Design Constraints

Software Engineering Design
Lecture 9

Design Goals

- Before leaping from Requirements Analysis into System Design, you should ensure that you have identified the design goals for your system
- Many design goals can be inferred from the non-functional requirements or the application domain. Others should be checked with the client.
- Design Goals need to be stated *explicitly* so that future design criteria can be made consistently, following the same set of criteria

Types of Design Goal

- There are many desirable qualities which may be design goals for your system:
  - performance
  - dependability
  - cost
  - maintenance
  - end user criteria
- Meeting some of these goals may conflict with meeting others - can you think of an example of conflicting goals ?

Design Goal Example

Classify each design goal below according to performance, dependability, cost, maintenance, end user criteria

- Users must be given feedback within 1 sec of issuing a command
- The TicketDistributor must be able to issue train tickets even in the event of a network failure
- The housing of the TicketDistributor must allow for new buttons to be installed if the number of different fares increases
- The AutomatedTellerMachine must withstand dictionary attacks (ie. ID numbers discovered by systematic trial)
- The user interfaces of the system should prevent users from issuing commands in the wrong order

Design Goals come from Requirements

- A *functional requirement* describes a system service or function.
- A *non-functional requirement* is a constraint placed on the system or on the development process
- *Note: we shall classify B&D's pseudo requirements as a special class of non-functional requirements*
- *Check lists* are useful for identifying non-functional requirements

Type of Non-functional Requirements

- 3.3.1 User interface and human factors
- 3.3.2 Documentation
- 3.3.3 Hardware considerations
- 3.3.4 Performance characteristics
- 3.3.5 Error handling and extreme conditions
- 3.3.6 System interfacing
- 3.3.7 Quality issues
- 3.3.8 System modifications
- 3.3.9 Physical environment
- 3.3.10 Security issues
- 3.3.11 Resources and management issues

NFR Trigger Questions (1)

- 3.3.1 User interface and human factors
  - What type of user will be using the system?
  - Will more than one type of user be using the system?
  - What sort of training will be required for each type of user?
  - Is it particularly important that the system be easy to learn?
  - Is it particularly important that users be protected from making errors?
  - What sort of input/output devices for the human interface are available, and what are their characteristics?

NFR Trigger Questions (2)

- 3.3.2 Documentation
  - What kind of documentation is required?
  - What audience is to be addressed by each document?
- 3.3.3 Hardware considerations
  - What hardware is the proposed system to be used on?
  - What are the characteristics of the target hardware, including memory size and auxiliary storage space?

NFR Trigger Questions (3)

- 3.3.4 Performance characteristics
  - Are there any speed, throughput, or response time constraints on the system?
  - Are there size or capacity constraints on the data to be processed by the system?
- 3.3.5 Error handling and extreme conditions
  - How should the system respond to input errors?
  - How should the system respond to extreme conditions?

NFR Trigger Questions (4)

- 3.3.6 System interfacing
  - Is input coming from systems outside the proposed system?
  - Is output going to systems outside the proposed system?
  - Are there restrictions on the format or medium that must be used for input or output?

NFR Trigger Questions (5)

- 3.3.7 Quality issues
  - What are the requirements for reliability?
  - Must the system trap faults?
  - Is there a maximum acceptable time for restarting the system after a failure?
  - What is the acceptable system downtime per 24-hour period?
  - Is it important that the system be portable (able to move to different hardware or operating system environments)?

NFR Trigger Questions (6)

- 3.3.8 System Modifications
  - What parts of the system are likely candidates for later modification?
  - What sorts of modifications are expected?
- 3.3.9 Physical Environment
  - Where will the target equipment operate?
  - Will the target equipment be in one or several locations?
  - Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibrations, magnetic fields, ...)?

NFR Trigger Questions (7)

- 3.3.10 Security Issues
  - Must access to any data or the system itself be controlled?
  - Is physical security an issue?
- 3.3.11 Resources and Management Issues
  - How often will the system be backed up?
  - Who will be responsible for the back up?
  - Who is responsible for system installation?
  - Who will be responsible for system maintenance?

Non-Functional (Pseudo) Requirements

- Non-functional (Pseudo) requirement:
  - Any client restriction on the solution domain
- Examples:
  - The target platform must be an IBM/360
  - The implementation language must be COBOL
  - The documentation standard X must be used
  - A dataglove must be used
  - ActiveX must be used
  - The system must interface to a papertape reader

Evaluating Designs

- When is a design **correct**?
  - If it can be shown to capture all the functions of the requirements document?
  - If it captures all the users' requirements?
- What makes a design a **good** design?
  - It is correct, complete, consistent, realistic and readable

Some Evaluation Criteria

- product vs process
- differing views: client, developer, user
- design goals (from non-functional requirements)
- cohesion and coupling in subsystems
- comparing designs: evaluation matrix
- rationale

Modular design

- A design is **modular** when
  - each activity of the system is performed by exactly one component
  - inputs and outputs of each component are **well-defined**, in that every input and output is necessary for the function of that component
  - the idea is to minimise the impact of later changes by abstracting from implementation details

Correct Designs

- Does the design correctly capture the requirements?
- Are the requirements the right ones?
- These questions can be addressed by:
  - testing the design against both the requirements document and against user expectations.
  - analysing the requirements for completeness, consistency, realism
  - design review meetings
  - formal proof that design model D satisfies requirements model R

Correct OO Designs

- Can every subsystem be traced back to a use case or nonfunctional requirement?
- Can every use case be mapped to a set of subsystems?
- Can every design goal be traced back to a nonfunctional requirement?
- Is every nonfunctional requirement addressed in the system design model?
- Does each actor have an access policy: what data and functionality is available to each actor?
- Is the AP consistent with the nonfunctional security requirement?

Complete OO Designs

- Has every requirement and every system design issue been addressed?
- Have the boundary conditions been handled?
- Was there a walkthrough of the use cases to identify missing functionality in the system design?
- Have all use cases been examined and assigned a control object?
- Have all aspects of system design been addressed?
- Do all subsystems have definitions?

Consistent OO Designs

- Does the design contain any contradictions?
- Are conflicting design goals prioritized?
- Are there design goals that violate a nonfunctional requirement?
- Are there multiple subsystems or classes with the same name?
- Are collections of objects exchanged among subsystems in a consistent manner?

Realistic OO Designs

- Can the design be implemented?
- Are there any new technologies or components in the system? Have the appropriateness and robustness of these technologies been investigated?
- Have performance and reliability requirements been reviewed in the context of the subsystem decomposition?
- Have concurrency issues been addressed?
  - See next slide

Concurrency Issues

- *Contention:* 2 processes competing for access to the same resource
  - e.g. writing to a network bus such as the CANbus
- *Deadlock:* 2 processes are waiting for each other and therefore can make no progress
  - e.g. the dining philosphers each holding one fork
- *Mutual exclusion:* a resource must only be accessed by one processes at a time
  - e.g. crediting and debiting a bank account

Readable OO Designs

- Can developers not involved in the system design understand the model?
- Are subsystem names understandable?
- Do entities with similar names denote similar phenomena?
- Are all entities described at the same level of detail?

Design Evaluation Matrix: a tool for comparing different designs

- Characteristics for comparison include:
  - easy to change algorithm
  - easy to change data
  - easy to change function
  - good performance
  - ease of reuse
  - modularity, testability, maintainability, efficiency,
  - ease of understanding, ease of modification, consistency

Comparing Designs - Measures

We can compare two different designs by

- identifying a list of relevant design characteristics c_0 to c_n and (optionally) a weight w_0 to w_n for each
- checking for each design characteristic whether the given design exhibits it or not: $e_i = 0$ or $e_i = 1$
- Quality = $e_0 * w_0 + e_1 * w_1 + \ldots + e_n * w_n$

Suitable characteristics include: modularity, testability, maintainability, efficiency, ease of understanding/modification, consistency ...

Design Evaluation Matrix Example

Design Characteristic	*Weight*	*Design 1*	*Design 2*
Portability	5	1	0
Easy to use & robust	2	1	1
Response time	1	0	1
TOTAL	8 max	7	3

Now you try one

- List up to 4 characteristics you would use in a ***design evaluation matrix*** for an automatic bank teller system
- Identify weights for each characteristic giving reasons for your choices
- What information do you need to evaluate each characteristic?