



Fundamentals of SE Ch1-4

Quals 2007

**[ARRESTED
COMPUTING]**

On this episode...

- ▶ Chapter 1 – Preview
- ▶ Chapter 2 – Software: It's Nature and qualities
- ▶ Chapter 3 – Engineering Principles
- ▶ Chapter 4 – Design and Architecture

1.1 : Preview

- ▶ SE means multiple people building multi-version software
- ▶ Engineering large software systems appears fundamentally more difficult than programming the individual components
- ▶ The “Water-fall” model
 - ▶ Requirements analysis and specification
 - ▶ System Design and specification
 - ▶ Coding and module testing
 - ▶ Integration and system testing
 - ▶ Delivery and maintenance

2.1 - Qualities

- ▶ External vs. Internal
- ▶ What's important depends on domain
- ▶ Correctness
 - ▶ implementation satisfies specification
- ▶ Dependability
 - ▶ Some number of errors almost unavoidable
 - ▶ Reliability – how long does it operate continuously
 - ▶ Availability – what percentage of the time is it working
 - ▶ Typically this is part of the spec for a critical system
- ▶ Robustness
 - ▶ Should behave “reasonably” in unanticipated circumstances

2.2 - Qualities

- ▶ **Performance**
 - ▶ Affects usability
 - ▶ Depends on other technology
 - ▶ Typically requires analysis and simulation
- ▶ **Usability**
 - ▶ Human factors
- ▶ **Verifiability**
 - ▶ Many techniques
- ▶ **Maintainability**
 - ▶ Software must evolve
 - ▶ A large part of SE consists of repurposing old or COTS systems

2.4 – Qualities

- ▶ **Reusability**
 - ▶ Can apply to software modules as well as other artifacts and processes
- ▶ **Portability**
 - ▶ Run in different environments
- ▶ **Understandability**
 - ▶ Engineers spend most of their time reading (and trying to understand) code
- ▶ **Interoperability**
 - ▶ Plugins...

2.5 – Process Qualities

- ▶ **Productivity**
 - ▶ Difficult to measure
- ▶ **Timeliness**
 - ▶ User needs vs. system capabilities
 - ▶ Incremental delivery
- ▶ **Visibility (transparency)**
 - ▶ Often exact status of product is unknown

3.1 – Engineering Principles

▶ Rigor & Formality

- ▶ Increasing formality can lead to better systems, but is expensive
- ▶ Knowing where and when its necessary is important
- ▶ Can apply to any stage, and the process itself

▶ Separation of concerns

- ▶ Grouping related decisions
- ▶ Grouping related “qualities” (e.g. efficiency and safety)

▶ Modularity

- ▶ A way to achieve separation of concerns
- ▶ Decomposing a system into simple pieces (top down)
- ▶ Composing a system from simple pieces (bottom up)

3.2 – Engineering Principles

- ▶ **Abstraction**
 - ▶ Hide the details
 - ▶ Many (including Sullivan) would say that this is the key to SE
- ▶ **Anticipation of change**
 - ▶ Need to be able to do this in order to choose the right abstractions (Parnas)
- ▶ **Generality**
 - ▶ Save time by solving the “general” case once
- ▶ **Increment-ality**
 - ▶ Early feedback
 - ▶ Less wasted work

4.1 – Design and Architecture

- ▶ Design is system decomposition into modules
 - ▶ What is a module? So unclear
 - ▶ Need to separate things so that they can be done in parallel

4.2 – So what changes?

- ▶ Algorithms
 - ▶ e.g., we want to make a process more efficient
- ▶ Data Representation
- ▶ The abstract machine
 - ▶ New OS, different compilers, etc.
- ▶ Peripherals
- ▶ Social Environment
 - ▶ Laws change etc.
- ▶ The Development Process

4.3 – Product (Program) Families

- ▶ We have a whole paper on this (Parnas)
- ▶ Essence: you don't just write one version
- ▶ Different people want different feature sets
 - ▶ Case in point: 6 versions of Vista
- ▶ May need to be able to support different platforms
- ▶ New versions

4.4 – Modularization

- ▶ Modules depend on or “use” each other
- ▶ We have a whole paper on “the USES relation” (Parnas)
- ▶ Ideally, the USES graph wont have cycles
 - ▶ “nothing works until everything works”
- ▶ **These is also** IS_COMPONENT_OF, IMPLEMENTS, COMPRISES, etc
- ▶ Note, that these are all *static* properties
- ▶ These are good, but they don’t capture everything

4.5 – More on Modules

- ▶ **Export the minimal amount of detail**
 - ▶ Makes things simple
 - ▶ Facilitates change
- ▶ **Sometimes things you would rather hide cant be hidden**
 - ▶ speed, memory usage, etc.
- ▶ **Mechanisms vs Policies**
 - ▶ In general, best to keep them separate
- ▶ **Abstract Data Types and Generic Data Structures**
 - ▶ Instead of CarList consider List<Car>

4.6 – Top down vs Bottom Up

- ▶ **Top-down – write main() first – Dijkstra**
 - ▶ A.k.a. stepwise refinement
 - ▶ More simple / intuitive / straightforward
 - ▶ But, ignores the important issues of info hiding etc.
 - ▶ Tends to be easy to break
- ▶ **Bottom-up – write main last() – Parnas**
 - ▶ Figure out what's is likely to change first – and hide those decisions
 - ▶ Then figure out how to connect everything
 - ▶ Harder to do, but usually results in a more maintainable system

4.7 – Standard Architectures

- ▶ **Pipeline**
 - ▶ String of data transducers
- ▶ **Blackboard**
 - ▶ Everyone writes and queries a single interfaces
- ▶ **Event-based**
 - ▶ Publish() subscribe() or listeners
- ▶ **Model View Controller (MVC)**
 - ▶ Good if there lots of user interation
 - ▶ E.g. Java swing

4.8 – Other Notable Architectures

- ▶ **Standard Template Library (STL)**
 - ▶ Lots of generic algorithms
 - ▶ Also, very annoying
- ▶ **Swing Widgets**
 - ▶ Many objects that support a standard set of methods (e.g. “on mouse click”)
- ▶ **JavaBeans**
 - ▶ reusable software components that can be manipulated visually in a builder tool
- ▶ **Common Object Request Broker Arch (CORBA)**
 - ▶ “wraps” program code into a bundle containing information about the capabilities of the code and how to invoke it.

4.9 – Other random things in this chapter

- ▶ **Conditional Compilation**

- ▶ `# ifdef x86 ...`

- ▶ **Software Generation**

- ▶ **Monitors for concurrency control**

- ▶ We'll save this for when we do the OS book

- ▶ **Client – Server Model**