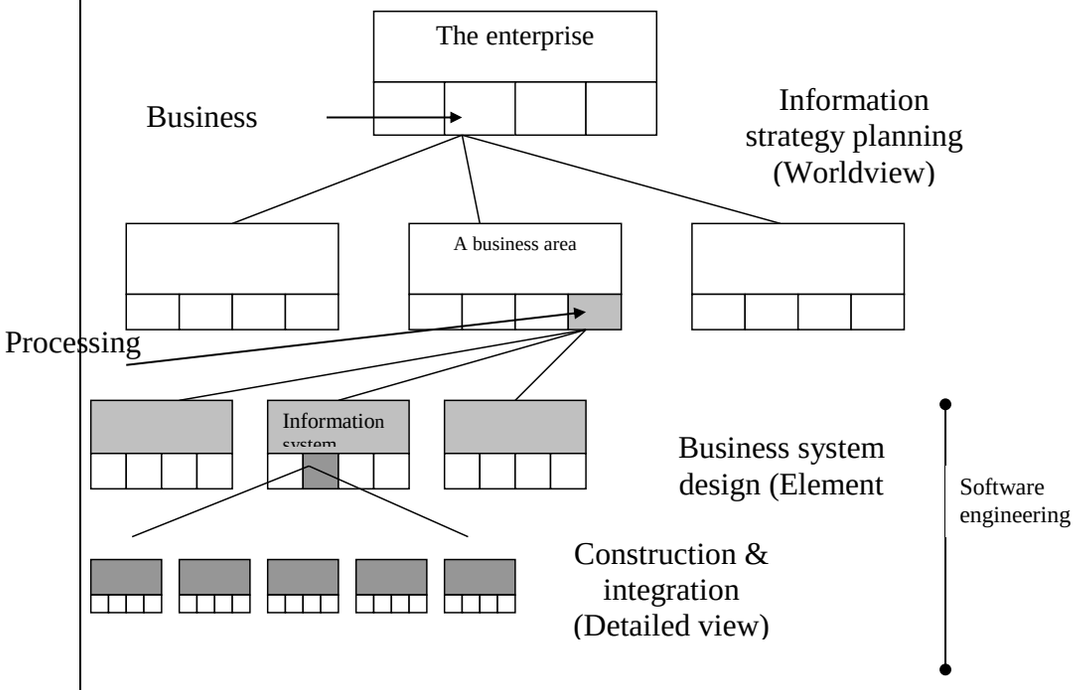
The goal is to define architectures that enable a business to use info effectively, BPE is an approach to create overall plan for implementing computing architecture,

Three different architecture within context of business objectives & goals- data, applications & technology architectures

Data architecture provides framework for info needs of business function, data objects serve as individual building block, contains a set of attributes that define quality, characteristic of data, Ex customer data object contains name, company, designation, address etc, relationship b/w data objects defined, data objects flow b/w business functions, stored in database, transformed to provide info

Application archi – contains elements that transform objects of data architecture, ex programs, also include role of people & business procedures

Technology ~~archi~~infrastructure – foundation for data & application archi, h/w & s/w used to support data & application, computer, os, n/w, telecommunication links



A hierarchy of BPE activities to model system archi, world view achieved thro Info Strategy Planning (ISP), views business as an entity, isolates domains of business such as manufacturing, marketing etc, defines data objects at enterprise level, relationship & how they flow b/w business domains

Business area analysis (BPE activity) focus on domain view, views business area as an entity, isolates functions & procedures of business area, defines data objects, relationship & flow, areas of opportunity is found where info system may support business area

After info system isolated, Business System Design step models basic requirements of specific info system, translated into data, application & technology architecture

Final step – Construction & Integration focus on implementation – data structure, data base, s/w components, ~~technology~~and technology infrastructure selection – to support design created by previous step. Each system components integrated to form complete info system.

6.4 PRODUCT ENGINEERING

Goal is to customer's requirements into a working product, must derive archi & infrastructure, archi encompass – s/w, h/w, data & people, infra includes techno to tie components together,

6.5 SYSTEM MODELING

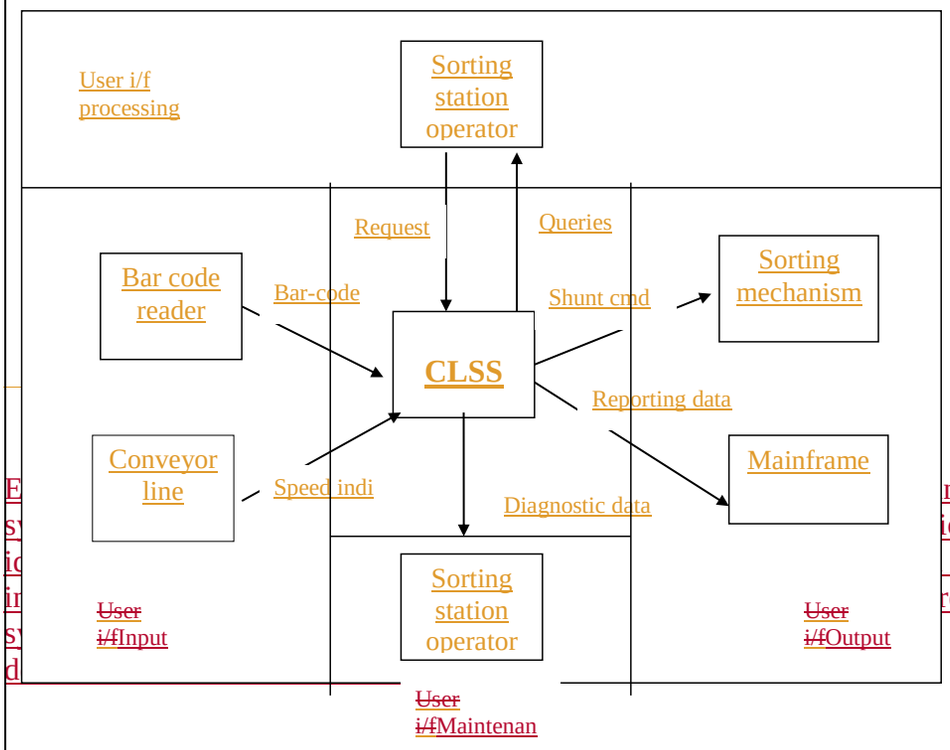
System models are layered in nature, at the top major data objects, functions & behavior without implementation concern, level is refined, component level detail & finally engineering model specific to appropriate engineering

6.5.1 HATLEY-PIRBHAI MODELING

Computer-based system as info transform as i/p-processing-o/p template, this model extended to include user i/f processing & maintenance & self test processing, a system model is created using above representations, system elements allocated to 5 processing regions – 1) user i/f, (2) i/p (3) system function & control (4) o/p (5) maintenance & self test, a system model template is used to create a hierarchy of details, system context diagram developed to establish info boundary b/w system implemented & env in which system is to be operated, i.e. scd defines all info producers, external consumers of info created by system & i/f

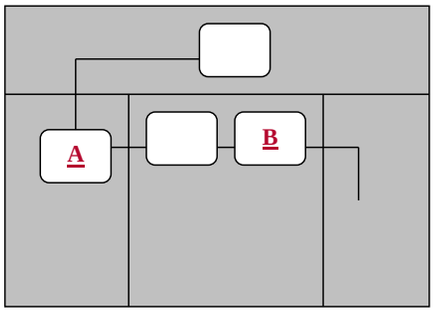
Conveyer line sorting example – boxes moving on conveyor line identified & sorted into one of the 6 bins, sorting station identifies boxes from no printed on box & bar code, shunted into appropriate bin, random in order & evenly spaced, computer containing CLSS s/w interacts with bar-code reader, interacts with conveyor line monitoring equip for conveyor speed, store all

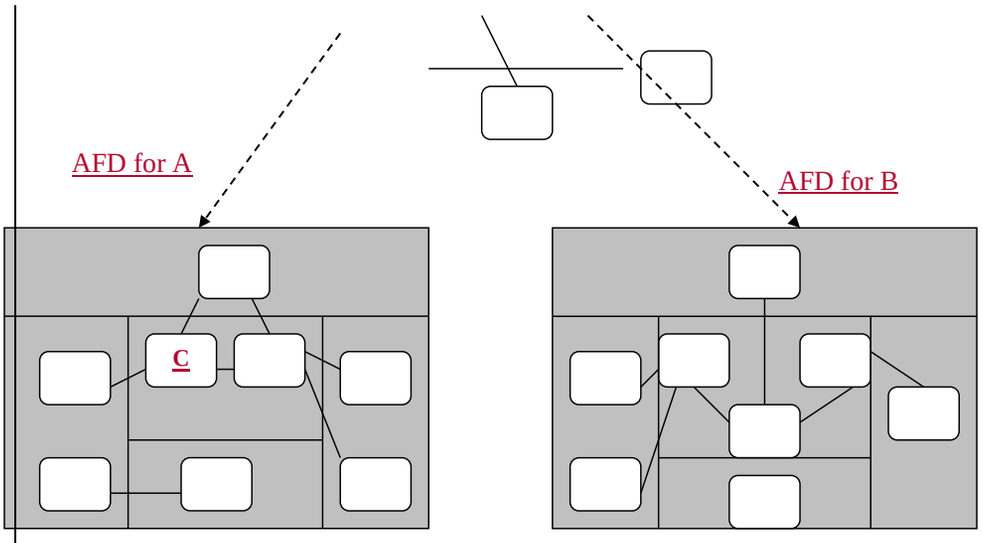
part no sorted, produces variety of reports, sends signal to shunting h/w to sort boxes, communicated to central factory automation system



E nfo,
s ions
ic &
ir re
s
d

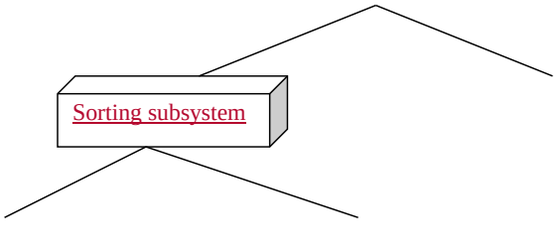
Top-level architecture flow diagram (AFD)

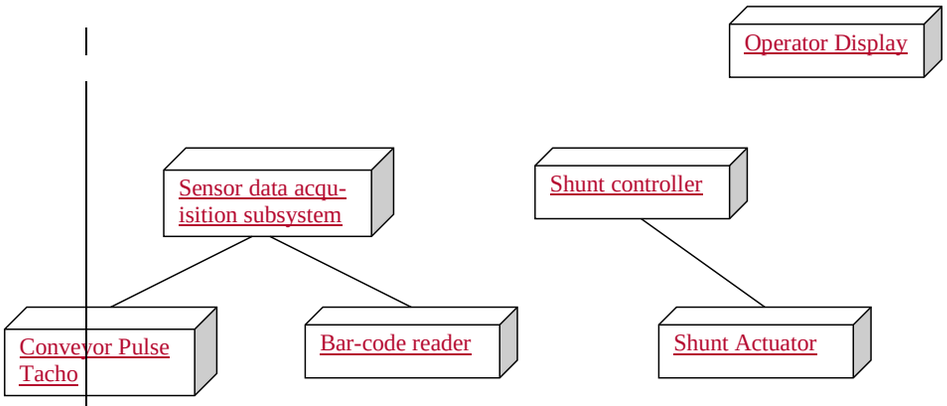




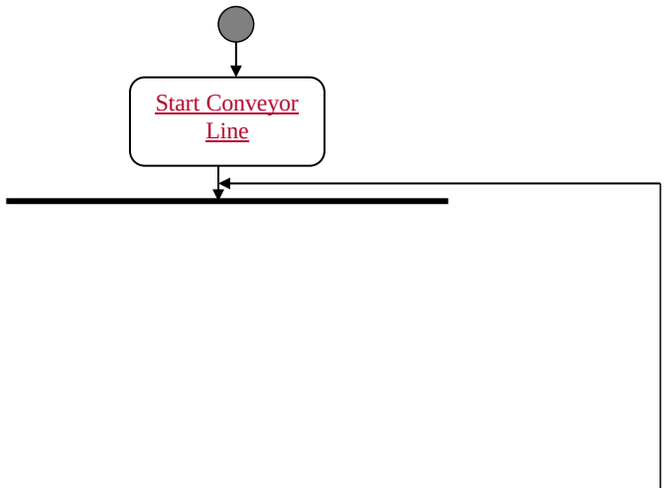
6.5.2 System Modeling with UML

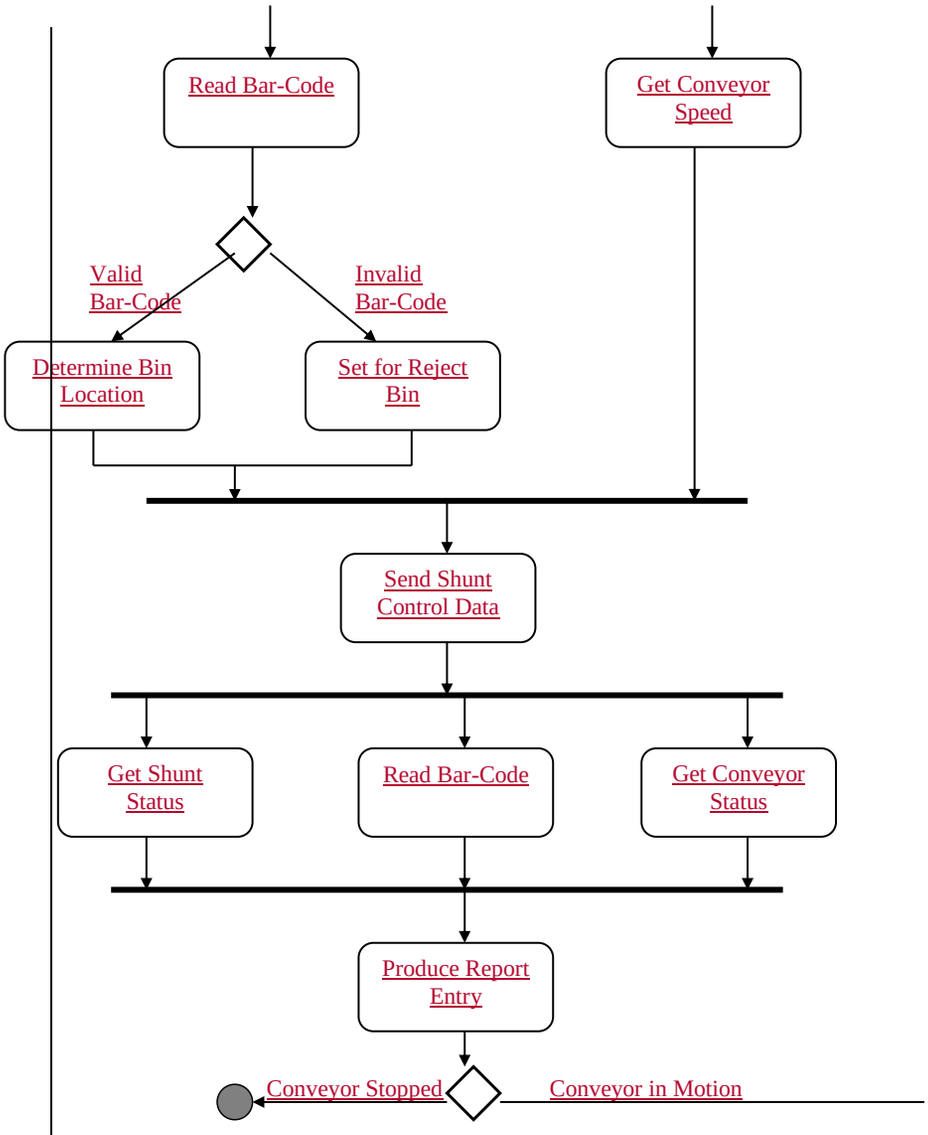
Analysis & design tool at system & s/w level, for CLSS system, 4 system elements are modeled (1) h/w that enables CLSS (2) s/w that implements database access & sorting (3) operators who submits various request to system (4) database that contain bar-code & destination info
CLSS h/w modeled at system level using UML deployment diagram, generally h/w acquired off-the-shelf, challenge is to properly i/f h/w elements



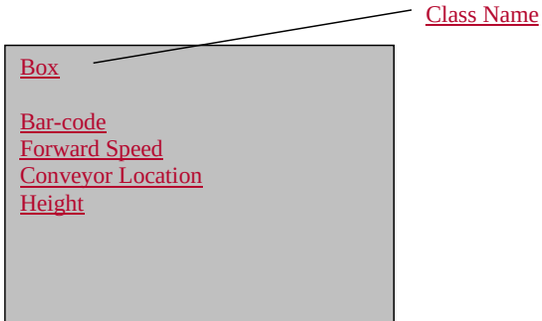


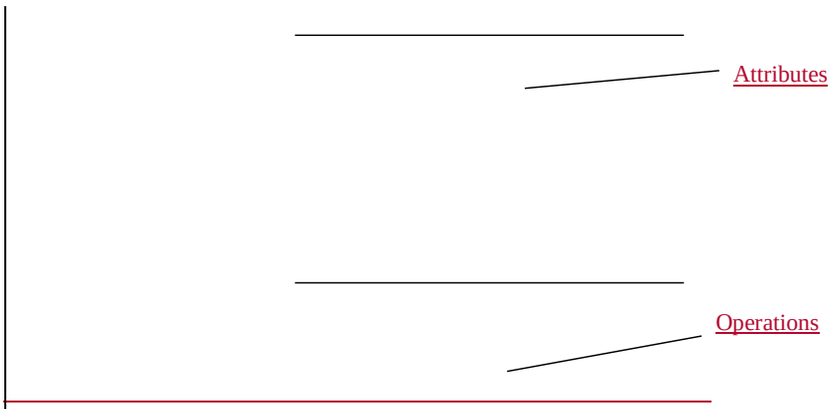
Procedural aspects of CLSS s/w represented using activity diagram, this UML notation similar to flowchart, used to describe system functions, rectangle specify system function, arrows imply flow through system, diamonds represents branching decision, solid horizontal lines imply parallel activities



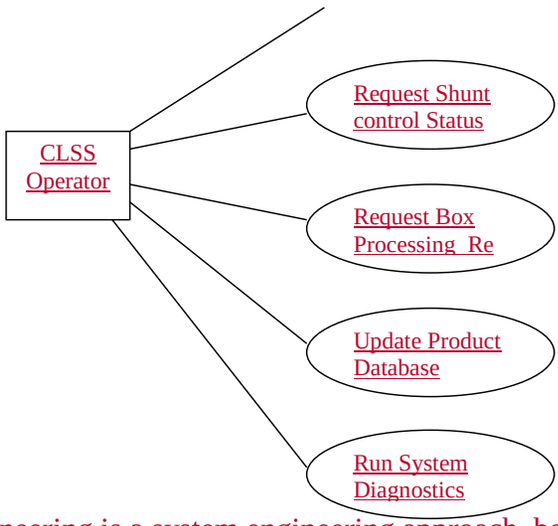


Another UML notation is Class Diagram, at system level classes are extracted fro statement of the problem, for CLSS candidate classes might be – **Box, Conveyor line, Bar-code Reader, Shunt Controller, Operator Request, Report, Product** etc, each class encapsulates a set of attributes that depict all necessary info about the class, also contain a set of operations applied to the class in context of CLSS system, UML class diagram for **Box**





CLSS operator can be modeled with UML use-case diagram, illustrates manner in which operator interact with system, each use-case describes an interaction with the system



Product engineering is a system engineering approach, begins with system analysis, system engineering identifies customer's needs, determine

economical & technical feasibility & allocate function & performance to s/w, h/w, people & database

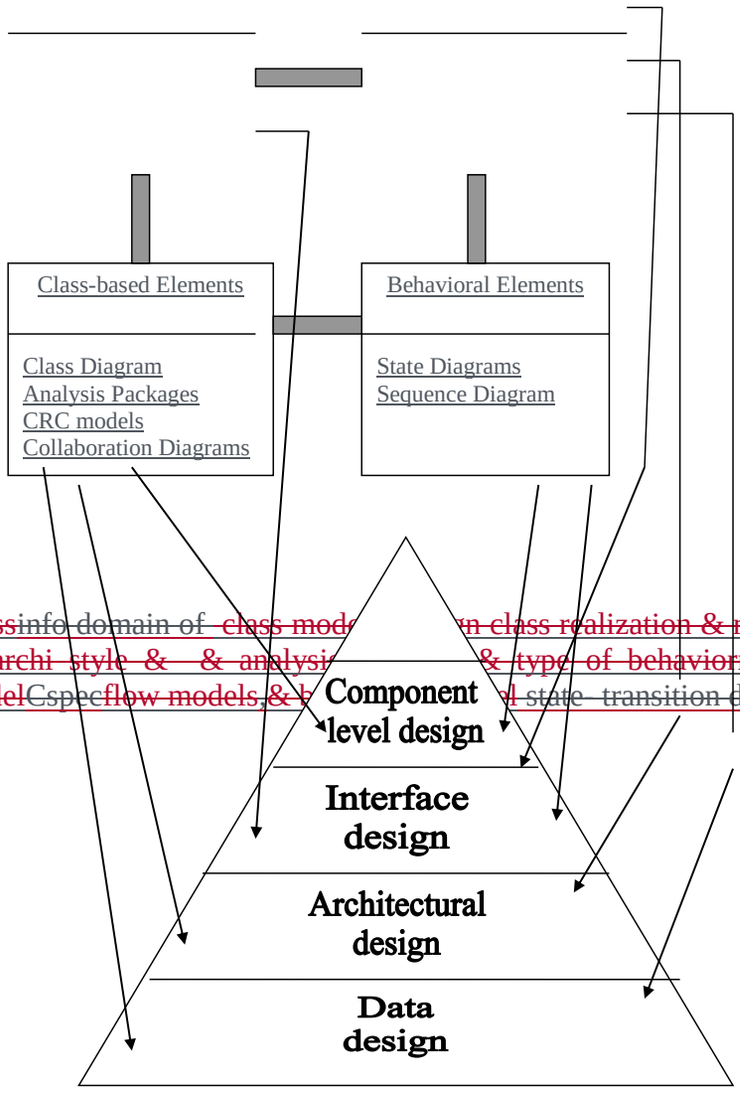
913. DESIGN ~~CONCEPTS AND PRINCIPLES~~ENGINEERING

Goal is to produce a model that will be built later, 2 major phase, Diversification – acquisition of materials such as components, component solution & knowledge ~~etc~~, Convergence – gradual elimination of all but one component, to final product, demands intuition & experience, based on principles guides the way in which model is built, set of criteria enables quality judgment & a process leads to final design

9.1 DESIGN WITHIN THE CONTEXT OF S~~/~~OFTWARE ENGINEERING ~~DESIGN & SE~~ :

S/w design at kernel of SE, applied regardless of process model, after analysis & specification design starts, first of three technical activities to build a product, elements of analysis model provide info to create four design models – data/class design, archi design, i/f design, component design

Data/class design transforms analysis class models into design class realization & required data structures, data objects & their relationship defined in ER diagram & details of data in data dictionary forms basis, Architectural design define relationship b/w major structural elements of s/w, archi style & design pattern to achieve requirements of system, constraints affecting the archi design, derived from system spec & analysis model, I/f design - communication of s/w within itself, with system, with human etc, implies flow of info & type of behavior, DFD & control spec form basis, Component-level design transform structural elements into a procedural description of compo, info comes from class-based model, flow models& behavioral model



~~/classinfo domain of class mode rn class realization & required structural of s/warchi style & & analysi & type of behaviornent~~Pspec~~class-based model~~Cspec~~flow models,& b l state transition dia~~

DESIGN MODEL

Decision during design affect success of s/w development & ease with which s/w maintained, during design quality is fostered, design can be assessed for quality translation of customer's requirements into s/w product, foundation for all SE, without design unstable product, fail if small change, difficult to test, quality inaccessible

9.2 DESIGN PROCESS & DESIGN QUALITY

An iterative process, requirements translated in s/w, initially design at a high abstraction i.e. system level which can be traced to detailed data, functional & behavior requirements, with the iterations, design moves to lower abstraction level

During design quality assessed with FTR, three signs to guide a good design (1) design must implement all explicit requirements, accommodate implicit requirements (2) readable, understandable for those who generate code, test & support s/w (3) provide complete picture of s/w, address data, functional & behavioral domains

Quality Guidelines: To evaluate quality of design, use following guide lines :

1. Design should exhibit archi structure that(1) created using recognizable design pattern (2) contains compo exhibiting good design characteristics (3) facilitates implementation & testing
2. Modular, logically partitioned that perform specific functions
3. Contain distinct representation of data, archi, i/f & compo
4. Lead to data structures appropriate for objects to be implemented
5. Lead to compo having independent functional characteristic
6. Lead to i/f that reduce complexity of connection b/w modules & external env
7. Derived using repeatable methods, driven by s/w requirements
8. Notations that is meaningful

Using these, good design achieved

Quality Attributes : HP developed a set of s/w quality attributes

Functionality- capabilities of program, functions that are delivered & security of overall system

Usability- considers human factors such as aesthetics, consistency & documentation

Reliability- evaluated by measuring frequency & severity of failure, o/p accuracy, MTTF, recovery from failure & predictability of program

Performance- measured by processing speed, response time, resource consumption, throughput & efficiency

Formatted: Indent: Left: -0.56", Hanging: 0.38", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.13" + Tab after: 0.43" + Indent at: 0.43", Tab stops: -0.19", List tab + Not at 0.25" + 0.43"

Formatted: Bullets and Numbering

Supportability- ability to extend program, adaptability, serviceability i.e. maintainability, testability, compatibility, configurability, ease of installation, localization of problem etc

Some application may stress on functionality, especially security, other put emphasis on processing speed, some may focus on reliability

9.3 DESIGN CONCEPTS :
Fundamental design concepts provide foundation for sophisticated designs, address following

9.3.1 Abstraction : for modular solutions, many level of abstraction, highest level - solution stated in broad terms using language of problem env, lower level – more procedural detail, language problem oriented plus implementation oriented, lowest level – solution stated in manner directly implemented
at each level of abs - procedural & data abstracts created, procedural abs – sequence of instructions having limited & specific function ex open a door, open has long sequence of procedure – walk to door, hold know, turn & pull door, step away etc, data abs – named collection of data describing data objects, ex door, encompass attributes of door – type, swing direction, weight etc, procedural abs use info contained in attributes of data abs
Programming lang allows creation of abstract data type – Ada, C++, third abs is control abs, implies program control mechanism

9.3.2 S/w Architecture : Overall structure of s/w & its integrity, archi is structure or organization of program components, their interaction manner & data structure, hierarchy of program components & the manner in which they interact & structure of data used, Properties of architectural design are : Structural properties – defines components & their interaction, Extra-functional properties – how to achieve performance, capabilities security etc, Families of related systems – repeatable design, common to similar system, reuse

No of models, Structural model – archi as organized as collection of components, Framework model – level of design abstraction, repeatable archi design framework, Dynamic model – address behavior of program archi, Process model – focus on design of business or technical process, Functional models – functional hierarchy

9.3.3 Patterns : Design pattern describes design structure that solve particular design problem within specific context, provides description that enables to check (1) if

pattern is applicable to current work (2) if pattern is reusable (3) if it can serve as a guide for developing similar pattern.

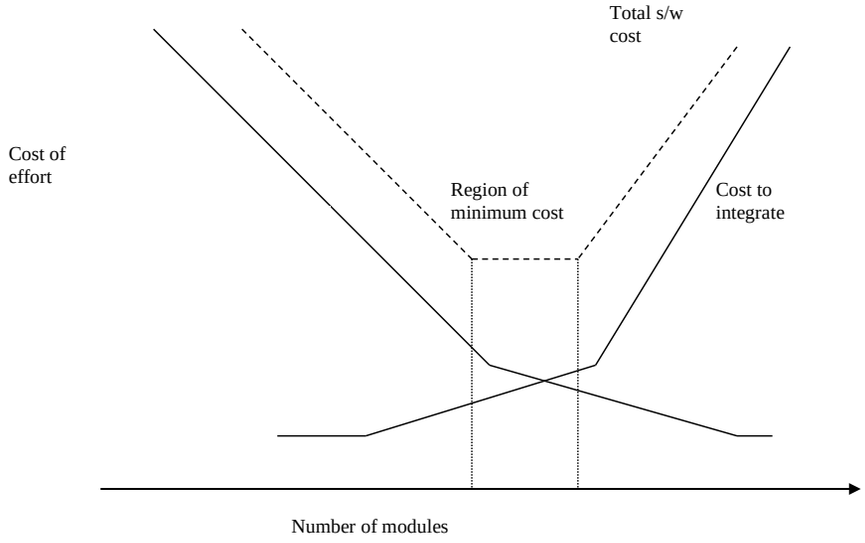
9.3.4 Modularity : S/w divided in separately named components, integrated large program in a single module not easy to grasp, increase in no of paths, span of references, variables & complexity, C(x) – defines complexity of problem, E(x) – efforts to solve problem, for two problem P1& P2, if C(p1) > C(p2) then E(P1) > E(P2) -------(1)

C(P1 + P2) > C(P1) + C(P2) -------(2)

i.e. combine complexity of P1 & P2 is more than sum of complexity of each, so

E(P1 + P2) > E(P1) + E(P2) -------(3)

Easy to solve complex problem by breaking into manageable pieces, if divide indefinitely effort required will be negligible, but other factors come in picture



Effort & cost reduces with increase in total no of modules, but effort & cost increases for their integration, unable to decide region of minimal cost, how much modular s/w, an effective modular design can be defined by criteria
Modular Decomposability : if design method allows decomposition of problem in sub problem, will reduce complexity
Modular Composability : if design method allows assembly of reusable components into new system, easier to build
Modular Understandability : Module understood as standalone unit, easy to build
Modular Continuity : Change in system requirements lead to change in individual module rather than system change, minimum change impact
Modular Protection : effect of errors contained within module, minimum side effects

Under-modularity or over-modularity avoided, Certain systems cannot spare minimal speed or memory overheads used by sub programs ex real-time & embedded s/w, s/w designed with modularity but developed in-line

9.3.5 Information Hiding : Decision made within models hide from all others, information contained within module inaccessible to other modules, hiding & effective modularity by defining independent modules, necessary communication, defines access constraints within module, benefit are easy to test & maintain

9.3.6 Functional Independence : direct effect of modularity, abstraction & info hiding, modules with single minded function & avoiding excessive interaction b/w modules, independent modules easy to develop because of compartmentalization & simple i/f, easy to maintain- secondary effects of modifications limited, reduces error propagation, reusable modules, a key to good design & hence good quality, measured using cohesion – relative functional strength of a module & coupling – relative interdependence of modules

9.3.7 Refinement : A top-down design strategy, A hierarchy is developed by decomposing functions in stepwise fashion until programming language statement, process of elaboration starts with statement of function, no info about internal working or structure, elaboration of original statement, abs & refinement complementary to each other

9.3.8 Refactoring : A reorganization technique that simplifies design of a component without changing its function or behavior, improves internal structure, eliminates - redundancy, unused elements, inefficient or unnecessary algo, inappropriate data structure & other design failure, ex, a component having low cohesion with three functions, designer refactors the component in three separate components resulting high cohesion, easy testing, integration & maintenance

9.3.9 Design Classes : analysis model defines a set of analysis classes, describes elements of problem domain, converted into design classes that (1) refine analysis classes by providing design detail to be implemented (2) create new set of design classes that implement s/w infrastructure that support business solution, 5 diff types of design classes representing diff layer of design archi

- User i/f classes defines abstractions for human-computer interaction visual representation
- Business domain classes refinement of analysis classes, identifies attributes & services required to implement business domain
- Process classes implements lower-level business abstraction that manages business domain classes
- Persistent classes represents data stores that exist o/s s/w
- System classes implements s/w management & control functions for system operation & communication

Level of abstraction reduces with each iteration, design class should be well-formed, 4 characteristics of well-formed design classes –

Formatted: Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Bullets and Numbering

Complete & Sufficient : complete encapsulation of attributes & methods for ex a class scene should contain all the attributes & methods associated with creation of video scene for video editing s/w, no more no less

Primitiveness : methods in a design class should focus on one service of class

High Cohesion : small focused set of responsibilities & single-mindedly apply attributes & methods

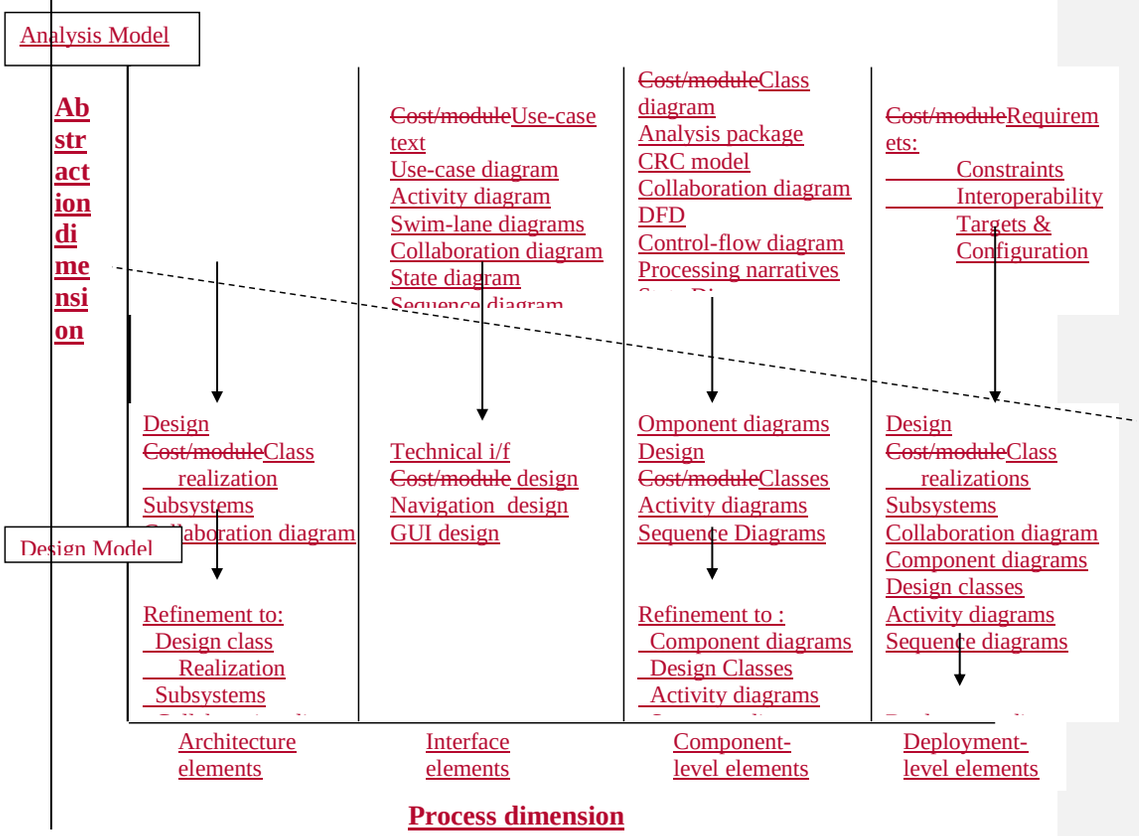
Low coupling : minimum collaboration, highly coupled system difficult to implement,

9.4 THE DESIGN MODEL

Design model viewed in 2 dimensions (1) process dimension indicates evolution of design (2) abstraction dimension represents detail of design models, many design

model use same UML as analysis model but in more detail, archi structure, style, compo residing within archi, i/f are emphasized, developed not in sequential manner, but may be in parallel

~~Fig~~



9.4.1 DATA DESIGN ELEMENTS

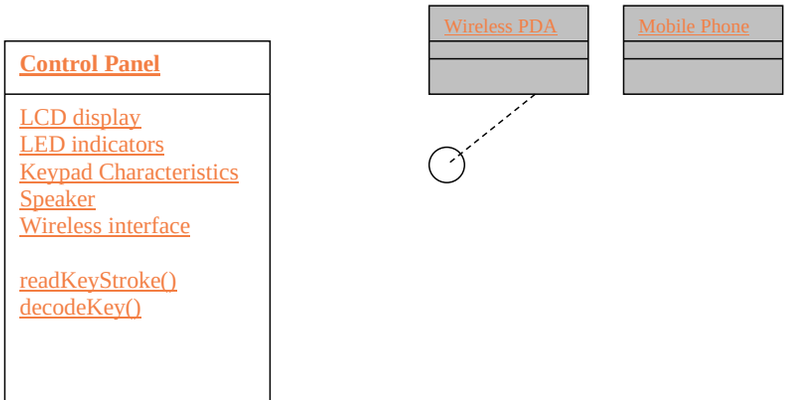
~~data~~Data/info present at high level of abstraction are refined to implementation ~~specifi~~specific representation, archi of data influence archi of s/w, at program compo level – design of DS & associated algo essential for high quality s/w
Structure of data an imp part of s/w design, at component level – design of DS & associated algos to manipulate them essential for high quality, at application level – data model translated into data base to achieve business objectives, at business level – collection of info stored in separate databases reorganized as data warehouse, enables data mining & knowledge discovery that impact on business

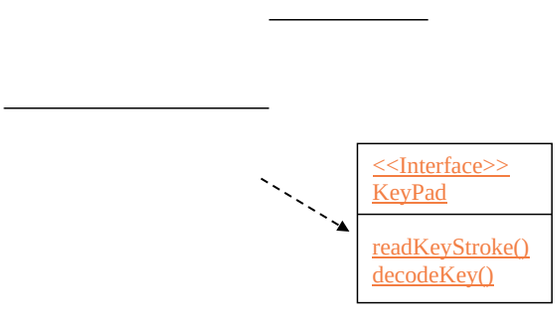
9.4.2 Architectural Design Elements

Gives overall view of s/w, derived from (1) info about application domain of s/w (2) analysis model elements such as DFD, analysis classes etc & their relationship & collaboration (3) availability of archi patterns & styles

9.4.3 Interface Design Elements

describes how info flows into & out of system & communication b/w components, 3 imp elements (1) user i/f (2) external i/f to other system, devices, n/w etc (3) internal i/f b/w components
Design of user i/f incorporates aesthetic elements such as layout, color etc, ergonomics elements such as information layout & placement, navigation etc, & technical elements such as reusable components etc
Design of external i/f require definite info about the entity to which it communicates, incorporate error checking & security features
Design of internal i/f enables communication & collaboration b/w operations in various classes, UML defines i/f as "An i/f is a specifier for the externally visible operations of a class, component or other classifier without specifying internal structure, for ex, safe home security functions make use of control panel that allows owner to control security function, may be implemented via wireless PDA or Mobile phone,

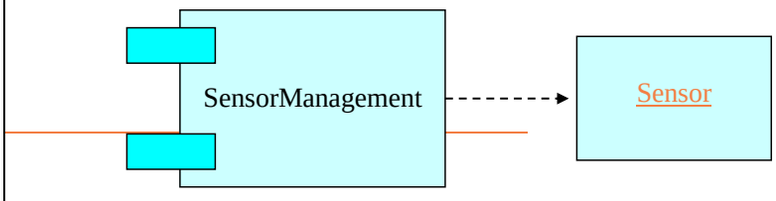




Control Panel class provide behavior associated with a keypad therefore provide functions readKeyStroke() & decodeKey(), i/f KeyPad class has no attributes, indicates that KeyPad operation is part of ControlPanel, this is known as realization in UML

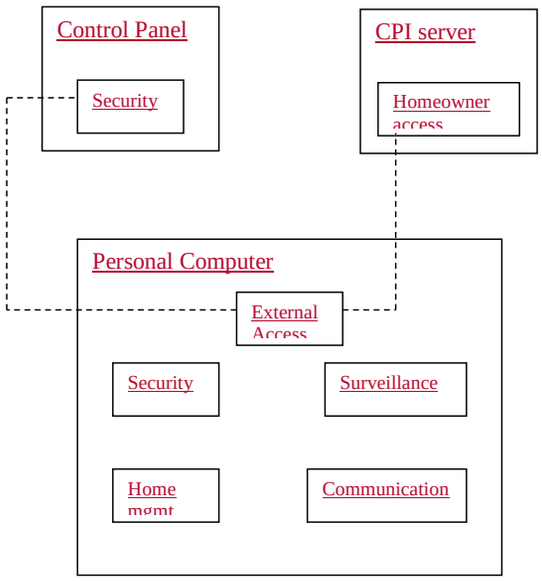
9.4.4 Component-level Design Elements

Fully describes internal detail of each s/w component, defines DS for all local data objects, algo detail for processing & i/f to other components, using UML a component is represented diagrammatically, ex SensorManagement, this component is connected to class Sensor, this compo performs functions associated with safehome sensors from monitoring to configuring



9.4.5 Deployment-Level Design Elements

Indicates how s/w functionality & subsystems will be allocated within physical computing env that support s/w. for ex safehome s/w configured to operate within 3 computing env – a home-based PC, safehome control panel & a server providing internet access, UML deployment diagram as in fig, 3 computing env & subsystem within each are shown, ex PC houses subsystem of security, surveillance, home management & communication, external access subsystem manages external source's attempt to access safehome system, diagram shows just the desription not the detail



Patterns combined with problem to create a solution

9.5.1 Describing a Design Pattern- Engineers use design patterns for engineering, attributes are inherent to patterns, ex mechanical engineers use two-step keyed shaft pattern with the attributes – diameter of shaft, dimension of keyway etc, & operations – shaft rotation, connection etc, pattern describe env & conditions that must exist to make pattern applicable, pattern characteristics indicates design attributes that may be adjusted to accommodate variety of problem, these attributes give characteristics of design for which appropriate pattern must be found
Names of design pattern chosen with care, one of the problems with s/w reuse is inability to find reusable pattern from thousands of candidate patterns, this can be resolved using meaningful name

9.5.2 Using Patterns in Design

Pattern can be used throughout s/w design, after analysis designer examines detailed representation of problem & constraints, problem description is examined to check if any pattern can be applied

Architectural Patterns. These patterns define overall structure of s/w, indicates relationship among subsystems & s/w components, defines rules for specifying relationship b/w elements of archi

Design Patterns. Address a specific element of design such as aggregation of compos, relationships b/w compos & mechanism for communication

Idioms. Also coding pattern, language-specific, implement algorithmic elements of a component, i/f protocol or mechanism for communication

9.5.3 Frameworks

it may be necessary to provide implementation-specific skeletal infrastructure – framework for design – a reusable mini-archi to provide generic structure & behavior, framework is not a archi pattern but a skeleton with many plug points to be adapted to specific problem domain

Designing is a multistep process - data structure, program structure, interface characteristics & procedural details to be synthesized –

10.1 S/W ARCHITECTURE :

Most imp in SE, many different attributes for architecture, for a building – shape, manner in which various component of building integrated, way the building fits in

env, degree to which meets its purpose & satisfy needs, aesthetic feel of structure – texture, color, material, small designs – lighting, flooring, it is art

10.1.1 What Is Architecture
S/w architecture – a representation that enable to (1) analyze effectiveness of design in meeting its requirements (2) consider archi alternatives (3) reducing risk associated with s/w development
Role of s/w component – simple program or database & middleware that configure n/w, s/w archi includes data design & architectural design

10.1.2 Why is Archi Imp :
key reasons are : (1) enable for communication b/w all interested in development (2) highlights design decisions having impact on all SE works & success of system (3) constitute small, graspable model of how system is structured & components work together

10.2 DATA DESIGN : creates a model of data & info at a high level of abstraction, then refined to more implementation specific representation, have influence on archi & s/w, at component level design of DS & associated algo affects quality, at application level data model translated into database, at business level collection of info stored in database & reorganized into data warehouse, data mining

10.2.1 Dtat Design At architectural Level :
Data objects of requirements analysis modeled using ER diagram & data dictionary, Design translate data model into DS at component level, database archi at application level, previously data archi was DS at program level & database at application level, today moderate size business have dozens of databases, extract useful info from this data, developed data mining techniques, navigate through databases to extract appropriate info, data mining difficult because of multiple databases, different structure, alternate solution is data warehouse, adds additional layer to data archi, is a separate env not directly integrated with regular applications, encompass all data of business, a large independent database, characteristics of data warehouse are :
Subject orientation : organized by major business subjects, excludes data not necessary for data mining
Integration : regardless of source, data have consistent naming convention, units, measures, encoding structure, attributes

Time variancy : for transaction-oriented appli – data accurate at the access & for short time span, for data warehouse, data accessed at a specific time, time span is 5-10 years
Nonvolatibility : business application data undergo continuous change, warehouse data do not change

10.2.2 Data Design at Component Level :
Focus on representation of DS directly accessed by programs, a set of principles to specify & design data

1. The systematic analysis principles applied to function & behavior should also be applied to data : Much effort in deriving, specifying functiona requirement & preliminary design, data flow & content developed & reviewed data objects identified, alternates considered, impact on s/w design evaluated ex linked list
2. All data structure & operation to be performed on each should be identified : operation to be performed must taken into account
3. Data dictionary should be established & used to define both data & program structure : data dictionary represents relationship b/w data objects & constraints on elements
4. Low-level data design should be deferred until late in the design process : overall data organization during requirement analysis, refined during preliminary design & specified in detail during component level design – top-down approach
5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure : concept of info hiding & coupling
6. A library of useful data structures & the operations that may be applied to them should be developed : reusability of DS
7. A s/w design & programming language should support the specification & realization of abstract data types : implementation of DS difficult if language chosen for development do not support

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

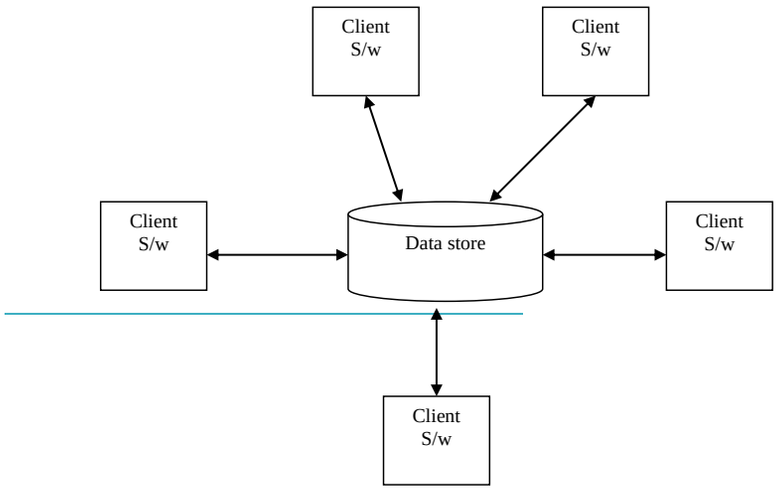
Formatted: Bullets and Numbering

10.3 ARCHITECTURAL STYLES & PATTERNS:

S/w exhibits one of many archi styles, each has (1) set of components that perform function (2) set of connectors to communication, coordinate & cooperation of components (3) constraints defining how components integrated to form a system (4) semantic properties that describe overall properties of system, common styles are

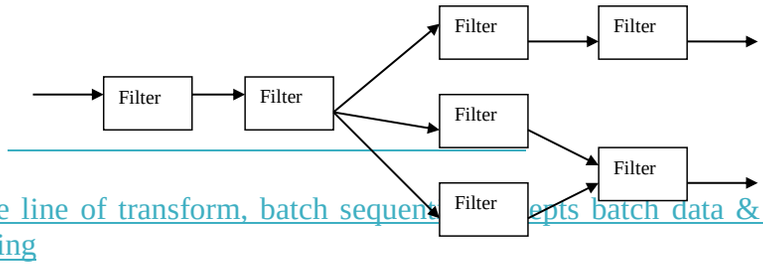
10.3.1 a Brief Taxonomy of Architectural Styles

Data Centered Archi : data store at the center, accessed & modified by components, client-server technology



Client s/w access central repository, sometimes passive i.e. client access data independent of changes, a notification is sent to client when data of interest change, blackboard, promote integrability – existing component can be changed, new component added without affecting other clients, through blackboard transform of info b/w clients

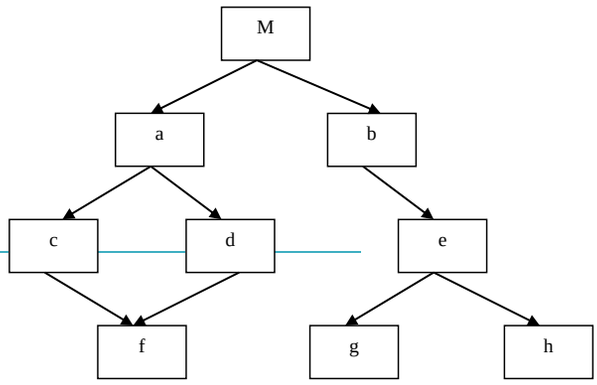
Data-Flow Archi : when i/p data transformed thro series of processing into o/p, pipe & filter pattern, pipes transmit data, filters work independently, expect i/p of certain form & produce specific o/p



If single line of transform, batch sequent[illegible]epts batch data & apply series of processing

Call & Return Architecture : program structure easy to modify, a number of sub styles

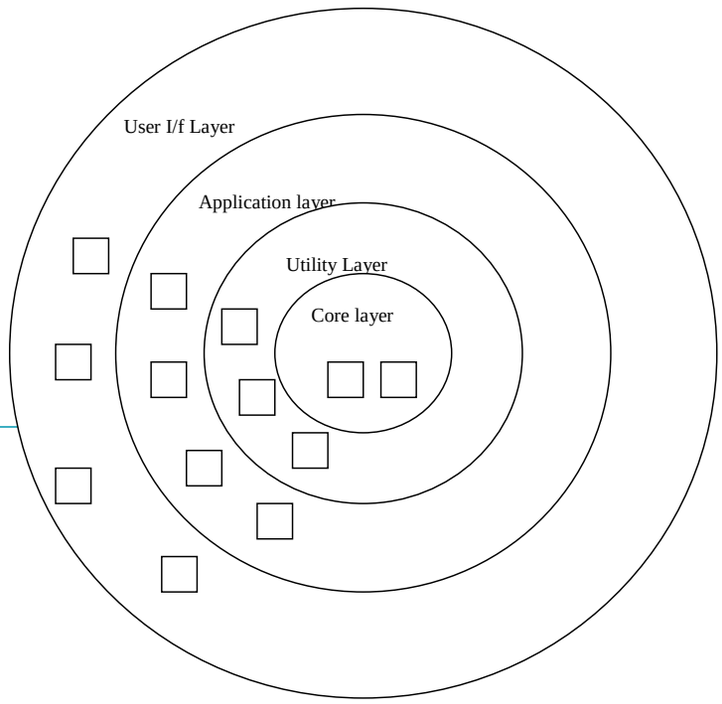
- Main Program/Subprogram Architecture : decomposed function in control hierarchy, main program invokes no of subprogram which again invoke other programs



- Remote Procedure Call Architecture : components distributed across multiple computers on n/w

Object-Oriented Architecture : encapsulation of data & operation on them, communication & coordination via message passing

Layered Architecture : number of diff layers, performing operations, progressively closer to machine instruction, outer layer user i/f operations, inner layer perform OS i/f, intermediate layer provide utility & application s/w functions



Formatted: Indent: Left: -0.5", First line: 0", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

Once requirements identifies characteristics & constraint of the system, archi style or combination best fit is chosen

10.3.2 Architectural Patterns

Archi patterns for s/w define a specific approach fro handling some behavioral characteristics of system, Some archi patterns are
Concurrency. Handle multiple tasks in a manner that simulates parallelism, no of ways to handle concurrency, each with diff archi pattern, one approach is use operating system process mgmt pattern, provides built-in OS features that allow components to execute concurrently, manages communication b/w processes, scheduling etc, another approach to define task scheduler at application level, contains a set of active objects

Persistence : Data persists if it survives past the execution of process, persisted data stored in database or file & later modified by other processes, values of attributes of objects, general state & other info stored for future retrieval, 2 archi pattern used to achieve persistence – dbms applies storage & retrieval & application level persistent pattern to build persistent feature
Distribution : the manner in which system or components of system communicate in a distributed env, 2 elements (1) the way in which entities connect with each other (2) nature of the communication. Most common archi pattern applied is broker pattern, broker acts as a middle-man b/w client & server compos, client sends message to broker, broker completes connection ex CORBA contains broker archi.

10.3.3
No of alternatives for archi design, a set of design criteria to assess designs, for insight of archi style
Control : how control managed, control hierarchy, role of components in it, control topology, synchronization etc must be looked
Data : data communication b/w components, flow of data – continuous or sporadic, mode of data transfer, how functions interact with data etc must be looked

For assessing design quality

10.4 ARCHITECTURAL DESIGN

Archi design should define external entities & nature of interaction, designer specifies structure by defining & refining components, iterative process

10.4.1 Representing the System in Context

Architectural context diagram used model s/w interaction with external entities, systems that interact with target systems are

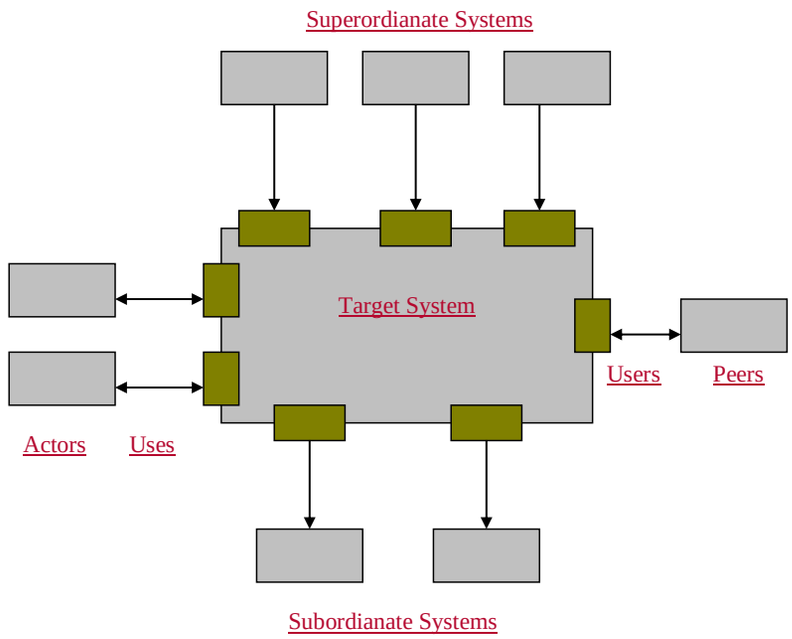
Super-ordinate systems – systems that use target system as part of higher level processing

Sub-ordinate systems – systems used by the target system & provide data or processing necessary to complete target system functionality

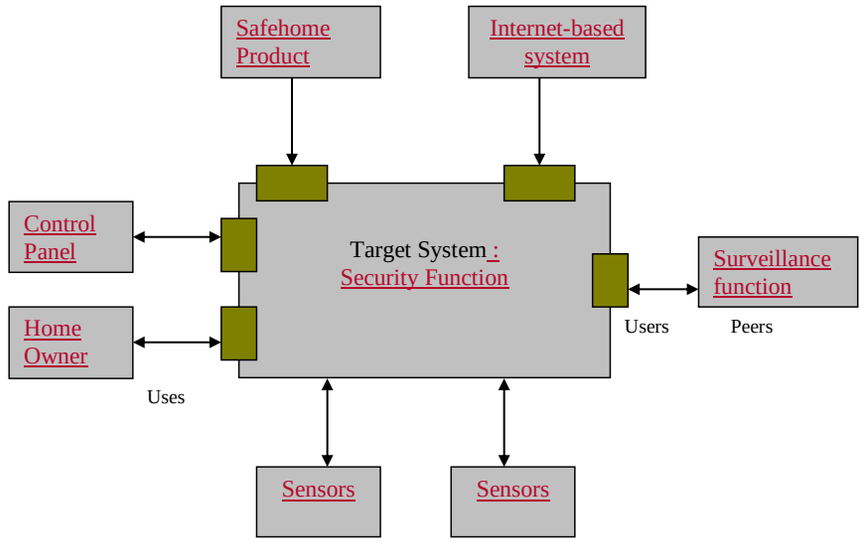
Peer-level systems- systems that interact on a peer-to-peer bases

Actors- entities (people, devices) that interact with target system by producing or consuming info

Each external entity communicates with s/w through i/f,



Ex Safehome security system using ACT,

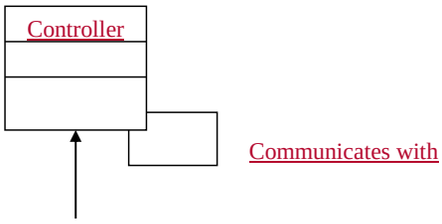


10.4.2 Defining Archetypes

An archetype is a class or pattern that represents a core abstraction for the target system, a small set of archetypes required to design even complex systems, target system archi represented using archetypes, represents stable elements of archi depending on behavior of system, for safehome security function, following archetypes can be defined

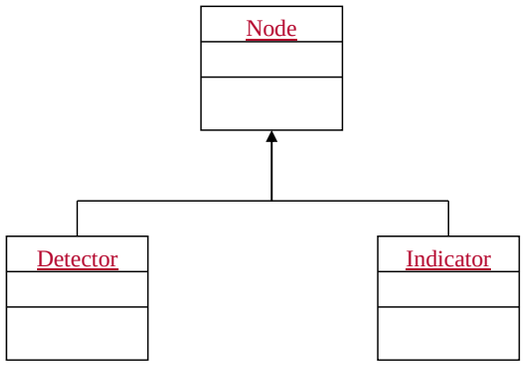
- *Node*. Represents collection of i/p & o/p elements of home security function ex various sensors, alarm indicators etc
- *Detector*. An abstraction that encompass all sensing equip that feed info to target system
- *Indicator*. Represents all mechanisms like alarm siren, bell, flashing lights for indicating alarm condition
- *Controller*. Depicts mechanism that allows arming & disarming of a node, if controller on n/w, ability of communication

Using UML notations each of these archetypes are depicted as



Formatted: Indent: Hanging: 0.88", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering



10.4.3 Refining the Architecture into Components

As archi refined into compo, structure of the system emerges, begins with the classes of analysis model, represents entities within business domain that must be addressed in archi, another source is infrastructure domain, many infrastructure components such as memory mgmt compo, communication compo accommodated & integrated in archi, a complete subsystem archi with components designed, with safehome security example, a set of top-level compos that address following functionality

- External Communication mgmt – Coordinates communication of security function with external entities such as internet-based system
- Control panel processing – manages control panel functionality
- Detector mgmt – coordinates access to all detectors
- Alarm processing – verifies & acts all alarm conditions

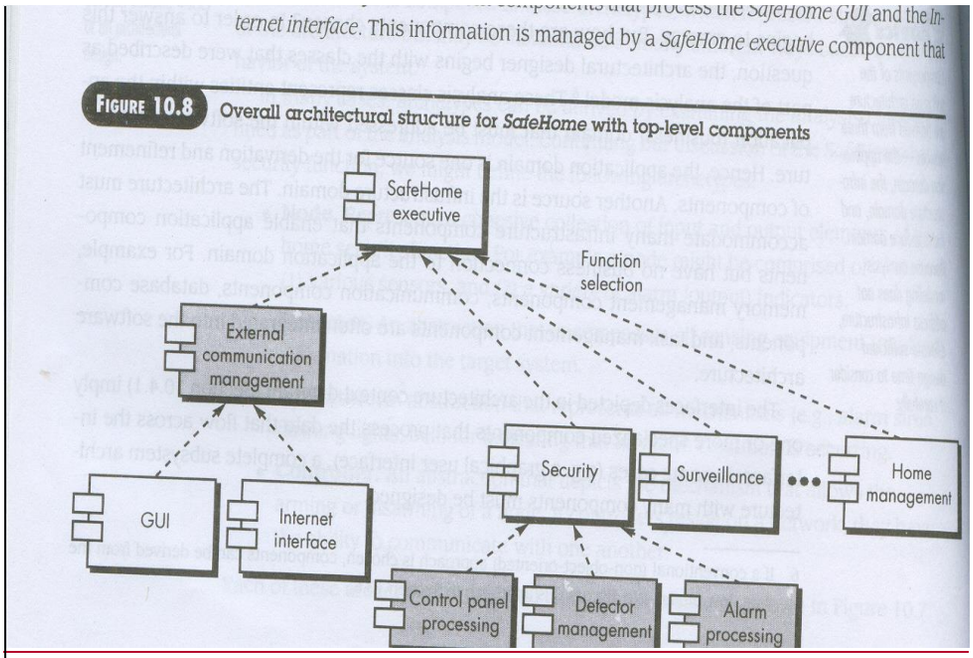
Each top-level compo elaborated iteratively & then positioned within safehome archi, design detail of all attributes not defined till component-level design,

Formatted: Indent: Left: -0.5", Hanging: 0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.06", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

ternet interface. This information is managed by a *SafeHome executive* component that

FIGURE 10.8 Overall architectural structure for *SafeHome* with top-level components



~~Fig 10.8~~

10.4.4 Designing Instantiations of the System

Still at high level of abstraction, further refinement still required, actual instantiation of archi is developed i.e. archi is applied to specific problem with intent of demonstrating appropriateness of structure & components

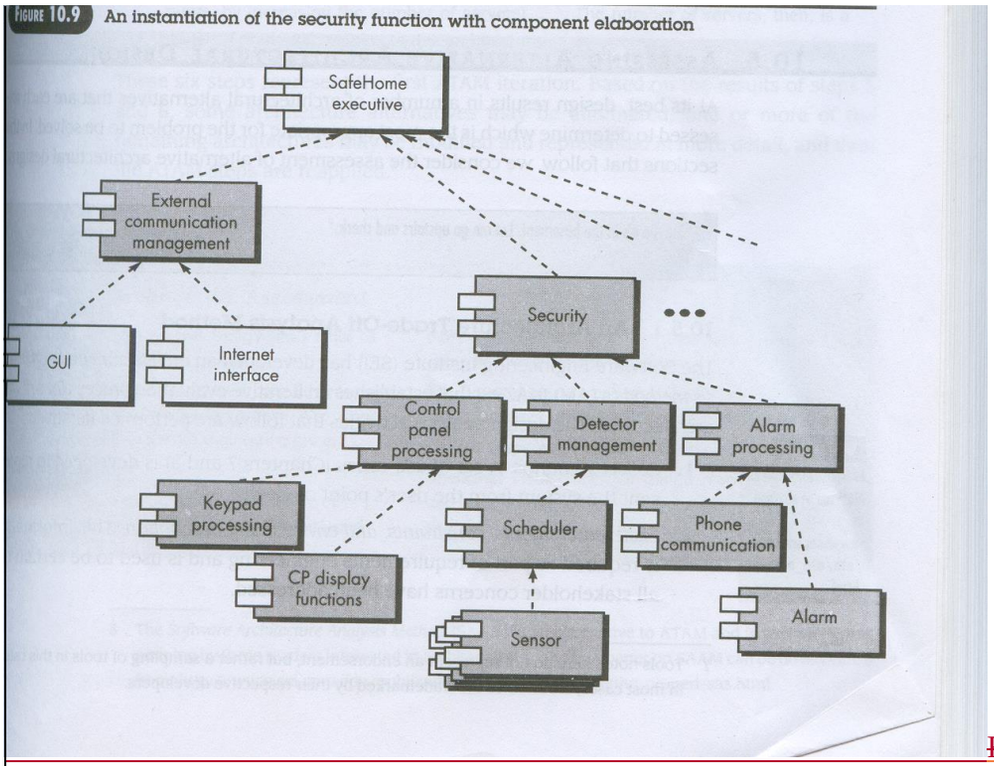


FIGURE 10.9 An instantiation of the security function with component elaboration

~~Fig 10.9~~

10.5 ANALYZING ALTERNATIVE ARCHITECTURAL DESIGN :

Two approaches to analyze alternate archi designs, first iterative method, second applies pseudo-quantitative technique

10.5.1 An Architecture Trade-off Analysis Method :

Developed by SEI, iterati~~ons are~~ve evaluation process, steps are

1. Collect scenario : set of use-cases developed to represent the system from user's view point
2. Elicit requirements, constraints & environment description : As part of SE make certain that all customer, user concerns addressed
3. Describe the archi style/pattern that have been chosen to address the scenario & requirements : should describe archi views such as

- Module view : to analyze components & info hiding achieved

Formatted: Indent: Left: -0.38", Hanging: 0.38", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

- Process view : for system performance
- Data flow view : degree to which archi meets functional requirements

4. Evaluate quality attributes by considering each attribute in isolation : include reliability, performance, security, maintainability, flexibility, testability, portability, reusability & interoperability
5. Identify the sensitivity of quality attributes to various archi attributes for a specific archi style : by making a small change in archi & determine how sensitive quality attribute to the change, attributes significantly affected by variation are sensitivity points
6. Critique candidate archi using sensitivity analysis : identify elements to which multiple attributes are sensitive

In step 5 & 6, some alternatives eliminated, remaining archi modified & presented in detail, again apply above steps

~~Quantitative Guidance for Architectural Design : unavailability of quantitative methods for assessing quality of design, a no of models to assist design in determining which archi meets goodness criteria, also known as design dimension, encompass quality attributes~~

10.5.2 Architectural Complexity :

F~~f~~or assessing overall complexity of archi, dependencies b/w components is considered, ~~simple metrics used~~3 types of dependencies

1. Sharing Dependency. Dependent consumers who use same resource or producers who produce for same consumer, 2 compos u & v refer to same global data then sharing dependency
2. Flow dependency. Dependency b/w producers & consumers of resources, 2 compos u & v, u must complete before control flows in v or u communicates with v by parameter that flow dependency
3. Constrained dependency. Constraints on flow of control, 2 compos u & v, if u & v are mutually exclusive then constrained dependency

Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: -0.5" + Tab after: -0.25" + Indent at: -0.25"

Formatted: Bullets and Numbering

10.5.3 Architectural Description Language

UML notations available for s/w archi, other diagrammatic tools available, ADL provides semantics & syntax for describing s/w archi, ADL should provide to decompose archi compos, compose individual compo I larger archi & represent i/f b/w compos, UML includes artifacts for archi description but not complete

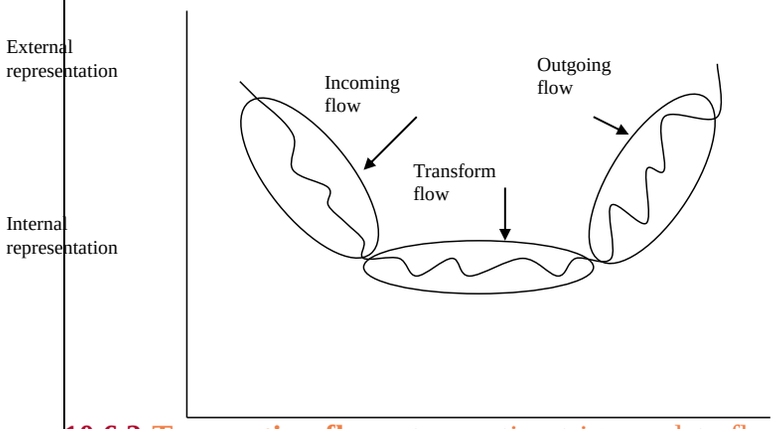
10.6 MAPPING ~~REQUIREMENTS~~DATA FLOW INTO A S/W ARCHITECTURE :

S/w requirements mapped into design model, no practical mapping for certain design, on ad-hoc manner, ex call & return archi – maping technique allows to derive complex archi from DFD, technique known as structured design – stress on modularity, top-down design & structured programming, provides convenient transition from DFD to s/w archi, the transition is six-step process (1) the type of info flow is established (2) flow boundaries are indicated (3) the DFD is mapped into program structure (4) control hierarchy is defined (5) resultant structure refined using design measures & heuristics (6) Archi refined & elaborated

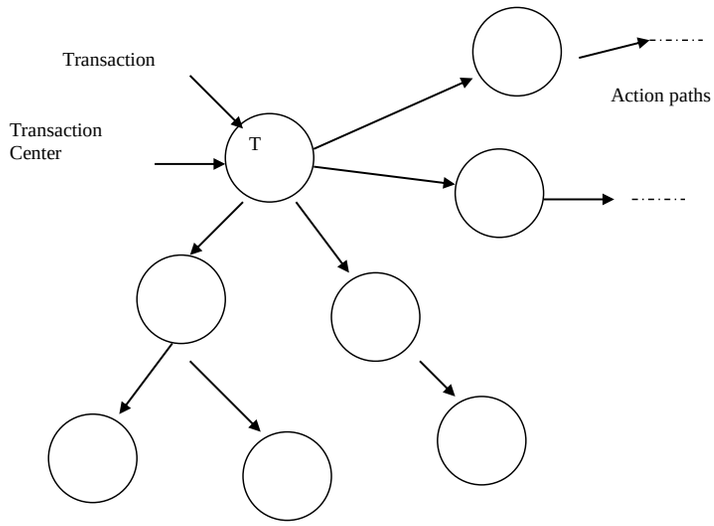
There are two types of info flow

~~Transform Flow : info enters & exits s/w in external world form ex. Data typed on keyboard, pictures, tone on telephone line etc – entered data are converted into internal form for processing - paths that convert external data to internal data is incoming flow, at a kernel transition occurs, incoming data pass thro transform center, move towards outgoing flow, overall flow of data o~~

10.6.1 Transform Flow : info enters & exits s/w in external world form ex. Data typed on keyboard, pictures, tone on telephone line etc – entered data are converted into internal form for processing - paths that convert external data to internal data is incoming flow, at a kernel transition occurs, incoming data pass thro transform center, move towards outgoing flow, overall flow of data occurs in sequential manner

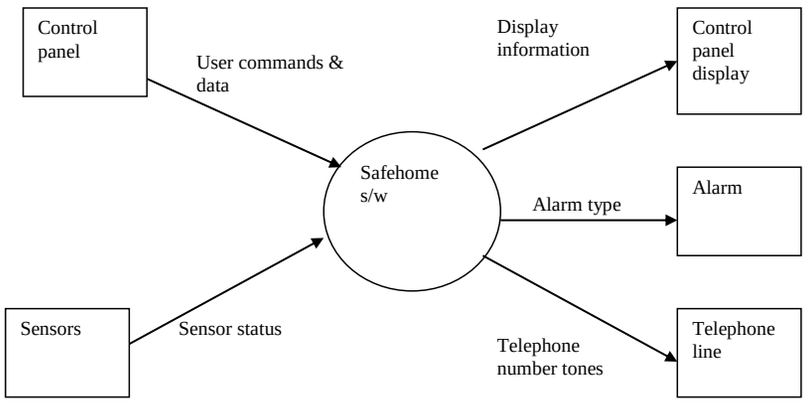


10.6.2 Transaction flow : transaction triggers data flow along one of many paths, when DFD is as shown in figure then it is a transaction flow- data comes along incoming path that converts external world info into transaction – based on its value flow along one of many action paths is initiated the hub is called transaction center.



In a DFD for a large system, transform & transaction flow both might be present.

10.6.3 TRANSFORM MAPPING : DFD with transform flow mapped into specific archi style – ex. Safehouse security s/w



Safehome

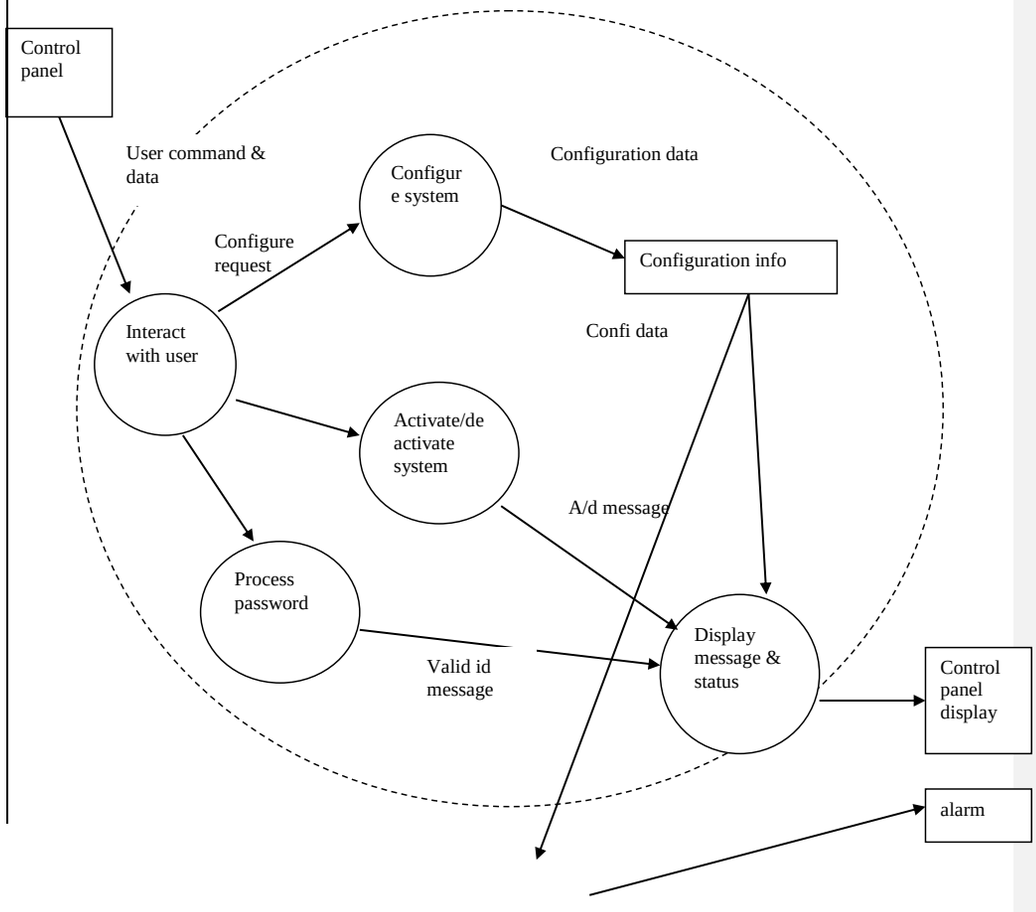
s/w monitors sensors & reacts to changes, interacts with user through typed commands – 0 level DFD as above

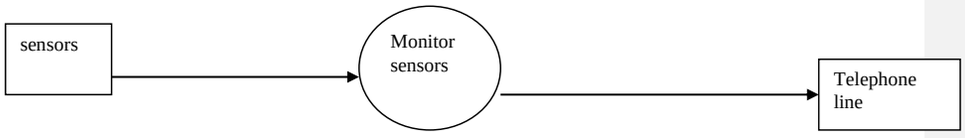
Design Step : begins with re-evaluation of requirement analysis

1. *Review the fundamental system model* : context diagram depicts security function as a single transformation, representing external producers & consumers of data Level 1 DFD

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

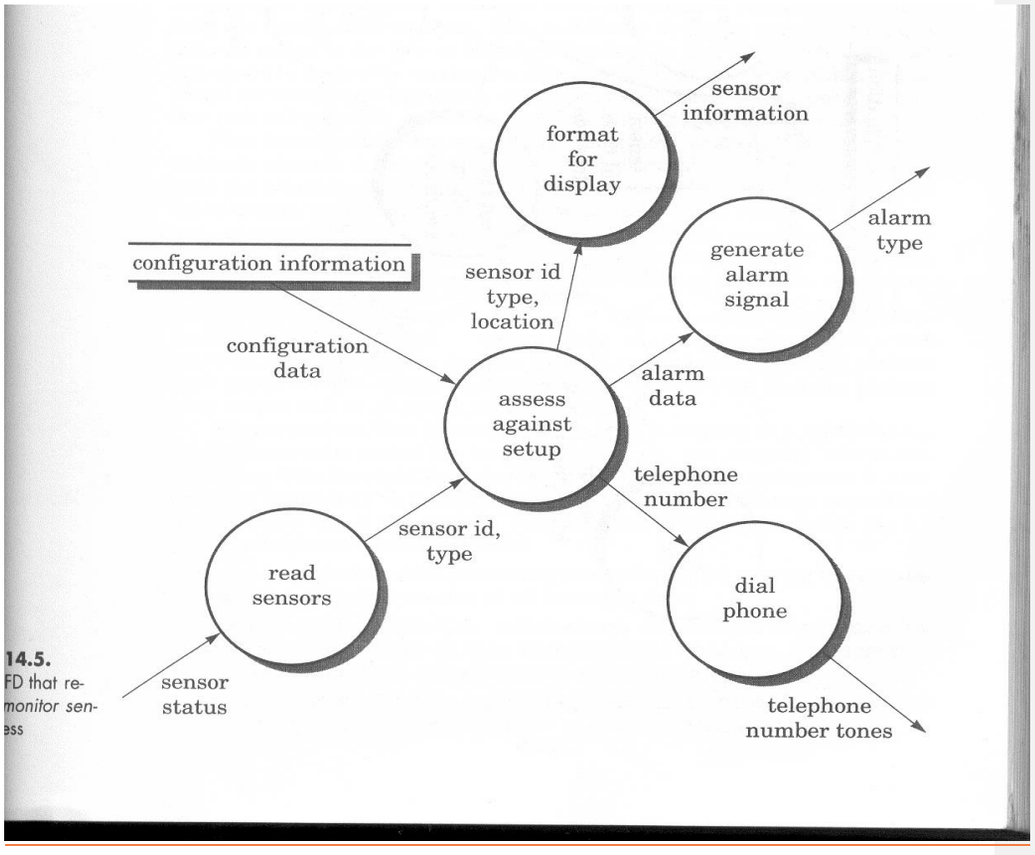




2. *Review & refine data flow diagrams for the s/w* : info of requirements spec refine to greater detail, level 2 DFD & level 3 DFD, each transform exhibit high cohesion, perform single function, can be implemented as module, provide for first-cut design, ex monitor sensors transform

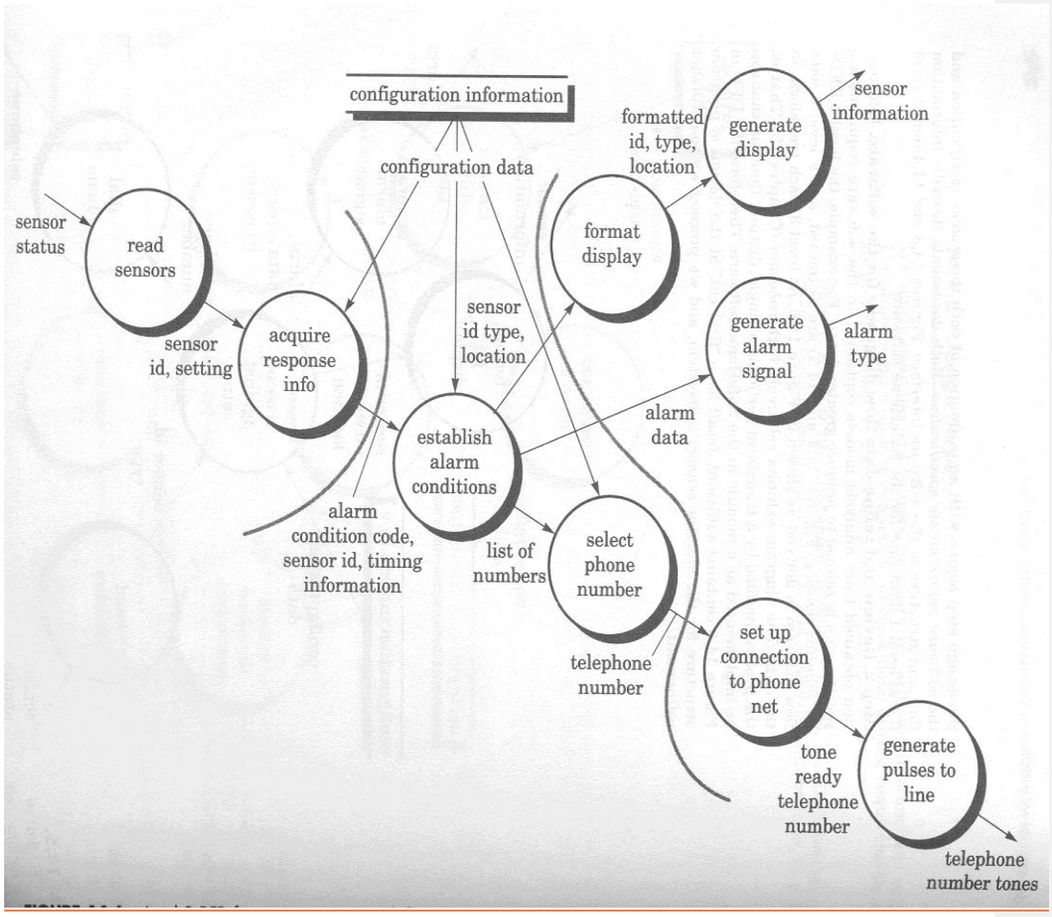
Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering



14.5.
FD that re-
monitor sen-
ess

Level 2 DFD that refines the monitor sensors peocess



Level 3 DFD for monitor sensors with flow boundries

3. *Determine whether DFD has transform or transaction flow characteristics* : info flow within a system can be represented as transform, but when obvious transaction flow- a different design mapping required – in above fig data enters on one incoming path & exit along 3 outgoing path – no distinct transaction center – assumed transform characteristic
4. *Isolate the transform center by specifying incoming & outgoing flow boundaries* : incoming flow is path where info converted from external to internal form & outgoing flow where internal to external form, boundaries according to interpretation, different flow boundaries – ex curved boundaries in above figure,

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

transform bubbles lie within the boundaries – emphasis on selecting reasonable boundaries

5. *Perform first-level factoring* : archi derived results in top-down distribution of control – factoring divides program structure in three – top level modules perform decision making – middle level performs some control & moderate work – low level performs most i/p, computation & o/p – in transform mapping, DFD is mapped to specific structure that provide control for incoming, transform & outgoing processes – first level factoring as shown in fig – main controller at top, serve to coordinate, subordinate control functions are:

- incoming info processor controller – sensor i/p controller, coordinates receipt o incoming data
- transform flow controller – alarm conditions controller, supervise operations on data
- outgoing info processing controller – alarm o/p controller, coordinates o/p info

No of modules at first level limited to minimum

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

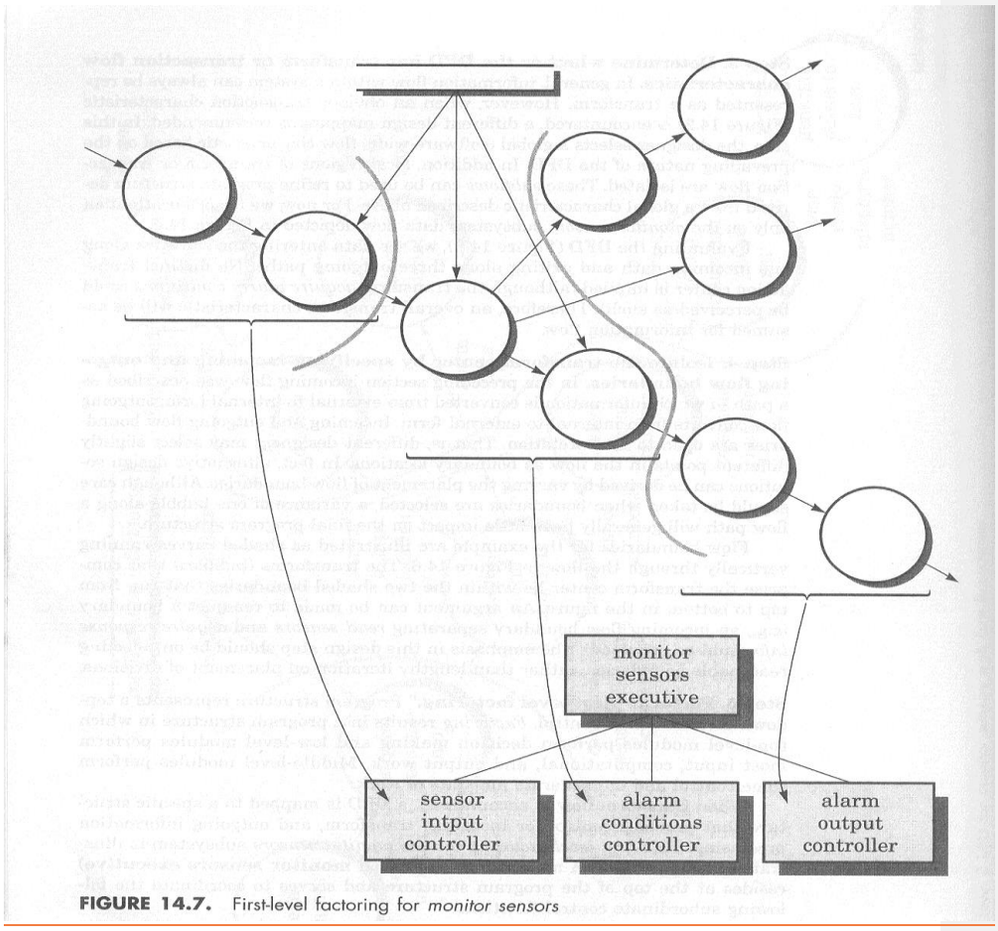


FIGURE 14.7. First-level factoring for *monitor sensors*

6. *Perform second level factoring* : map individual transforms into appropriate module – begin at transform boundary move outward on incoming & outgoing paths –

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

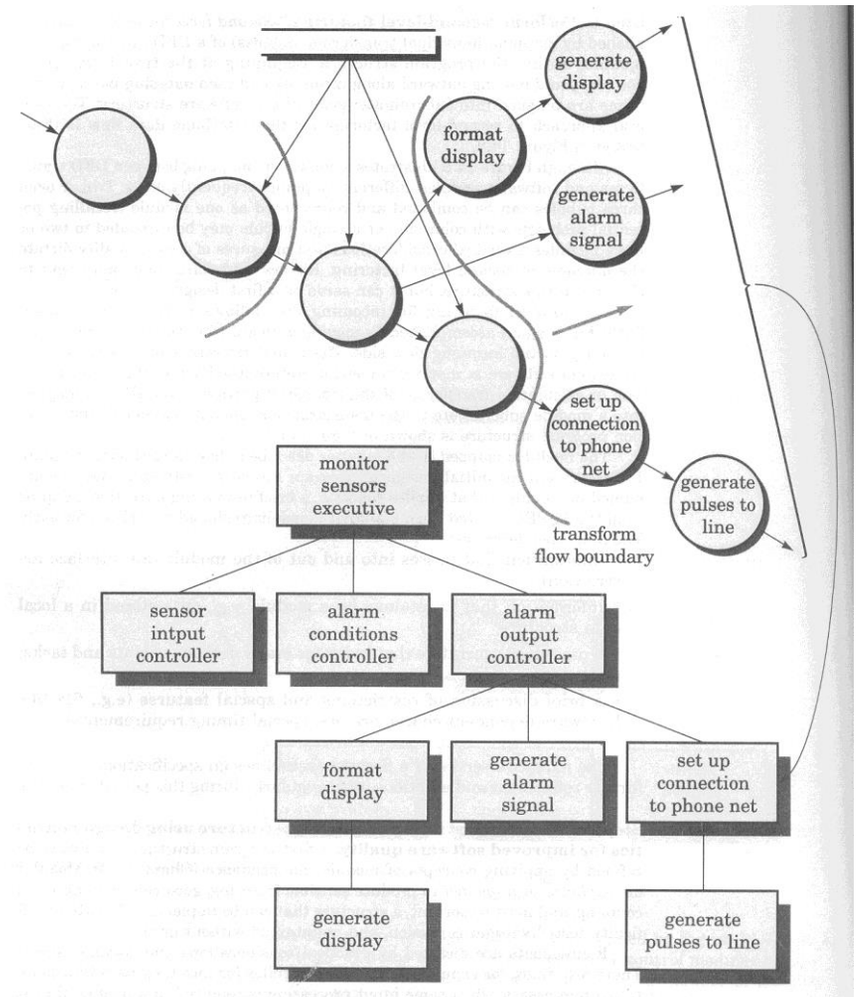


FIGURE 14.8. First level factoring for *monitor sensors*

fig shows one-to-one mapping of DFD & module – different approach – two or three bubbles combine to represent one module – single bubble expanded to two or more modules – depends on practical consideration & design quality – transform center is mapped little differently – for monitor sensor subsystem, each transform bubble is mapped as a subordinate bubble of transform controller – a brief processing narrative should be written for each module such as information coming in & out, retained in module, major decision points & tasks etc – serve as design spec

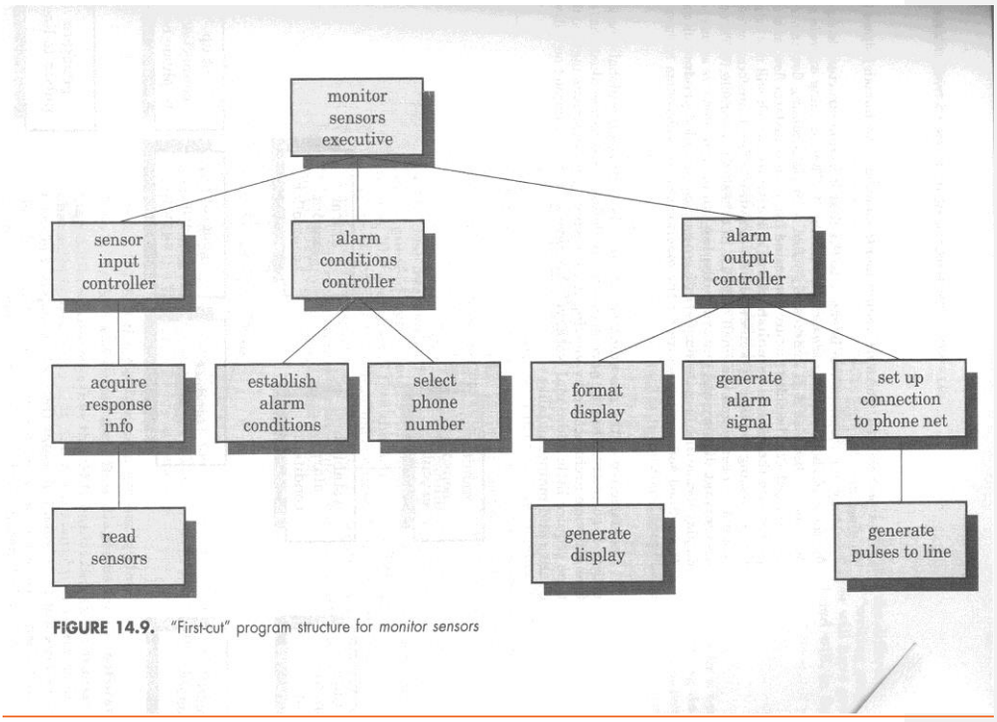


FIGURE 14.9. "First-cut" program structure for *monitor sensors*

7. *Refine the first iteration architecture using design heuristics for improved s/w quality* : apply module independence – explode or implode modules for sensible factoring, good cohesion, minimum coupling, for a structure that can be implemented without difficulty, testable & maintainable – in our ex incoming controller is removed, it was unnecessary – final structure will be

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

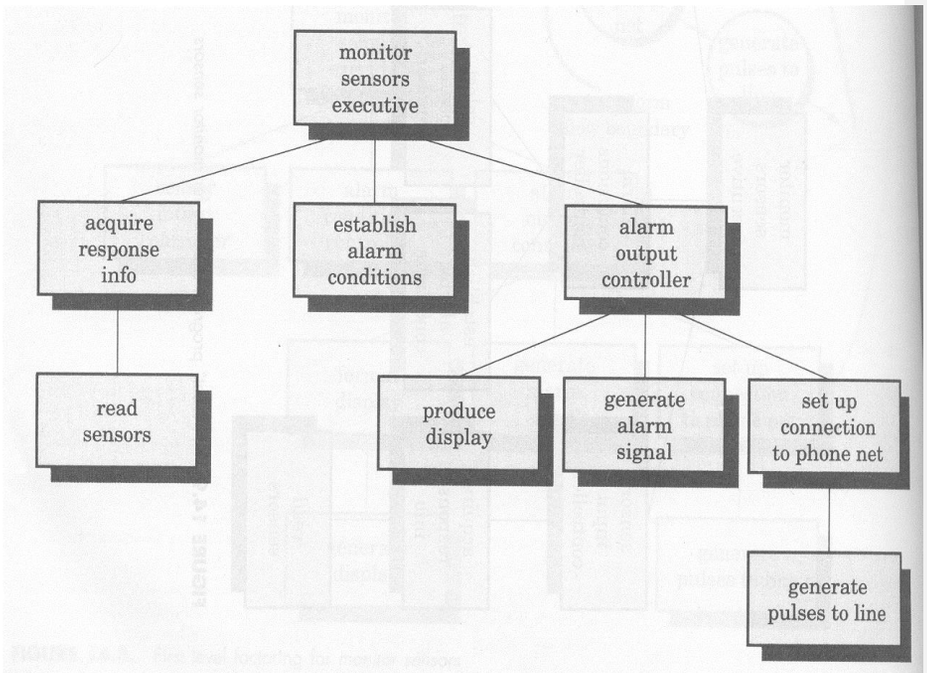


FIGURE 14.10. Refined program structure for *monitor sensors*

Thus a representation of s/w is created

10.6.4 TRANSACTION MAPPING : a single data item may trigger one of many information paths –
Our ex will be user interaction subsystem of safehome s/w - user command flows into the system – results in flow along one of three paths, level 2 DFD is as shown

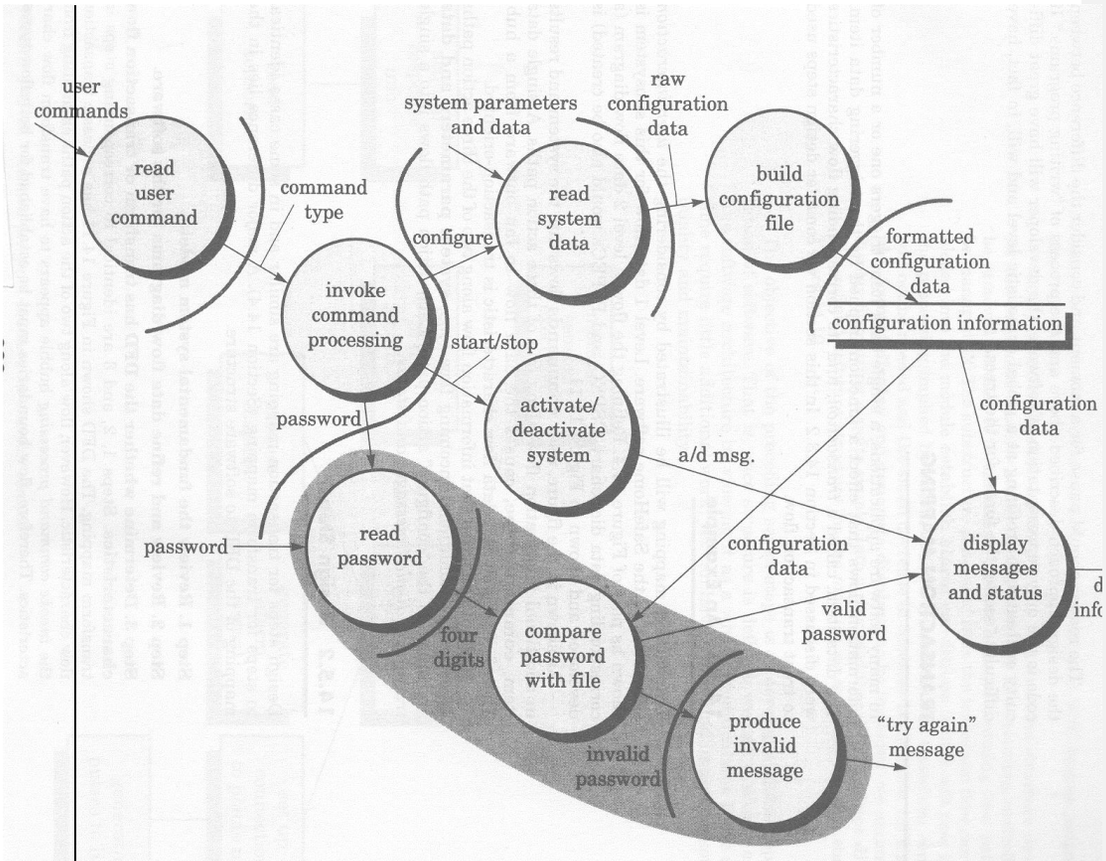
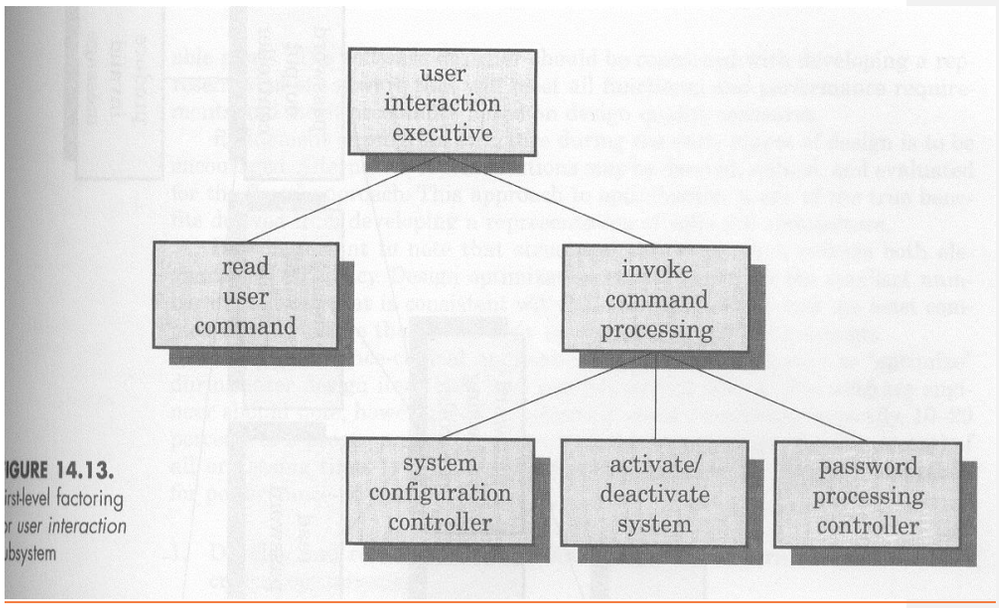


FIGURE 14.11. Level 2 DFD for *user interaction* subsystem with flow boundaries

Design steps : similar to transform mapping

1. *Review fundamental system model*
2. *Review & refine data flow diagrams for the s/w*
3. *Determine if it has transform or transaction flow* : in fig invoke command processing has transform flow – flow boundaries for both flows
4. *Identify transaction center & the flow characteristics along each action path* : can be found from DFD – at the origin of no of action path – each action path evaluated for its individual flow characteristics –
5. *Map the DFD in a program structure amenable to transaction processing* : mapping into a programs structure containing incoming branch & dispatch branch, incoming branch as transform mapping, dispatch branch contains dispatcher modules that controls subordinate modules – each action flow path mapped – ex user interaction first level factoring as shown in fig



IGURE 14.13.
irst-level factoring
r user interaction
ubsystem

6. *Factor & refine the transaction structure & structures of each action path* : each action path has its own flow characteristics – password processing (shaded part) exhibit transform characteristics – resultant archi will be as shown in fig

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

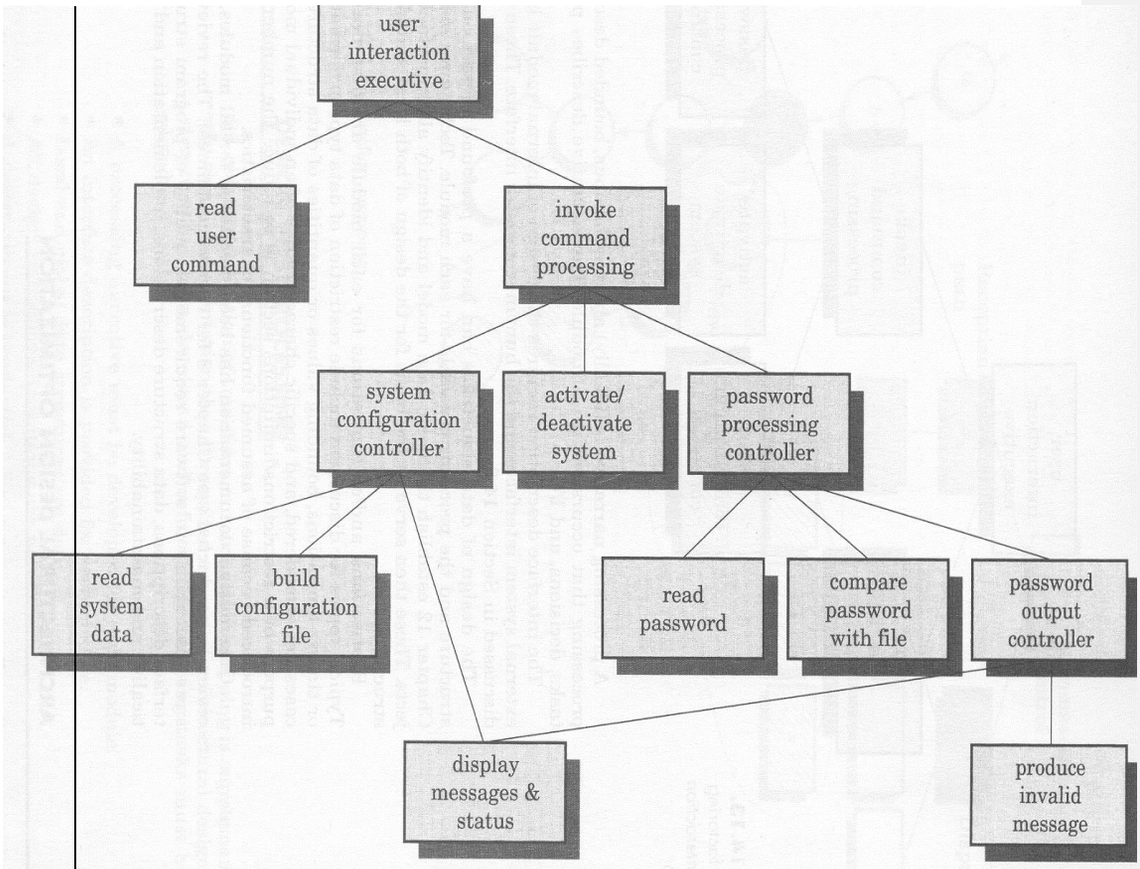


FIGURE 14.14. First-cut program structure for *user interaction* subsystem

7. *Refine the first iteration architecture using design heuristics for improved s/w quality* : criteria such as module independence, practicality etc considered

10.6.5 REFINING THE ARCHITECTURAL DESIGN :

Supplementary documents as part of archi design, after structure has been developed following tasks must be completed

(1) processing narrative must be developed for each module – unambiguous bounded description, describes processing tasks, i/o, decisions

(2) interface description provided for each module – design of internal module i/f external system i/f, human-computer i/f

(3) local & global data structure are defined – impact on archi & procedural detail

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.5", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.75" + Indent at: 0.75", Tab stops: -0.25", List tab + Not at 0.25" + 0.75"

Formatted: Bullets and Numbering

(4) design restrictions/ limitations noted – data type/format, memory, timing, purpose is to reduce no of errors introduced by assumed characteristics
(5) conduct design review – for quality & maintainability
(6) refinement is considered – for optimization, structural simplicity, less no of modules with effective modularity

Develop a s/w that meet all functional & performance requirements & have acceptable quality – refinement of program structure – alternative approach – structure simplicity reflects elegance & efficiency – design optimization – smaller no of modules for effective modularity, less complex data structure that fulfill the requirement – for performance critical s/w find those small process that are most critical – use case tools

11. COMPONENT-LEVEL DESIGN

Occurs after archi design , high level of abstraction, program at low level of abstraction, conversion introduce bugs, a design guidelines

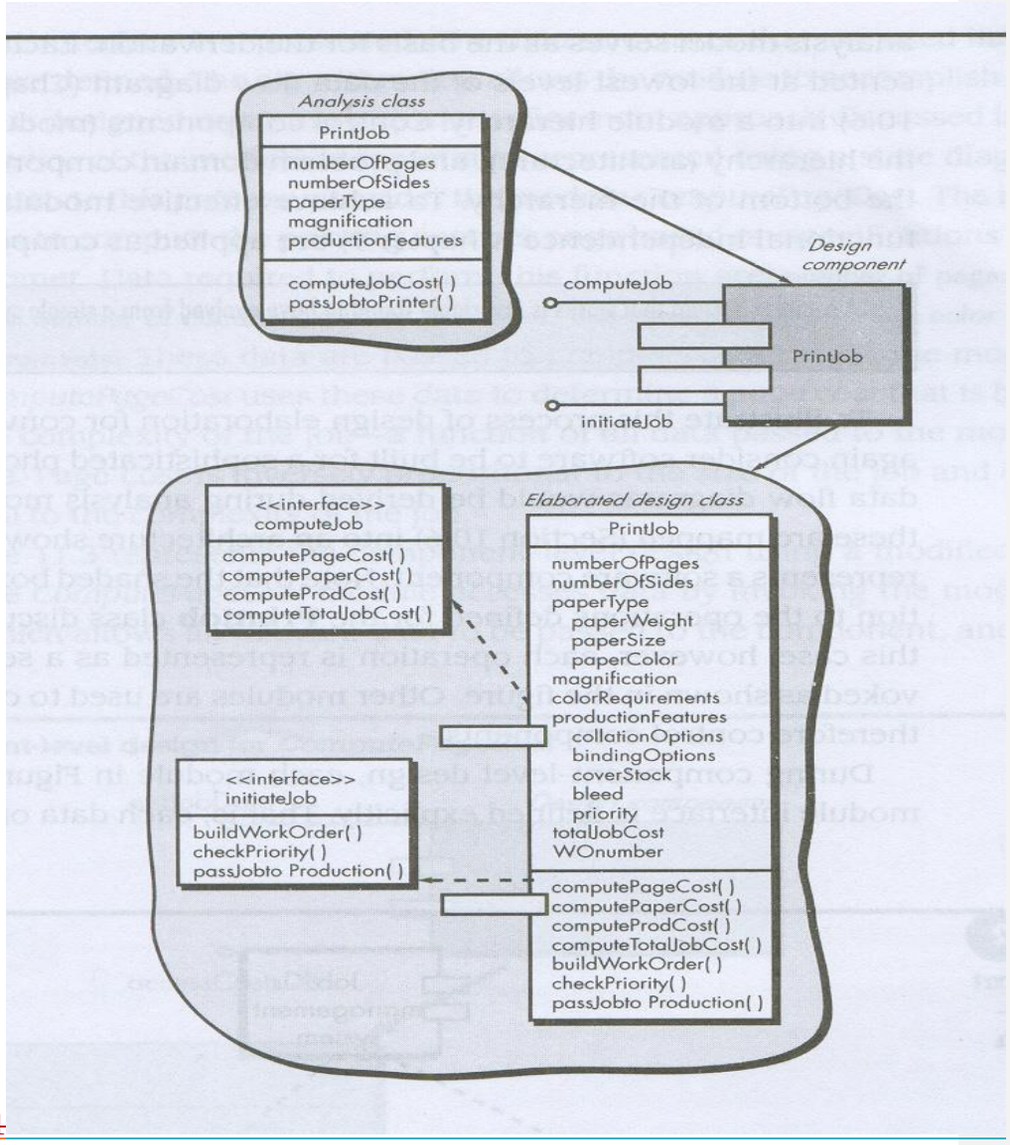
11.1 WHAT IS A COMPONENT?

Compo is a modular building block for computer s/w, UML definition – a modular, deployable & replaceable part ofs a system that encapsulates implementation & exposes a set of i/f, compos play imp role in achieving objectives & requirements of system, communication with other compos & external entities, 3 imp views representing what a component is & how it is used in design model

11.1.1 An Object-Oriented View

In OO approach, a compo contains a set of classes, each class includes all attributes & operations necessary for its implementation, i/f with other components also described, designer start by elaborating analysis class & infrastructure class, ex. A sophisticated printing shop system – s/w collects customer's requirements, cost a print job & pass job to an automated production facility, during analysis a class called **printjob** derived, its attributes & operations defined , during archi design **printjob** defined as component within s/w archi, presented using UML, **printjob**

~~fig~~



~~11.1~~

11.1.2 The Conventional View

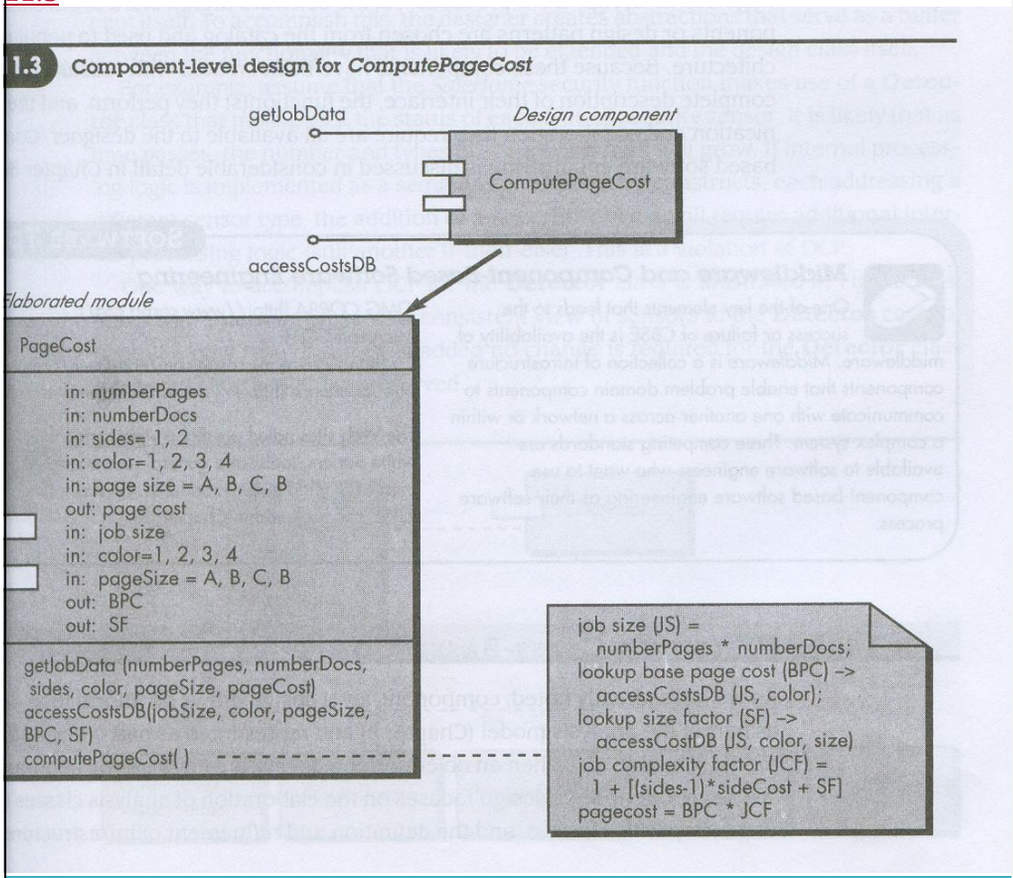
In conventional view, a compo is a functional element of a program that incorporates processing logic, internal data structures to implement processing logic, i/f to invoke compos & data passing. Called a module, reside within s/w archi, 3 imp roles (1) a control component – coordinates invocation of all other problem domain compo (2) problem domain compo- implements function required by the customer (3) infrastructure compo – responsible for functions that supports processing of problem domain

Data flow-oriented elements of analysis model serves to derive compos, each transform at the lowest level of DFD mapped in module hierarchy, control compo reside at the top of hierarchy, problem domain compo reside at bottom, design concepts applied

Ex printing shop, DFD is mapped into archi structure as in fig, each box represent s/w compo, shaded boxes perform actual function, others are control function, during compo-level design, each module elaborated, i/f defined explicitely i.e. each data object that flow across i/f is presented, DS defined, algo that provide functionality are designed, behavior of module presented using state diagram

Consider computepagecost module, computes printing cost per page based on specifications, data required are : no of pages, total copies to be produces, one or two side printing, color scheme, size etc, these data are passed to computepagecost via i/f, function determines cost of the page, fig represent component-level design using UML, module accesses data by invoking module getjobdata, relevant data is passes to module~~a~~ database i/f accesscostsDB enables to access database, module elaborated to provide algo & i/f details, algo with pseudocode or UML activity diagram, i/f represented by collection of i/p & o/p data items

Fig 11.3



11.1.3 A Process_Related View

previous views assume that compo designed from scratch, a new compo is created, another approach uses existing s/w compos, a catalog of proven design or code-level compos, with the development of s/w archi compos or design patterns chosen from catalog& used to populate archi, reusable compos have a complete description of i/f, functions they perform, required communications & collaborations

When an OO approach is chosen, compo-level design focus on elaboration of analysis classes & definition & refinement of infrastructure classes, detailed description of attributes, operations & i/f details required for designing

11.2.1 Basic Design Principles

4 basic design principles applied to compo-level design for OO SE, create designs that are amenable to change & less error propagation

The Open-Closed Principle (OCP). A Module should be open for extension but closed for modifications, allows to be extended without making internal coding or logic modifications, create abstraction that serve as buffer b/w functionality to be extended & design class, ex safehome security function use Detector class that checks status of sensors, as time passes no & types of security sensors may increase, if internal processing logic implemented as a sequence of if-then-else construct, addition of new sensor require additional internal processing logic, this is violation of OCP, to achieve OCP, sensor i/f created for Detector compo, new sensor addition requires no change to Detector class

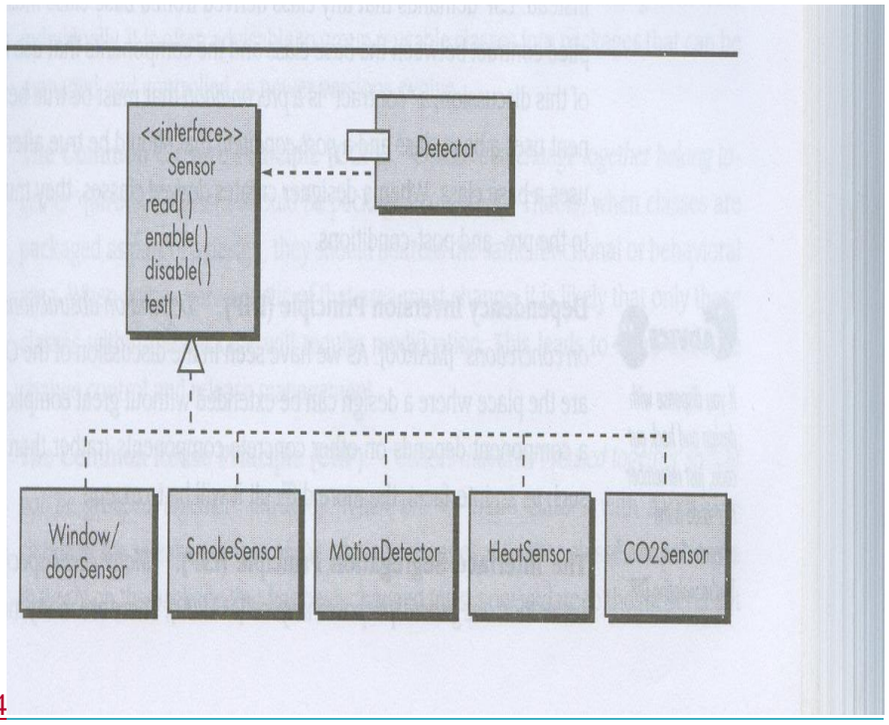


Fig 11.4

The Liskov Substitution Principles (LSP). Subclasses should be substituted for their base classes, a compo using base class should function properly if a class derived

from base class is passed to the compo, Pre-condition & post-condition considerations

Dependency Inversion Principle (DIP). Depend on abstractions, do not depend on concretions, with abstractions, design can be extended without complications, a component that depend on concrete compos- more difficult to abstract

The Interface Segregation Principle (ISP). Many Client-specific i/f are better that one general purpose i/f, multiple client compos use operations of server class, a specialized i/f design to serve each major category of client

Many times compos or classes organized into subsystems or packages, how this packaging is done & components organized in it? Additional principles

The Release Reuse Equivalency Principles. The granules of reuse is the granule of release, when compos for reuse, developer commits to establish release control system that support & maintains older version, reusable classes are grouped for mgmt & control

The Common Closure Principles (CCP). Classes that change together belong together, classes packaged together should address same functional or behavioral area, when some characteristic of that area change, only that particular package to be modified, more effective change control

The Common Reuse Principle (CRP). Classes that aren't reused together should not be grouped together, when one or more classes in a package change, release no of package change, other classes or package relying on changed package should be updated & tested, if classes not grouped cohesively a class having no relation to classes within package is changed, more integration & testing efforts

11.2.2 Component-level Design Guidelines
Guideline for compos, i/f, dependencies & inheritance characteristics

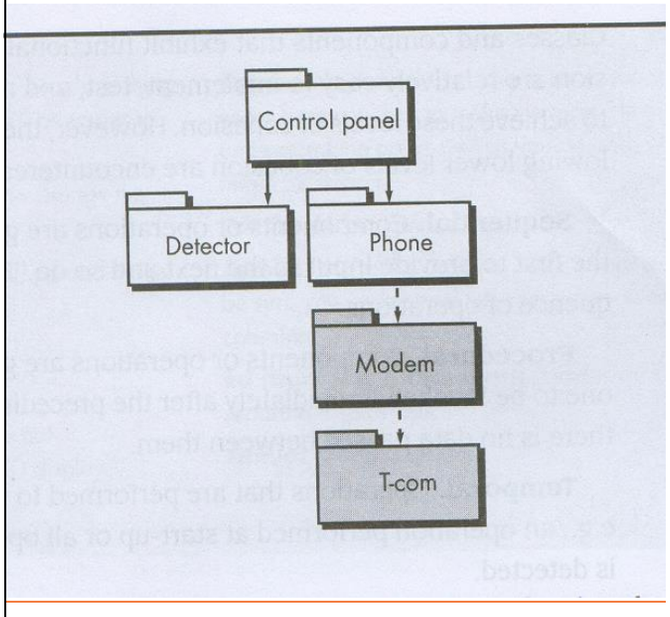
Components. Naming conventions for compos, names drawn from problem domain, meaningful to stackholder, ex class floorplan, infrastructure compos or elaborated compo-level classes named to reflect implementation, ex for linked list mgmt class name managelist() is appropriate, use of stereotype to~~m~~ identify ~~b~~nature of compos <<infrastructure>>, <<database>>, <<table>>

Interfaces. Provides info about communication & collaboration, (1) UML box & dashed arrow used for complex diagrams, (2) for consistency i/f from left side of compo box (3) show those i/f relevant to compo under consideration, simplifies UML comp diagram

Dependencies & Inheritance. For readability, model dependency from left to right& inheritance from bottom to top, compo dependency represented via i/f

11.2.3 Cohesion

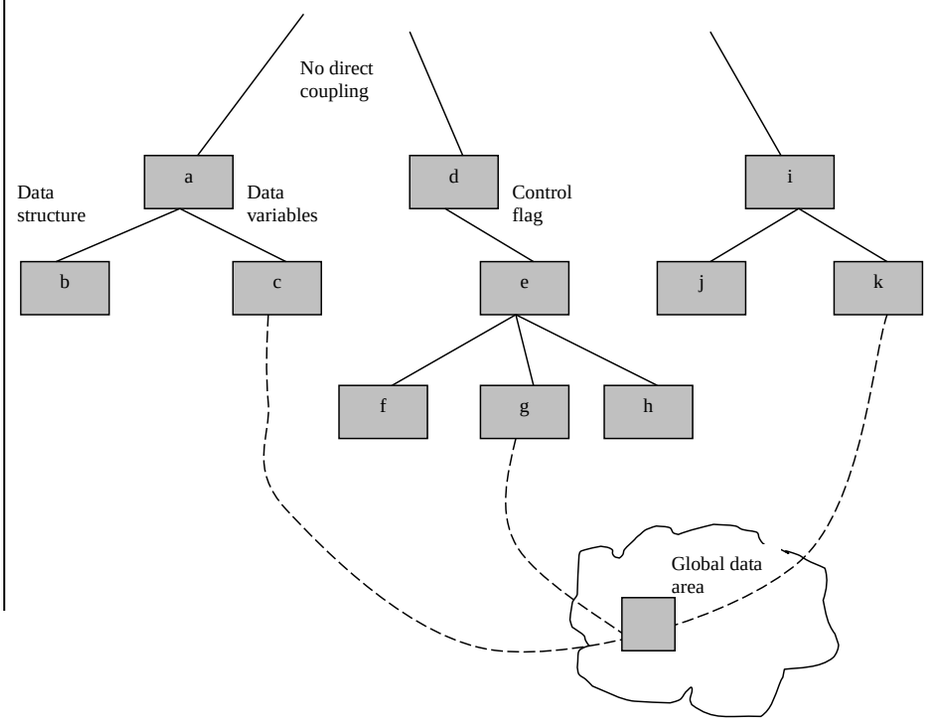
Single mindedness, in context of OO design – cohesion implies a compo or class encapsulate only those attributes & operations that are closely related to one another, thrive for high cohesion, low level cohesion avoided during design
Functional Cohesion- exhibited by operations, when a module perform one & only one computation & returns a result
Layer- Exhibited by packages, compos & classes, occurs when a higher layer accesses services of a lower layer, lower layer has no access to higher layers ex fig 11.5 access is from control panel to downward



Communicational- All operations that access same data are defined within one class, focus on data access & storage

Classes & compos that exhibit functional, layer & commu cohesion easy to implement, test & maintain, high level of cohesion,
Sequential- compos grouped in manner that allow first compo to provide i/p to next & so on, sequence of operations
Procedural-compos are grouped in manner to be invoked immediately after preceding compo invoked, even no data passing
Temporal- Operations performed to reflect a specific behavior or state, tasks that must be executed within same span of time, ex an error processing module performs (1) computes supplementary data based on computed data (2) produce an error report (3) follow-up calculation if requested (4) updates database (5) menu selection for further processing – all tasks functionally independent, best as separate module, likelihood of error propagation
Utility– compos, classes exist within same category but unrelated, class statistics perform 6 unrelated statistical operations

11.2.4 Coupling :
Communication & collaboration imp b/w compos, as amount of communications increases, i/f complexity increases & increase in implementation, testing & maintenance, Coupling is qualitative measure of interconnection of classes, strive for lowest coupling, simple connection result in s/w easy to understand & less error propagation

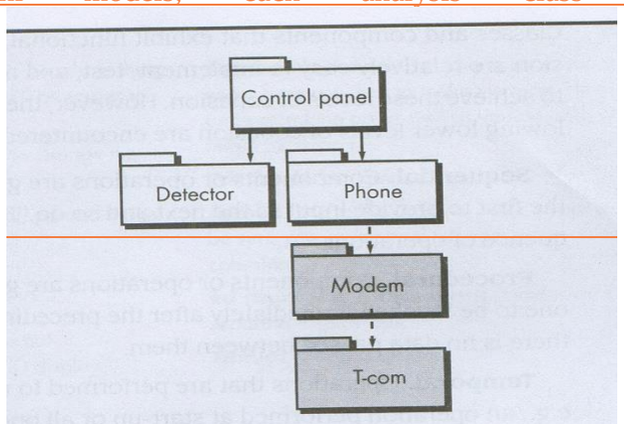


content coupling – highest degree, one module make use of data or control info maintained within boundary of another module, branches made in middle of module, must avoid
common coupling – no of modules reference global data area, modules c, g & k, modules c initialize item, g recomputed & update it, assume error occurs & g updates item incorrectly, k reads item, try to process it but fails, error occurred in k but underlying cause is g which is difficult to find
control coupling – control passes b/e modules, control flag b/w modules d & e, unrelated change in one module needs to change meaning of flag
stamp coupling – occurs when class B declared as type of an argument of an operation of Class A, class B now a part of definition of Class A, modification complex
data coupling –long strings of data arguments passes b/w classes, as bandwidth & compos increase i/f complexity increase
Routine call coupling – when one operation invoke another, common & necessary coupling,
Type use coupling – compo A uses data type defined in compo B, if type definition changes, every compo using that data modified
Inclusion or import coupling – compo A imports or includes a package or content of compo B
external coupling – when compo communicates with infrastructure compo, i/o couples a module to specific device, essential but limited use

11.3 CONDUCTING COMPONENT-LEVEL DESIGN

A task-set for compo-level design for OO system

1. Identify all design classes that correspond to the problem domain. Using analysis & archi models, each analysis class & archi compo

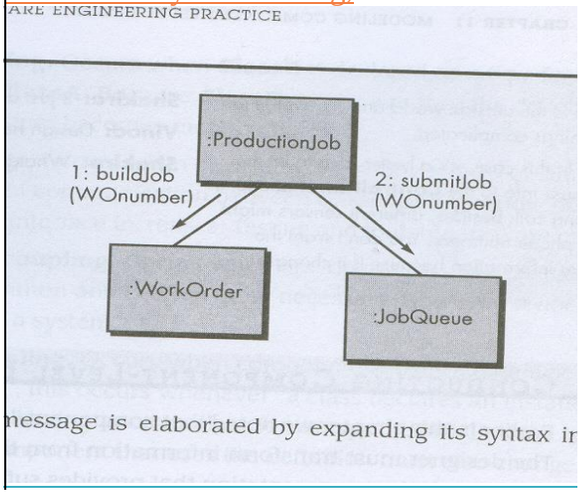


elaborated ~~Fig 11.5~~

2. Identify all design classes that correspond to the infrastructure domain. Classes not described in analysis model missing from archi model, they must be noted at some point, GUI compos, OS compos Object & data mgmt compos etc fall in this category

3. Elaborate all design classes that are not acquired as reusable compo. All i/f, operations & attributes described in detail, design heuristics considered

3a. Specify message details when classes or components collaborate. Use collaboration diagram to show collaboration of classes, specify structure of message passing b/w objects of a system ex, print shop system, 3 objects – ProductionJOb, WorkOrder & JobQueue collaborate for printing job, message passing shown by arrow in analysis modeling,



~~fig 11.6~~

During design message is expanded using syntax as

[guard condition] sequence expression (return value):=
message name (argument list)

[guard condition] is object constraint language, specifies set of conditions met before massge can be sent; sequence expression is an integer value indicating sequential order in which message is sent; (return value) returned value after operation invoked by message; message name operation to be invoked; (argument list) list of attributes passed to operation

3b. Identify appropriate i/f for each component. UML i/f ia a group of externally visible operations, i/f have no internal structure, no attributes, no associations, i/f are equivalent to abstract class, provides controlled connection b/w classes, each i/f has to be cohesive – single minded

3c. Elaborate attributes & define data types & data structures required to implement them. DS & data types defined within context of programming language, UML syntax for defining data types

name : type-expression = initial value { property string}

name is attribute name; type-expression is data type; initial value value of attribute at the time of object creation; property string property of attributes, during first design iteration attributes described by name, later on attributes defined using UML attribute format ex paperTyprweight

paperTypeweight: string = “a” {contains 1 of 4 values – A, B, C or D}

If an attribute appears repeatedly across a no of classes, create a separate class to accommodate it

3d. Describe processing flow within each operation in detail. Using programming-language based pseudocode or UML activity diagram, first iteration describe each operation as part of design class, high cohesion addressed, next iterations expand ex operation computePaperCost() elaborated as

computerPaperCost(weight, size, color): numeric

if algo is more complex further design elaboration done, UML activity diagram for computPaperCost() is, else use of pseudocode

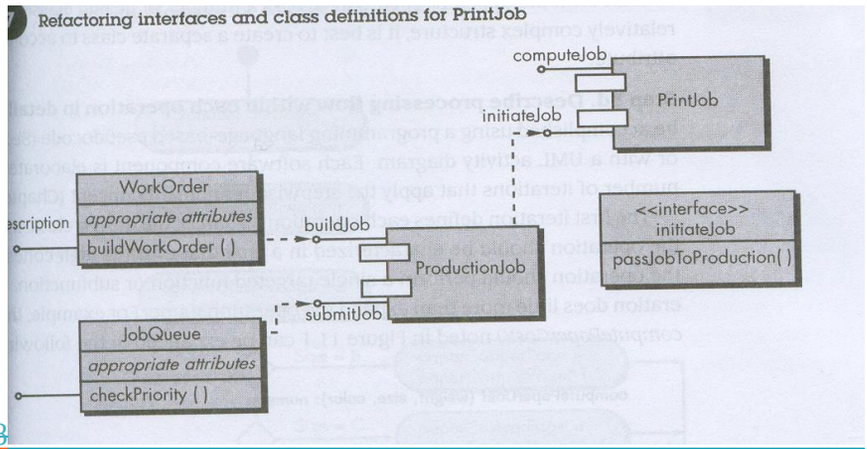


fig 11.8

4. Describe persistent data sources (databases & files) & identify the classes required to manage them. Persistent data source described at archi design, As design elaboration proceeds, provide additional details about structure & organization of persistent data sources

5. ~~5.~~ Develop & elaborate behavioral representations for a class or component During compo-level design, model behavior of a design class, dynamic behavior o an object affected by external events & current state, to understate dynamic behavior of an object examine all use-cases, delineate event that the object & states in which objects reside, transitions b/w states using UML statechart

Formatted: Indent: Left: -0.5", First line: 0", Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25", Tab stops: -0.19", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

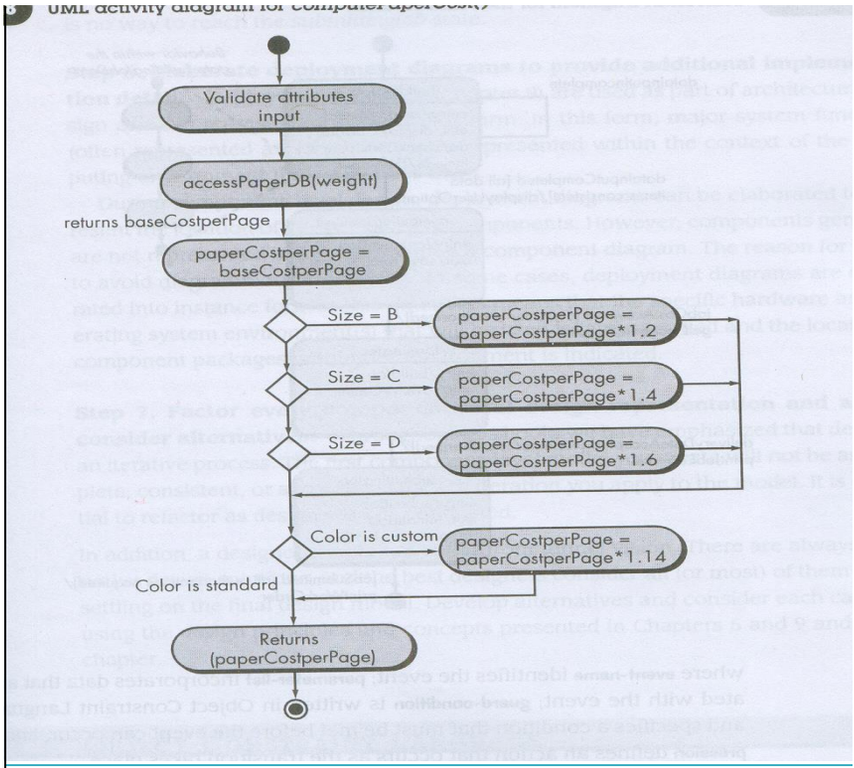


Fig 11.8

Transition from one state to another as a consequence of an event,

Event-name(parameter-list)[guard-condition]/action expression

event-name identifies event; parameter-list gives data associated with event; guard-condition written in object constraint language specifies condition to be fulfilled before event occurs; action-expression defines action as the transition takes place

Each state has entry/ & exit/ action as transition into & out of state, do/ indicates activities occurring in state, include/ provides a means for elaborating behavior, behavioral model contains info not obvious to other design model, in above ex, printjob class is contingent upon customer's cost & schedule approval, without approval print job cannot be submitted

6. Elaborate deployment diagrams to provide additional implementation detail. Deployment diagram elaborated to represent location of key packages
7. Factor every compo-level design representation & always consider alternatives.

11.4 OBJECT CONSTRAINT LANGUAGE

UML provides a rich presentation form for design model, along with graphical presentation a mechanism to explicitly represent info – a formal language known as Object Constraint Language, formal grammar & syntax to construct statements about design model elements, OCL statements have 4 parts (1) a context - define limited situations in which statement is valid (2) a property – some characteristics of context (if a class context that property is an attribute) (3)an operation (arithmetic)- manipulates / qualifies a property (4) keywords(if, and, not, or etc) – to specify conditional expressions

Ex consider guard condition on jobCostAccepted event that causes transition b/w states computingJobCost & formingJob, OCL expression will be

```
Customer
        Self.authorizationAuthority = 'yes'
```

Class customer has Boolean attribute authorizationAuthority, set yes to satisfy guard condition, Ocl provides a powerful tool to specify pre- & post-condition ex print shop system, a customer provides an upper cost bound & a fixed delivery date, if cost & print date exceeds these boundsjob is not submitted, in OCL

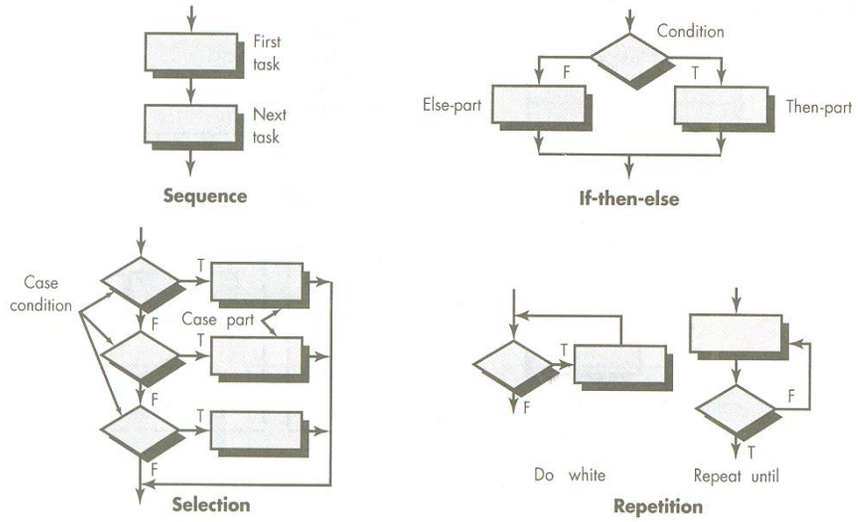
```
Context PrintJob::validate(upperCostBound : Integer, custDeliveryReq : Integer)
    Pre: upperCostBound>0
        And custDeliveryReq >0
            And self.jobAuthorization = 'no'
    Post: if self.totalJobCost <= upperCostBound
        And self.deliveryDate <= custDeliveryReq
        Then self.jobAuthorisation ='yes'
    Endif
```

This OCL statement defines conditions that must exist before & after some behavior,

11.5 DESIGNING CONVENTIONAL COMPONENTS :

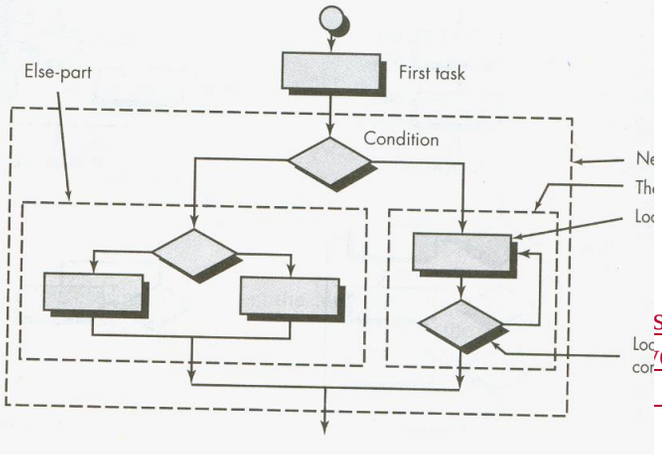
A set of constrained logical constructs to form program, emphasize on maintenance of functional domain i.e. construct has a predictable structure, entered at top & exit at bottom, constructs are sequence, condition & repetition, sequence implements processing steps of an algo, condition provide facility to select process & repetition allows looping, fundamental for structured design, structured constructs used to

limit design to small no of operations, reduces complexity, enhances readability, testability & maintainability, and enables understanding process called as chunks
Graphical Design Notation : depict procedural detail, misuse lead to wrong s/w, ex flowchart



Nesting may be applied, a complex schema can be developed,

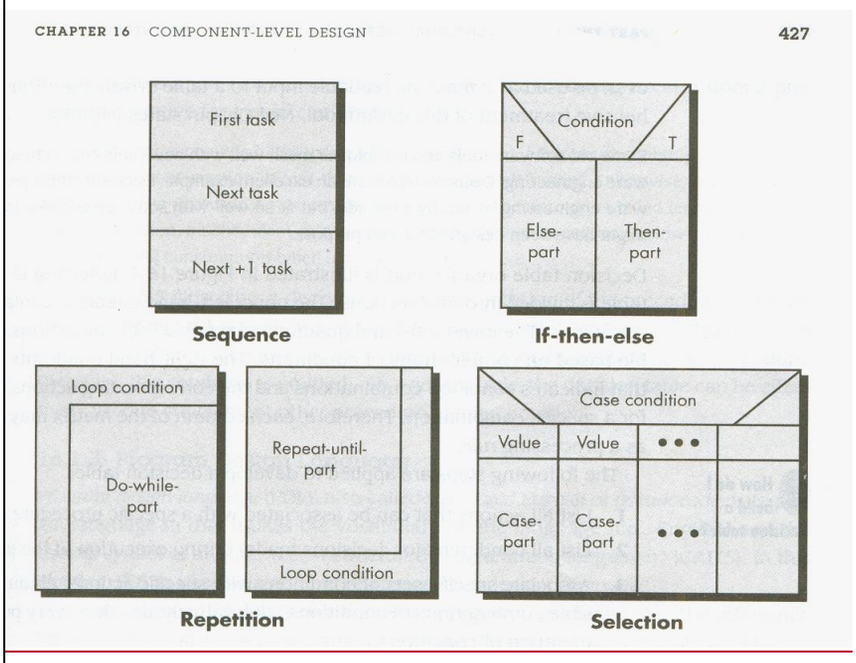
FIGURE 16.2
Nesting constructs



Dogm sted loop or
neste e impact on
readal procedural

representation so escape branch not needed (2) structured constructs violated in controlled manner, a constrained branch out of nested flow, ideally 1st option, 2nd option accommodated without violating spirit of structured programming

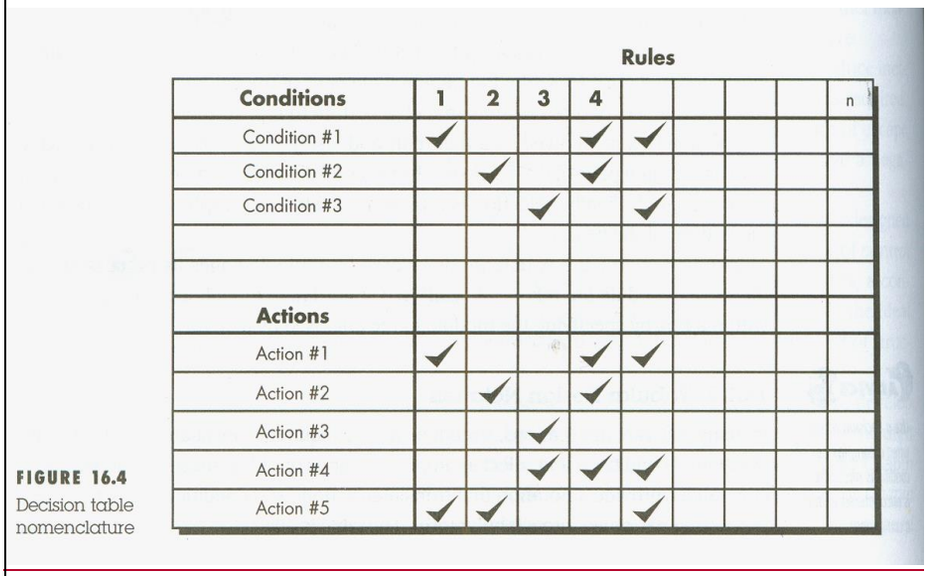
Another tool – box diagram – not allow to violate structured construct, characteristics (1) functional domain well defined & clearly visible (2) arbitrary transfer of control impossible (3) scope of local & global data easily determined (4) recursion easily represented



fundamental element is box, sequence by connecting 2 boxes, condition represented by a condition box followed by then-part & else-part, repetition by bounding pattern that enclose process to be repeated, selection using graphical form, box diagram layered on multiple pages, call to subordinate module represented within box by module name enclosed in oval

Tabular Design Notation : module evaluates complex combination of conditions & select appropriate action, decision table translated actions & conditions into tabular form, easy to interpret, used as machine readable i/p, table divided in 4 sections, upper left-hand quadrant – lists all conditions, lower left-hand – list of all actions on

combination of conditions, right-hand – a matrix indicating conditions combinations & corresponding actions for specific combination



	Rules								
Conditions	**1**	**2**	**3**	**4**					n
Condition #1	✓			✓	✓				
Condition #2		✓		✓					
Condition #3			✓		✓				
Actions									
Action #1	✓			✓	✓				
Action #2		✓		✓					
Action #3			✓						
Action #4			✓	✓	✓				
Action #5	✓	✓			✓				

FIGURE 16.4
Decision table nomenclature

To develop a decision table, steps are :

1. List all action associated with specific procedure
2. List all condition of procedure
3. Associate specific set of conditions with specific actions, eliminate impossible combinations
4. Define rules, indicate actions occurring for a set of combinations

Ex. Printing shop, 3 types of customers – regular, silver & gold (depending of amount of business), regular customer charged normal print rates & delivery; silver customer gets 8% discount in all quotes & placed ahead of regular customer; gold customer gets 15% discount & placed ahead of silver & regular customer, a special x% discount over other discounts at discretion of mgmt

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Bullets and Numbering

Conditions	**1**	**2**	**3**	**4**	**5**	**6**
Regular customer	T	T				
Silver Customer			T	T		
Gold customer					T	T
Special Discount	F	T	F	T	F	T
Actions						
No Discount	√					
8% Discount			√	√		
15% Discount					√	√
Additional X% Discount		√		√		√

Program Design Structure : (PDL) also structured English or pseudo code, combines syntax of programming language with vocabulary of English, like modern programming language, difference b/w PDL & programming lang is use of text embedded in syntax, can not be compiled, tools to translate PDL to programming lang, a design lang should have following characteristics :

1. fixed syntax of keywords for all structured constructs, data declaration & modularity characteristics
2. Free syntax of natural lang
3. data declaration facility, includes simple & complex DS
4. subprogram definition & calling techniques

Formatted: Indent: Left: -0.51", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.51" + Indent at: 0.51", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

should include construct for subprogram definition, i/f description, data declaration, block structure, condition construct, repetition & I/O, can include keywords for multitasking, concurrent processing, interrupt handling, interprocess synchronization etc, this design dictates final form

Ex – design of SafeHome Security system, following PDL describes its procedure

```
PROCEDURE security-monitor;
INTERFACE RETURNS system. status;
TYPE signal IS STRUCTURE DEFINED
   name IS STRING LENGTH VAR;
   address IS HEX device location ;
   bound.value IS upper bound SCALAR;
   message IS STRING LENGTH VAR;
```

```
END signal TYPE;
TYPE system.status IS BIT (4);
.
.
.
.
initialize all system ports & reset all hardware;
CASE OF control.panel.swotches (cps)
   WHEN cps = “test” SELECT
   CALL alarm PROCEDURE WITH “on” for test.time in seconds;
   WHEN cps = “alarm-off” SELECT
         CALL alarm PROCEDURE WITH “off”;

.
.
.
ENDCASE
REPEAT UNTIL activate.switch is turned off
   Reset all signal.values and switches;
   DO FOR alarm.type = smoke, fire, water, temp, burglar;
   .
   .
   .
   .
   ENDFOR
ENDREP
END security.monitor
```

COMPARISION OF DESIGN NOTATION :

No of different tech for procedural design, comparison for best result, procedural representation easy to understand & review, design should enhance code to ability so code derived from notation as a byproduct of design, easily maintainable, attributes of design notations are :

Modularity : support modular s/w & i/f facility

Overall simplicity : simple to learn, easy to use & read

Ease of editing :

Machine readability : can be i/p to a computerized development system
Maintainability : s/w maintenance, most costly, means maintenance of procedural design
Structure enforcement : structured programming concept, good design
Automatic processing : can be processed for better insight for correctness & quality
Data representation : local & global data – essential element of component-level design, directly
Logic verification : automatic verification of design logic
"code-to" ability : code generation easy, less effort & errors

program design lang offer best combination of above characteristics,
structured programming constrains no & type of logical constructs used to represent logical details, intent is to assist designer to define algos easy to read, test & maintain

10.6 Component-Based Development

Reuse of components, reuse of ideas, abstractions, processes in erlier days, an ad hoc approach, today's system more complex, high-quality, short development period - organized approach to reuse. "Component-based software engineering" - design & construction of systems usings reusable compo. No of questions
- is it possible to construct complex systems by assembling them from catalog or reusable s/w compo?
- accomplish in a cost & time- effective manner?
- s/w engineers be encouraged to use reusable compo than reinvent?
- added expenses for developing reusable compo?
- Library accessible to needed?
- compo found by needed?

Formatted: No underline

10.6.1 Domain Engineering

to identify, construct, catalog & disseminate s/w compos in a particular appli domain. Goal is to enable s/w engineers to share & reuse for new & existing systems. 3 major activities - analysis, construction & dissemination. domain analysis is OO SE.
1. define domain to be investigated
2. categorize items extracte from domain
3. collect representative sample of applis in domain
4. analyze each appli in sample & define analysis class

5. develop requirement model for classes

10.6.2 Component Qualification, Adaptation & Composition

Domain engineering provide library of reusable compo. some are in-house developed, some extracted from existing applis, some from third party. existance does not guarantee that compo integrated easily/effectively in archi of appli. a sequece of compo-based development actions when to reuse compo.

Formatted: No underline

~~PROCESS DECOMPOSITION : s/w team flexible to choose process model best for project, small project similar to past one best using linear model, strict time constraints & modular then RAD model, tight deadline & do not require full functionality then incremental model~~

12. PERFORMING USER INTERFACE DESIGN

~~Focus on three areas (1) i/f b/w s/w components (2) i/f b/w s/w & external entities (3) i/f b/w human & the computer~~

12.1 THE GOLDEN RULES

(1) Place the user in control (2) reduce user's memory load (3) make the i/f consistent

12.1.1 Place the user in control : Most i/f constraints & restrictions for simple interaction, such i/f easy to build, complex to use, to maintain control

- Define interaction modes in a way that does not force user into unnecessary o undesired action - able to enter & exit current state with little or no effort
- Provide for flexible interaction – different users have diff preferences, choice provided, interact via keyboard command, mouse, digitizer pen, voice recognition etc, use common sense
- Allow user interaction to be interruptible & undoable – in sequence of actions, able to interrupt sequence & undo action
- Streamline interaction as skill levels advance & allow the interaction to be customized – repeated work, macro mechanism for advanced user
- Hide technical internal from casual user – user not be aware of OS, file management functions or technicality, i/f never allow user inside the machine
- Design for direct interaction with objects that appear on the screen – user feels control when manipulate objects to perform tasks, ex, stretch the size

12.1.2 Reduce User's Memory Load : more to remember, more errors, system should remember info & assist user

- Reduce demand on short term memory – during complex task, short term memory in demand, reduce requirement to remember past actions & result provide visual cues

Formatted: Indent: Left: -0.5", Hanging: 0.25", Outline numbered + Level: 3 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.81" + Indent at: 0.81", Tab stops: -0.25", List tab + Not at 0.25" + 0.81"

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Indent: Left: -0.5", Hanging: 0.25", Outline numbered + Level: 3 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.81" + Indent at: 0.81", Tab stops: 0.06", List tab + Not at 0.25" + 0.81"

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

- Establish meaningful defaults – sensible initial defaults, reset option
- Define shortcuts that are intuitive – mnemonic for system functions meaningful
- The visual layout of the i/f should be baseds on a real world metaphor – same as real world layout
- Disclose info in a progressive fashion – hierarchical organization, first at high level of abstraction, detail after user indicate interest

12.1.3

- Allow user to put the current task into a meaningful context – provide indicators to know context of work, user able to determine from where he has com & alternatives
- Maintain consistency across a family of applications – same design rules
- If past interactive models have created user expectations, do not make changes unless necessary – user adaptive to interaction sequence, change cause confusion

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

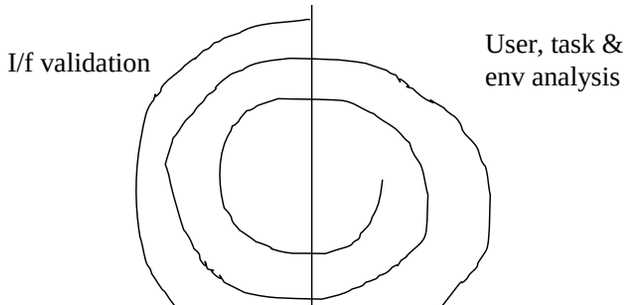
12.2 USER I/F ANALYSIS & DESIGN :

Begins with modeling of system functions, interactions required decided, design issue applied & tools used to create prototype
& the actual

12.2.1 I/f Analysis & Design Model : 4 models to design i/f, s/w engineer creates design model, human engineer – user model, end user – system perception, implementer – system image, each may differrent , i/f designer reconcile differences & derive consistent i/f, to build effective i/f – begin with understanding users, their age, abilities, education, background, goals & personality, user fall to one of the categories – Novice – no syntactic knowledge, little semantic knowledge, Knowledgeable, intermittent users – reasonable semantic knowledge, low syntactic info, Knowledgeable, frequent user – good semantic & syntactic knowledge, looks for shortcuts & abbreviations

System perception – image of system by end-users, system image combines outward manifestation & supporting info describing system syntax, when system image & perception coincide – user comfortable with such s/w, design models to accommodate user model, system image & system perception

12.2.2 The User I/f Design Process : iterative process, spiral model, 4 framework activities (1) User, task & env analysis & modeling (2) I/f design (3) I/f construction (4) i/f validation



Each 1 Implementation than once, each spiral I/f Design ation of requirements & resultant design, Analysis focus on profile of users, their skill level, business understanding, etc recorded, every user categorized, requirements of each category elicited, after then task analysis, tasks performed by users to accomplish goal of system identified, described & elaborated

User env analysis focus on physical work env, questions

- Where i/f will be located
- Will the user sitting, standing or performing other tasks unrelated to i/f
- Does the i/f h/w accommodate space, light or noise constraints
- Are there special human factors considerations driven by env factors

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

Info used to create analysis model, serve as a basis

I/f design define set of i/f objects & actions to perform tasks~~,~~ in a manner to meet usability goal, ~~implementation~~construction begin with creation of prototype, Validation focus on (1) implementation of every user task correctly, achieve all user requirements (2) Degree of easiness of using & learning i/f (3) user's acceptance

12.3 ~~TASK~~INTERFACE ANALYSIS ~~& MODELING~~ :

Understand problem before designing, for GUI understand (1) end-user who interact with system (2) tasks that end-user performs to do their work (3) content presented as part of i/f (4) environment in which tasks are conducted

12.3.1 User Analysis:

To understand system perception, designer must understand users & how users will use the system, many sources to accomplish this

User Interviews : Direct approach, s/w team representatives meet end-user to better understand their needs, motivation, work culture & other issues, one-to-one meeting or group meeting

Sales Input : Sales people meet customers & users regularly, gather info helpful to s/w team to categorize users & understand needs

Marketing I/p : market analysis, defines market segments & how each segment use s/w in diff ways

Support I/p : support staff talks with users, gathers info on what works & what do not, user's likes & dislike, features difficult or easy to use etc
A set of questions to understand users of system
- Are users trained professionals, technicians, clericals or manufacturing workers?
- What level of formal education does the average user have?
- Are users capable of learning from written material or want classroom training?
- Are users expert typists or keyboard phobic?
- What is the age range of user community?
- Users are of same gender?
- How are users compensated for the work they perform?
- Normal office hours or until the job is done?
- S/w as integral part of work users do or occasionally used?
- Primary spoken language
- Consequences of user making mistake using the system
- Expert in subject
- User want to know about the technology of the application

These leads to understand end-users, their motivation, how they can be grouped in diff profiles, system perception, system characteristics to meet their requirements,

12.3.2 Task analysis and Modeling

- What work will user perform in specific circumstances?
- Tasks & subtasks performed as the user works
- Specific problem domain objects manipulated as work performed
- Sequence of work tasks
- Hierarchy of tasks

Formatted: Indent: Left: -0.19", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.06", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

To answer these questions, apply techniques
Use-Cases. Describes manner in which actor interacts with system, use-case written in an informal style (paragraph), ex a s/w company want to build CAD system for interior designers, for better understanding actual interior designers asked to describe their specific design function, in answer to question – how to decide where to put furniture in room, designer describes:

I begin by sketching floor plan of room, dimensions & location of windows & doors. Light entering the room, about view out of the windows, running length of unobstructed walls, movement in room. I create list of furniture – sofa, chair, tables, cabinets, other accessories like lamps, paintings etc. then draw each item using a template scaled to the floor plan. I consider a no of alternatives & decide on one I like best. Then draw rendering of the room to make customer feel the look.

This becomes use-case for imp task of CAD, system engineer extracts tasks, objects & flow of interaction, additional features for interior designer also conceived ex. Digital photo looking out of every window, when room is rendered actual o/s view represented,

Task elaboration. To understand human activities to be accommodated in user i/f, task analysis in 2 ways – interactive computer-based system that replace manual system or semi-automated activity, human engineer understands tasks that human currently perform & map them in similar tasks implemented by user i/f, another way is to study existing specification for system & derive user tasks, human engineer define & classify tasks, ex CAD s/w, by observing interior designer at work major activities found are – furniture layout, fabric & material selection, wall & window covering selection, presentation, costing & shopping, these major tasks elaborated in subtasks, ex using info of use-case furniture layout subtasks are (1) draw a floor plan based on room dimensions (2) place windows & door at appropriate location (3a) use furniture template to draw scaled furniture outline (3b) use accent template to draw scaled accents (4) move furniture & accent outline for best placement (5) label all furniture & accent outlines (6) draw dimensions to show locations 97) draw rendering view
These subtasks can be refined further, each tasks consistent with user model 7 system perception

Object elaboration.
S/w engineer focus on use-cases & other info from users, extracts physical ojects used by interior designer, objects categorized in classes, attributes defined & actions applied to each object evaluated, Ex furniture class with attributes size, shape, location etc, designer select object from furniture class, move it to a position on floor plan, draw furniture outline etc, tasks select, draw, move are operations on object, details of each operation defined

Workflow analysis
When no of diff users with diff roles use same user i/f application of workflow analysis, allow to understand how a work process is completed when several people are involved, ex a Co. to fully automate process of prescribing & delivering drugs, a web-based application accessed by physicians, pharmacist & patients, workflow represented by UML swim lane diagram

FIGURE 12.2 Swimlane diagram for prescription refill function

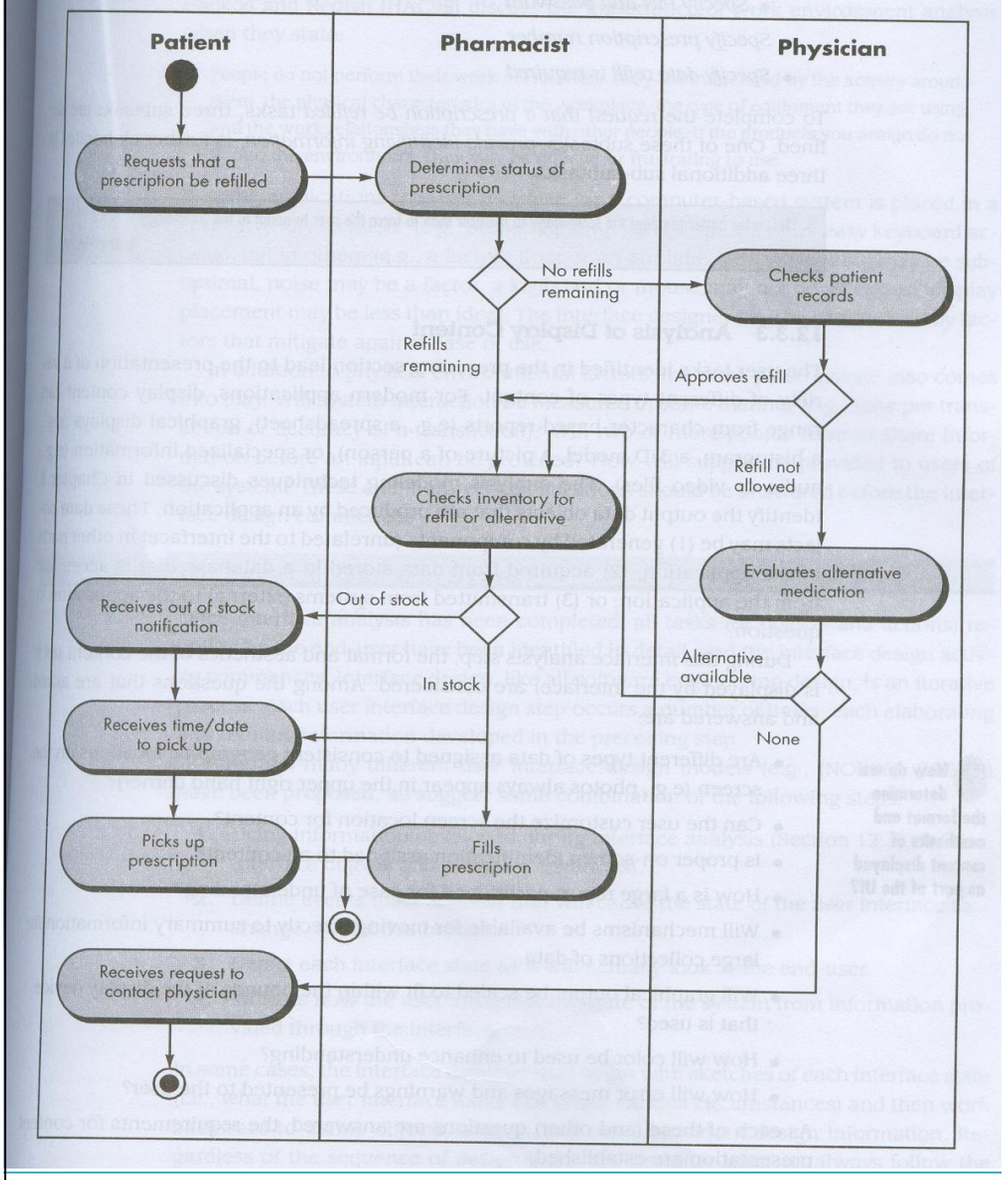


Fig 12.2

Suppose when a patient asks for refill, 3 key i/f characteristics

1. Each user implements diff tasks via i/f, look & feel of i/f for patient diff than for pharmacist or physician
2. i/f design for pharma & phys accommodate access & display of info from secondary sources ex inventory or alternate medications
3. activities of swim lane dia elaborated using task analysis or object elaboration

Hierarchical Representation

After work flow established, task hierarchy defined for each user type, ex user task for prescription refilling, following task hierarchy

Request that a prescription be refilled
- Provide identifying info
    - Specify name
    - Specify userid
    - Specify PIN & password
- Specify prescription no
- Specify data refill is required

Two subtasks defined, one subtask further elaborate to 3 sub-subtasks

12.3.3 Analysis of Display Content

user-tasks identify a variety of contents, display content range from character-based reports, graphical display or specialized info, o/p data objects may be (1) generated by components in other parts of application (2) acquired from data store (3) transmitted by external systems, format & aesthetics are imp, considerations are

(1) Diff types of data assigned to consistent geographic locations
(2) User able to customize screen locations of content
(3) Proper on-screen identification
(4) Partitioning of large reports
(5) Directly moving to summary info for large data
(6) Graphical o/p scaled to fit within VDU
(7) Use of colors to enhance understanding
(8) Presentation of error message & warnings

All these answers requirements of content presentation

12.3.4 Analysis of the Work Environment

Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: -0.5" + Tab after: -0.25" + Indent at: -0.25"

Formatted: Bullets and Numbering

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.25", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

Formatted: Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.25", List tab + Not at 0.25" + 0.5"

Formatted: Indent: Left: -0.5", Numbered + Level: 2 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.75" + Tab after: 1.25" + Indent at: 1.25", Tab stops: 0", List tab + Not at 0.25" + 1.25"

Formatted: Bullets and Numbering

User do not work in isolation, surrounding activity influence, some system works in user-friendly env but others sub optimal lighting, noise, i.e. less than ideal env, i/f should be constrained against ease of use, work place culture also play role, ex when two or more people share info before providing i/p etc must be looked after

12.4 ~~— map into similar set of tasks in context of user i/f — another method – by observing & studying existing system — derive set of user tasks to accommodate user model, design model & system perception — first step is to define & classify the tasks- one two approaches — step wise elaboration : observes work, find major activities, elaborated in sub activities, model consistent with user model & system perception — another is OO approach : observes physical objects used by users & action applied to each object~~

1. Define i/f objects & actions
2. Define events that will cause state of the i/f to change
3. Depict each i/f state, how actually it will look to end user
4. Indicate user interpretation of state of info

Formatted: Indent: Left: -0.5", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

Formatted: Bullets and Numbering

Designer draws sketches of each i/f state, work backward to define objects, actions & other imp design info, designer must (1) always follow golden rules (2) model hoe i/f implemented (3) consider env

12.4.1 Applying I/f Design Steps:

Imp step is to define i/f objects & actions on them, use-cases described & list of objects & actions isolated, categorized by type, target, source & application objects, source object is dragged & dropped onto a target object e report icon dragged & dropped on print icon, result is printed report, application object represent application specific data which can not manipulated directly as part of screen interaction ex. A mailing list store names, list sorted, merged but cannot dragged & dropped

All objects & actions defined, screen layout performed, screen layout present graphical design & placement of icons, definition of screen text, specification & titling of windows, definition of major & minor menu items etc, real world metaphor applied & layout organized to complement metaphor ex safe-home s/w, a use-case for the i/f is

Use case: user want to gain access to system from any remote location via internet, using browser s/w can determine status of alarm system, arm or disarm system, reconfigure security zones, view diff rooms via cameras, for remote location access provision of id & password, they define level of access & provide security, user can check status of system 7 change status by arming/disarming system, reconfigure system by displaying floor plan of house, viewing each sensor, display configured zones, & modify zones, view interior of house with cameras, pan & zoom camera for diff views

Based on this use-case, following tasks, objects & data items identified

- *Access* the safe-home system
- *Enter* **ID** & **PASSWORD** for remote access
- *Checks* **system status**
- *Arms & disarms* system
- *Display* **floor plan** & **sensor location**
- *Display* **zones** on floor plan
- *Changes* **zones** on floor plan
- *Displays* **video camera location** on **floor plan**
- *Selects* **video camera** for viewing
- *Views* **video images**
- *Pans* or *zooms* **video camera**

Objects in bold, actions in italics, extracted from homeowner tasks, **video camera location** can be dragged & dropped onto **video camera** to create **video image,** screen layout for video monitoring created

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: -0.25", List tab + Not at 0.25" + 0.5"

Formatted: Bullets and Numbering

12.4.2 User Interface Design Pattern

A wide variety of user i/f patterns for GUI, a pattern is an abstraction of a design solution to a design problem, patterns such as Whole UI, Card stack etc, includes design classes, attributes, operations & i/f

12.4.3 Design Issues

4 issues : system response time, user help facility, error handling & command labeling, considered in beginning when easy to change & low cost

System response time – primary complaint for interactive systems, time taken from some control action to system response – 2 characteristics, length & variability – too long response time creates frustration, very brief response time may force user

to rush & make mistakes – variability refer to deviation from average response time, – avg response time must not vary – establish interaction rhythm

Help facilities: must for interactive systems – on-line help to resolve difficulties without leaving i/f – design issues to be addressed are :

- Will help available for all system functions & at all times
- How will user request help – menu, function key and HELP command
- Help representation – separate window, reference to printed document, one – two line suggestion in a fixed location
- How to return back to normal interaction – return button, function key or control sequence
- Information structure – flat structure- all info accessed through a keyword, layered hierarchy- provide increasing hierarchy or use of hypertext

Error handling: Error message & warnings must produce proper information, misleading info increases user frustration, should have following characteristics :

- Massage describe the problem in user understandable manner
- Provide constructive advice for recovering
- Indicate negative consequences of error
- Audible or visual cue, beep, color, flash
- Nonjudgmental error message

Effective error message improve quality of interactive system & reduce user frustration

Menu & Command Labeling:

before few years typed command, now window-oriented, many like command oriented mode leads to , design issues for typed commands or menu labeling

- Every menu options have command
- Form of command, control key, function key, typed command etc
- Difficult to learn & remember, if forgotten
- Customized or abbreviated command
- Menu labels self-explanatory
- Submenu consists same function implied by master menu item

Applications Accessibility.

Easy access for special needs, accessibility for physically challenged, visually, hearing, mobility, speech & learning impairment

Internationalization.

Accommodate needs of diff local languages, create globalize s/w, Unicode std developed to manage dozens of languages

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

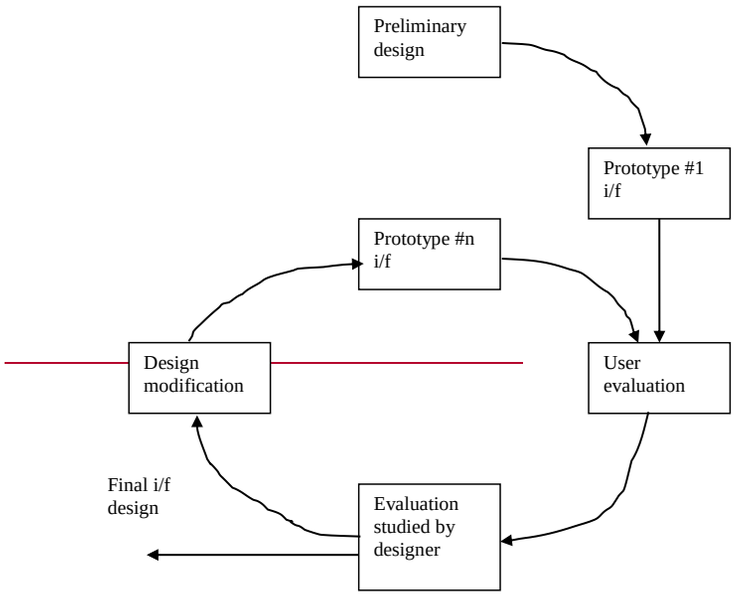
Formatted: Bullets and Numbering

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

12.5 DESIGN EVALUATION :

After creating operational prototype, evaluate for requirements conformance, informal test drives to formal evaluation



Prototype approach effective, long development cycle, if design model of i/f created, evaluation criteria applied in early phases such as :

- Length & complexity of written specification & its i/f provide indication of amount of learning required by users
- No of user tasks & avg no of actions per task– interaction time & efficiency of system
- No of actions, tasks & system states – memory load of users
- I/f style, help, error handling – complexity of i/f, degree of acceptability

Formatted: Indent: Left: -0.5", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Tab stops: 0.25", List tab

Formatted: Bullets and Numbering

Qualitative & quantitative data collected & used to evaluate i/f, to collect data questionnaires created & distributed to users – yes/no type, numeric response, scaled, percentage etc ex. Were icons self explanatory, how easy to learn basic system operations etc

Quantitative data collected by observing users during interaction, data – no of tasks completed over time period, frequency of actions, sequence of actions, no of errors, type of errors, error recovery time etc – collected & provide a guide for i/f modification

This, 3 imp principle to guide the design – place user in control, reduce user's memory load, consistent i/f, i/f is window into s/w, if it is smudged, wavy or broken user may reject it