CS4504 Tuesday 17th September 2013

Advanced Software Engineering

Dr. John Herbert (178) jherbert@cs.ucc.ie

Lab: Wednesday 3-5. (from week 6.)

http://www.cs.ucc.ie/~herbert/cs4504

- Requirements.
- Process
- Model Based Development
- Architectures
- Distributed Architectures
- Global based.
- Cloud based
- Non-functional Properties

Other items that are not on Module Descrip.

- Non-functional Properties
  - Scalability
  - Performance / latency
  - Usability.
  - Maintainability
  - Reliability / Dependability.
  - Security
  - Running costs.

Requirements

- Design — What !
- Architecture

How !

Code

CS4504 Thursday 19th September 2013

Ignite Program - 1st October 2013

Lecture 1 - Software Processes.

How you organise the activites!
- Waterfall
- Agile

} Review of Year 3.

Lecture 2

Boehm's Spiral Model of the software Process

Rational Unified Process.

70's → Models / Notations
- Data Flow Diagrams
- Entity Relationship Diagrams.

90s → - Object Oriented Design (Anaysis)
- OOA
- OOD
- single open standard developed

UML

Requirements —?→ Code

UML → General Information Systems
→ Embedded Systems

First → Second and how do they relate

RUP — Own case tool, and process for using the notation.

Chapter (Lecture 3) 3 - Agile

★ Principles of Agile Methods
- Customer Involvement
- Incremental Delivery
- People not process
- Embrace Change
- Maintain Simplicity

CS4504 Tuesday 24th September 2013

Functional Requirements
- Items absolutly required of the system.
  eg Football Scores App should have an actual scores page.

Non-Functional is everything else speed, performance and quality of the system. eg system developed in C# or Java.

Users
- System Customers
- Managers
- System Engineers
- System Test Engineer
- System Maintenance Engineer

★ Requirements vs Design

- Requirements :- what the system should do!
- Design :- how you should do it!

Req A
Req B
> conflict ⇒ Compromise to meet GOAL (resolve conflict)

CS4504 Thursday 26th September 2013

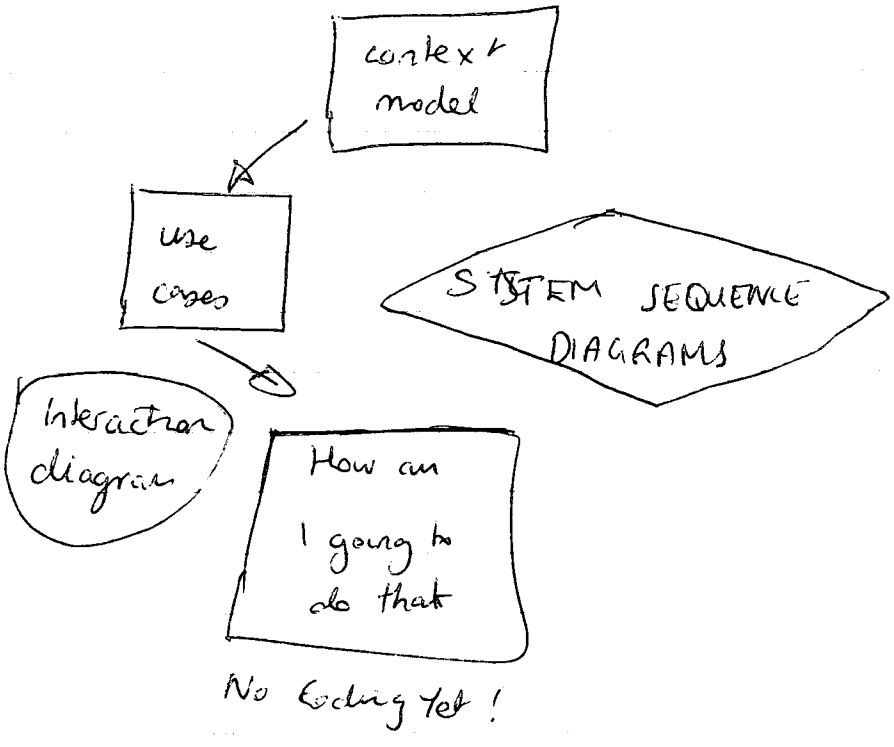
Topics Covered
- Context Models.
- Interaction Models
- Structural Models
- Behavioral Models
- Model-driven Engineering

o Unified Modelling Language (UML)

Context models – set the boundry of the system
– Boundries – what is inside/outside the system

Use cases – a case of use of the system from the users point of view.

Lecture



Aggregation vs Composition

State Diagram (eg of a microwave oven)

MDA - ◦ Model Driven Architecture
◦ Agile methods can be used.
◦ Model Business Process.
- It doesn't matter what the code looks like but that it does what is required.
◦ - Abstract view of the software. comes next, you can modify this but the MBP is not modified

CS4504 Tuesday 1st October 2013

Ignite Talk. (2nd Floor)
Eamon Curtin

MealthBox (Sandbox)

Eric Reese
Steve Beberal?

Copyright - text
Patent - process
TM - name / logo
Industrial Design - eg furniture.

a set of workshops (funding / IP / law)
(network / business model / attitude)

CS4504 Thursday 3rd October 2013

Chapter 5 - System Modelling - Lecture 1 (continued)

omg.org - Model Driven Architecture.
Object Management Group.

Model Driven Engineering (MDE)

Use Cases

- User story
  - Try to capture all possibilities so that testing can be easier in the end.

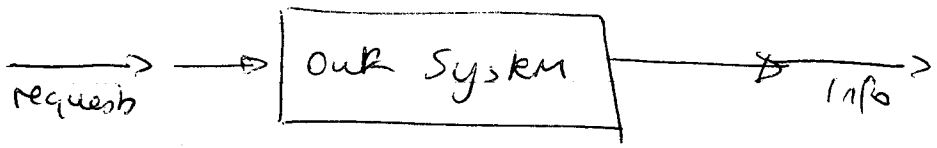
ES4504 Tuesday 8th October 2013

Use Cases ⇒ Req for System.

Library System
- Stake Holder ← Actors (Interact with sy.tem)
  Publishers / Book Suppliers
  Owners (UCC / City council et) Funding etc.
  Data Protection Authority
- Actors
  - Library Uses / Member
  - Library Staff
  - Publishers
  - Sys Admin
  - Guest

Use Case

→ Borrow Book
→ Actors → Member / Guest / Library Staff
→ Interactions involved →
  : → member takes book to check out at desk
  library staff records book (enters details)
  System records book as being checked
  LibStaff enter user details
  System returns user details
  LibStaff checks at book out
  System records book as checked out.



Larman Domain Model

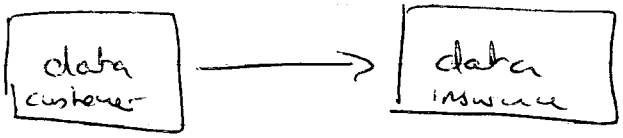
(Entity Relationship Diagrams)
(Objects + Relationships)

- Analysis of the domain based concepts (classes, objects) not functions
- Purpose: to make a model that decomposes the problem and communicates the important domain terms and how they are related

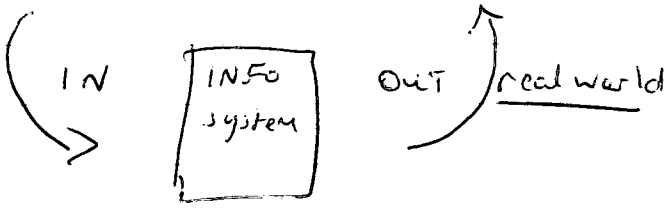
OO Data Analysis — Domain Model
Map Diagram of problem domain (concept).

Conceptual Model — real world not software
- no software artifact such as GUI, database
- No responsibilities

Data — customer records
— insurance records / policies



Classes plus attributes



That is what it is all about!

UPC - Universal Product Code.

CS 4504 Tuesday 1st October 2013

Scaling Lean and Agile Development ← BOOK

Large Scale Agile Design and Architecture ways of working

Eddie Kenny – IT Manager @ BETFAIR

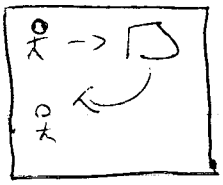
eddiekenny.jobs@gmail.com

SIDE NOTE NOT IMPORTANT

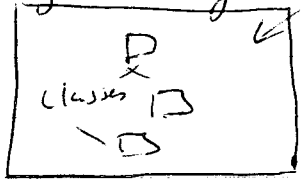
UML – External View of the System.

Sequence Diagram – sequence of events ('System as a black box')

Use Cases



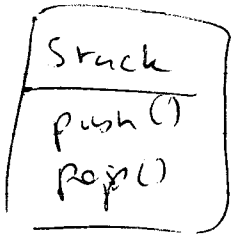
System (API of the) Methods IN



Not designing code yet just figuring out what happens

UML Modelling Language → Methods

- What is/is not in the system!

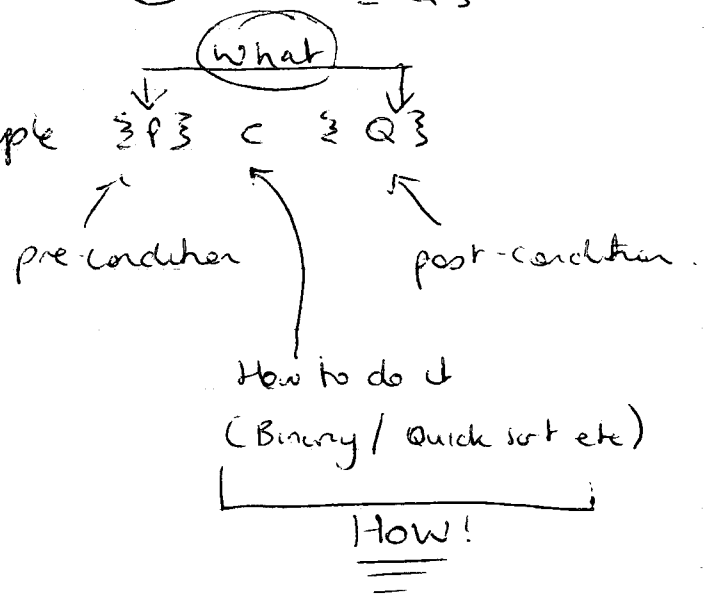


linked List
Arrays
Vectors
etc

Next we have the Contract.
The Contract is what the method does.
- Do some computation and return the result. For example. Computes
- Change the attributes of the system (ie the data)
- Could call other Methods.

★ Design by Contract – Bertrand Meyer

Sorting : Bubble / Merge / Binary / Random / Quick
They all take a set of items and returns an ordered set of items.



c.f. Hoare triple

Software Engineering is about the what and not the how. (How is just basic overview ie BUBBLE SORT – not actually doing it).

Contract

	example
Name	enterItem -(ItemID, quantity etc) operations
Responsibilities	Use Cases ; Process Sale
Notes	
Output	
Pre-conditions	Sale underway
Post-conditions.	gathers info such as quantity of items total cost

Contracts give meaning !.

Conceptual Model : (Domain Model)
The problem domain
- The effects of what these methods are.
(We still haven't designed the software yet)
- What you need to keep track of !

UML Notation — write the process down in english.

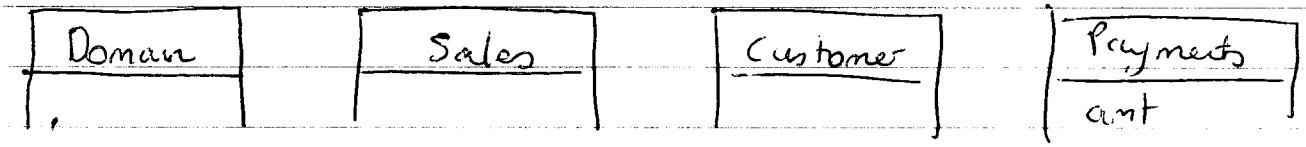
OCL : Object Constraint Language. is a UML extension expressing constraints such as pre, post-conditions.

CS4504 Tuesday 22nd October 2013

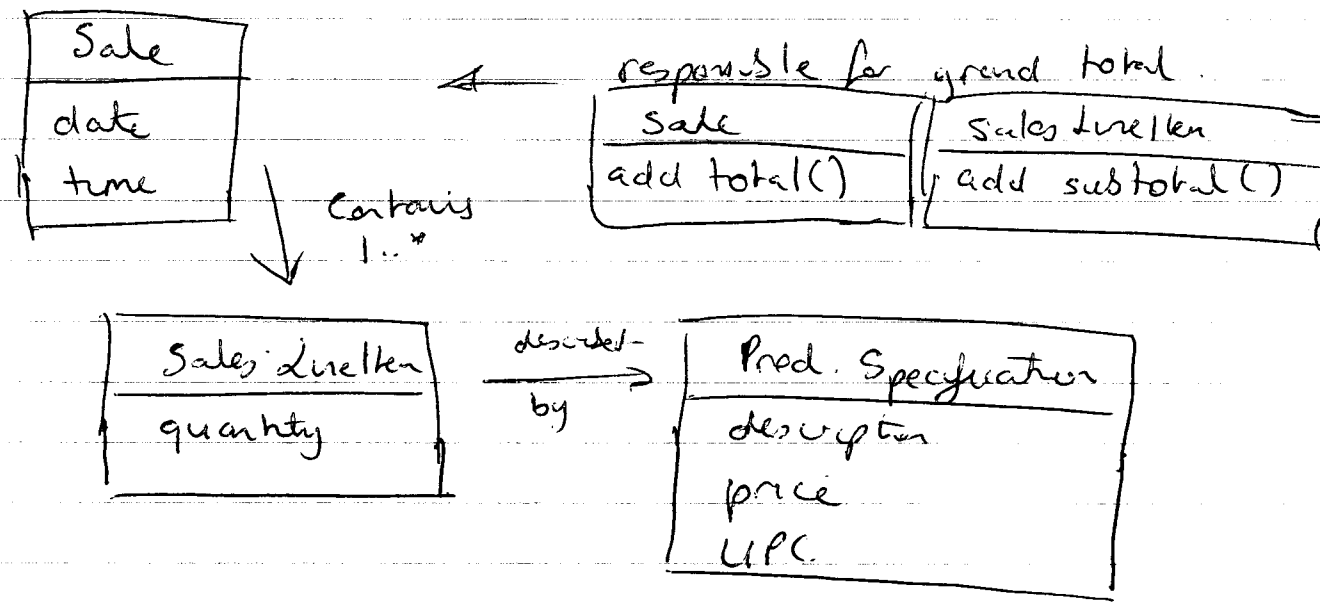
Xerox - Fumbling the future.
Alan Kay - Dynabook (OO Programming)

Kent Beck / Ward Cunningham

GRASP : General Responsibility Assignment software Patterns



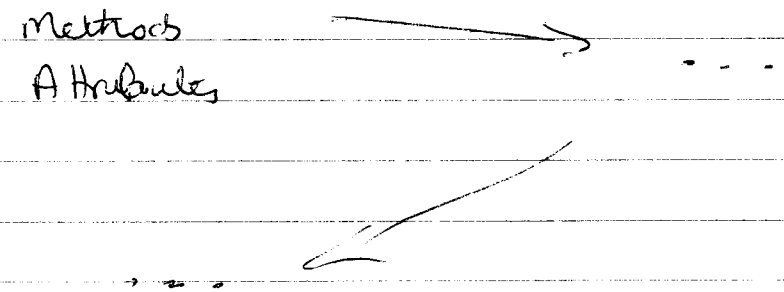
Part of Conceptual Model



Peter Coad - "Do it myself" strategy

Coupling & Cohesion.

Classes
Methods
Attributes

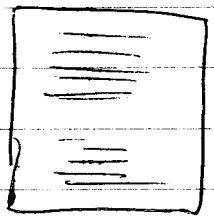


Cohesion – work together inside a class
" is a property inside a class

Coupling – working between classes

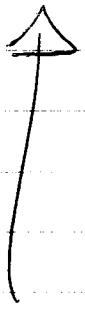
WHAT & HOW !!!

Extremes :-

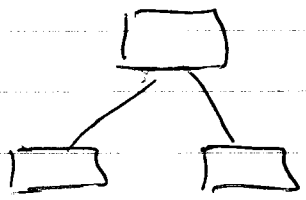


to much in one class

to little requires lots of classes



What we want



High Cohesion
Low Coupling

△ Cohesion

When the elements of a class (component) all work together to provide some well-bounded

△ Coupling

Assign responsibility to keep <u>coupling low</u>

- A connected to B
- A has knowledge of B
- A relies on B

* Cohesion ← Trade off → Coupling

CS4504 Tuesday 14th January 2014

Term 1

SE - Process

- How to do it - agile / risks etc
- UML Model based process (key -> code)
- Verification / Validation / Testing
- Quality

Term 2

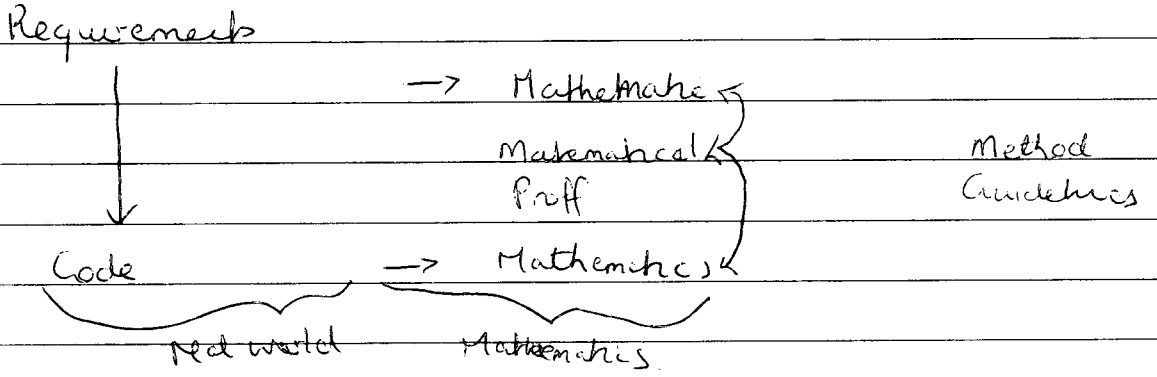
System Engineering - Correctness of Systems

- General - architecture
- Distributed systems
- Non functional properties
- Cloud / Small dis. sys.
- Big data
- Real time systems

- Correctness of Systems (starting point)

Labs:

- Google App Engine
- Azure Compute Emulator
- IBM BigInsight (Hadoop)
- Dropbox -> Amazon S3



2 Issues
- Bugs
- Demonstrably correct - does that it is correct.

The mathematical way will allow us to show that there are no bugs. We can prove it.

It may not always be the Java code but the compiler code that has the bugs There is a link between mathematical & real world as the real world must work!

Logic : an informal introduction.

General Principles
- True premises (Facts basic accepted truths on which argument is based)
- Valid argument. (or rules)

Facts + Rules = new Facts

Calculus

$1 + 3 = 4$ ← Symbolic

$x + y = y + x$, Calculus

$A = T \quad , B = F$

$T \wedge F = F$

Boolean logic can be calculated:

OR , AND

CS4504 Thursday 16th January 2014

Labs - 3-5pm Wednesday

Not finish Report → study day Break

F	F	T
T	T	T
F	F	F
F	T	T

Expert System in Prolog

:- = implies

, = AND

Syntax → ∧, → , syntaches — Truth Table
Syntatic Transformation

"¬(X ∧ Y)" ⇐> "¬X ∨ ¬Y"

This is a syntactic rule.

eg: "A ∧ A" ¬¬A
⇒ "A" A

Tautology = if something always simplies to true

Normal Form => Conjunctive Normal Form (CNF)
=> Disjunctive Normal Form (DNF)

Predicates - an argument or more
eg isClosed (valve 1) -> Property
eg aslessthan (tempNum, 50) -> Relation

$\forall x$ (isMan(x) -> IsMortal(x))
for all x, if x is a man then (may) he is mortal
$\forall x$ (isMan(x)) -> (isMortal(x))

$\forall x$ for all x
$\exists y$ should be able to find a y

Review

Syntax
Smantics
Rules
Boolean Logic -> Truth tables -> Sytactic
1st order Logic -> Symmantics
Interpretation Model
Un?

Rules -> certain facts above the line
Resultant fact below the line

$\frac{A, \quad A \rightarrow B}{B}$ (Fact, Fact 2, Fact 3)
know A, can get B

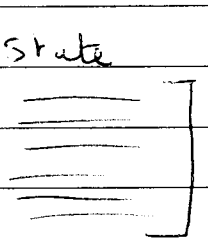
Program Verification

→ Floyd-Hoare Logic
(The Emperors Old Clothes (Hoare Turing Award)
- QuickSort (cleaner than SHELLSORT)

→ Partial and total correctness
→ axioms and rules for small programming
→ Concept of an invariant

CS4504 Tuesday 21st January 2014

Natural language Specifications
- Inconsistent
- ambiguous (allows some wiggle room - not precise)
- difficult to follow
- cannot be reasoned about / Mechanically analysed.



State	Data		Unsorted list	- what to do
	Program		Program	- How to do it
State	New Data		Sorted list	- Result / Done.

Floyd-Hoare Logic

[ ] = must terminate

[X = 1] Y := X WHILE T DO SKIP [Y = 2]

↑ never terminates because it is always true.

Auxiliary Values

$\{X = x \wedge Y = y\}\ \underbrace{R := X,\ X := Y;\ Y := R}_{\text{Program}}\ \{X = y \wedge Y = x\}$

R := X ; assign X to R

Q := 0 ; assign 0 to Q

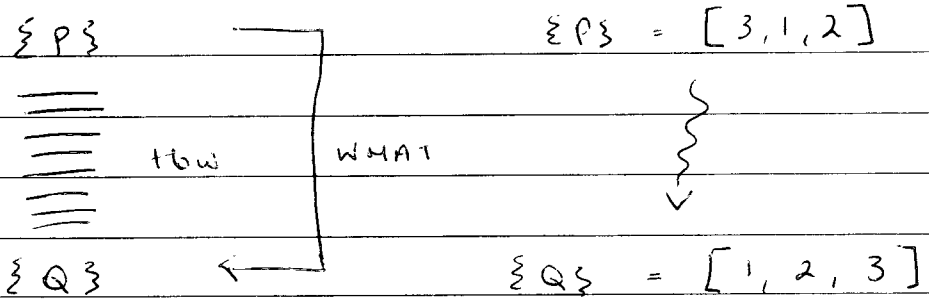
Axiom = a sentence that is asserted to by true without proof. (aka: postulate)

Rule -> $\frac{A \quad , \quad A \to B}{B}$

$$\frac{}{A \wedge B = B \wedge A}$$

Don't need anything above the line because it is always true.

Axiom is always true, $A = A$, $B = B$, so $A \wedge B = B \wedge B$.



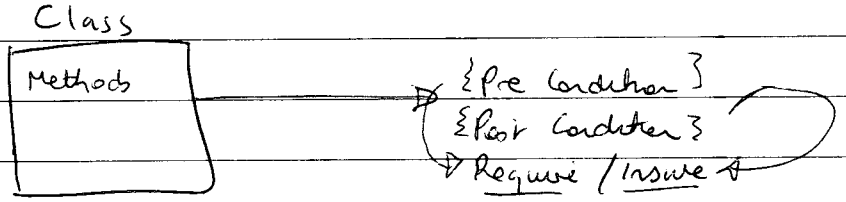
Compare How with WHAT.

Go line by line and find the meaning of the line.

How = Code / Implementation

CS4504 Tuesday 4th February 2014

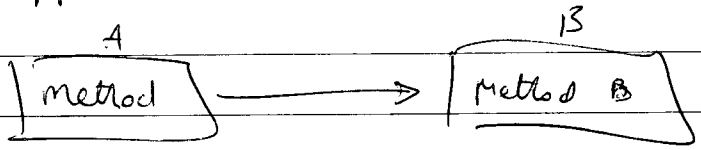
Design by Contract



$\{P\}\ C\ \{Q\}$

Pre-Condition — set before → ensured by the first programmer
B ensures precondition true
before calling method.

Post-condition — supplier i.e. A



△ Fail fast — cannot check (validate ie. $0 < d < 360$
you can't check if it is, it will just fail
when you try to do something.

D by C — Only one check

✸ Contrast with Defensive Programming (D by C vs DP)
- Completely the opposite
- Loads of checks in DP.

You can use assertions in any language

D by C – loads of documentation
Pre & Post Conditions

There are various attributes inside a class
- a class is cohesive
- some relationship between each other
so they can't have arbitary values
Example: keep a running total instead of doing calculations when required – just call the total.

* 0 Possible Exam Question

Invariants

It is easier to keep a list of what is not allowed eg ¬(can't to this), or ¬() ∧ ¬( )
never this ∧ this combination eg (stopped) (moving)
can only be at one state at once

Another example is traffic lights: (red) (orange) (green)
(r) ∧ (o) ∧ (g) and you also have (out of order)
4 boolean methods

(r) ∨ (o) ∨ (g) ∧ ¬(out-of-order) ∧
¬(r ∧ g) ∧
¬(r ∧ o) ∧
etc.

Method: Turn to Green
Pre-Cond: (¬ green ∧ ¬ out-of-order)
Post-Cond: green ∧ ¬ o-of. o

∧ ¬ red ∧ ¬ orange
Don't need this as it is covered in the not invariant list

CD Player Example

Play
Pause
Stop
Open
Close
CD_Present boolean
Out of Order

Train door class : Open, Moving, Locked, AtStation

Lift :

△ Benefits of D by C - see slides

- Ariane 5 - pre-cond missing

---

*1 $\{P\}$ C $\{Q\}$

↑ what (Pre) ↑ How Program ↑ what (Post)

[2, 1, 4, 5, 3] ...> [1, 2, 3, 4, 5]
SORT

2 What does the program does, go through the code line by line and apply rules eg $\{P(V/E)\}$ $V := E$ $\{P\}$ for example.

Each line of code is part of the syntax. if the rules don't match the code, we use Boolean rules to fix it.

▲ Turing used ASSERTIONS to divide long scripts up into smaller sections to check it at various points

if it is ok at 1, but not at 2 we know the error is between 1 and 2.

CS4504 Wednesday 5th February 2014

C:/CS4504/BigData

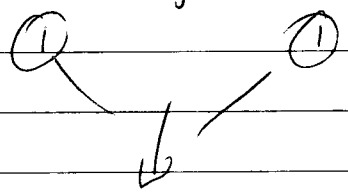
Hadoop - not great graphics

cs4504bmar1@gmail.com (Dropbox)
... *2014#

biadmin → create director → inputs.

cs-2014-code.appspot.com

~~Asc~~ ~~X~~ xyz asc



Ok for 10m or over 500ms but for 10Tb or 200+s it is impossible thats why we require distr. systems

Hadoop FS
Red Hat FS
Windows FS

HFS - Hadoop File System

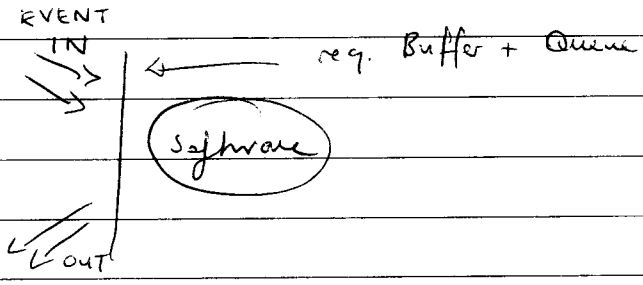
CS4504 Thursday 6th February 2014

Real-Time Software Design.

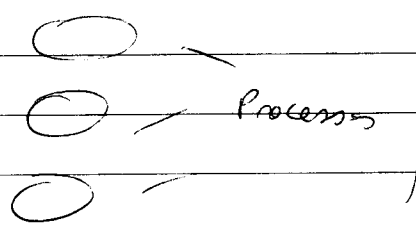
(Ian Sommerville - SE 7th Edition Chapter 15.)

- Business Enterprise (OO items)
- Low level Embedded Real Time (C, FORTRAN etc) → In UML
- People really use a subset of the notations
  - Use Cases
  - Sequence Diagrams
  - State Charts
  - UML

Real-Time — Soft — not end of the world
— Hard — is " " " "



To solve the events issue, use priority.



We also need interrupts as well as priorities

We need a real time operating system
Real-time scheduling, → interrupts + suspended.

Apple - 1 process - Tiny OS for ~~end~~ devices
is one process thread can handle
everything so its much faster.

Real - Time
Embedded - eg sensor to open gate. (high 90's of systems are embedded)
Reactive - reacts to events. (Hard RT)

ARM - design
low powered
processor.

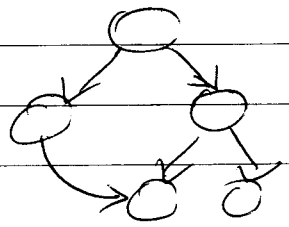
Reactive

- But how react depends on state
  eg Air Traffic Control => Bank Left., what state is the plane in at that point.

eg Telecoms => incoming call => off the hook, idle. call in progress etc. There are various states.

If / else
Case event :

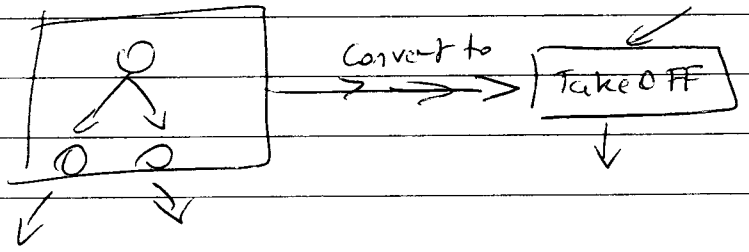
State Charts vs Code => SC more compact



State Charts - By HAREL. (1990's)
(David Harel)

↓

Israeli Fighter Jets.



Hierarchy.

Ignore all the smaller things and create a hierarchy for each sector.

data → any additions for example x += Y;

Control → if we use the data for something

eg if (x = 100) { }

△ Harel Reactive Animation .pdf.

△ Priestly Statecharts Slides .pdf.

Report

Main Body - 60 pgs.
→ Shots / work. → appendix.

Intro. - Background -
↳ Domain
↳ Software Support Mgt.

Bigger Picture - general.
↳ Issues of MgS - basic info.
↳ Technologies used. - MySQL ✗, javascript
↳ related research → Comp / Market.
similar research projects

Main ① → Design / Features diff. features - (Trans Incl. / 1 page each)
② → Implementation
→ Security features. ↓ specifics Code!
→ Tech. Know /
→
→ Testing / Evaluation
→ Coach.
→ Summary
→ Our Discuss.
↳ Limitations
↳ future work.
↳ weaknesses

Username / Password for users.

Appendix → Detailed desc.
App. on each Major feature - more detail.

Prompts - Tasks

CS4504 Thursday 13th February 2014

Read-Time UML.

✡ Benefits of State Charts. State Machine/Chart

- Hierarchy
- State chart is more general, a state machine is specific
- Guards (Boolean conditions)
- History State
- Reduce entry and exit. into these modules.
- Much more compact (less code to go through)
- flow control also better
- It is not tied to any language - is independent of its code.

○ Rhapsody Code Generating <IBM>

NEW TOPIC

✸ Architecture

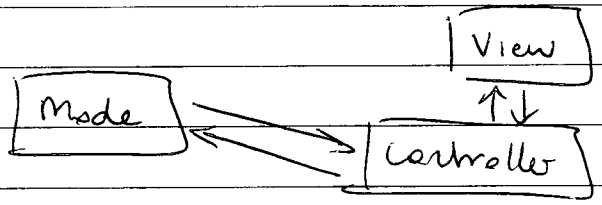
- Sean Garland (1990's)
- Software Architecture

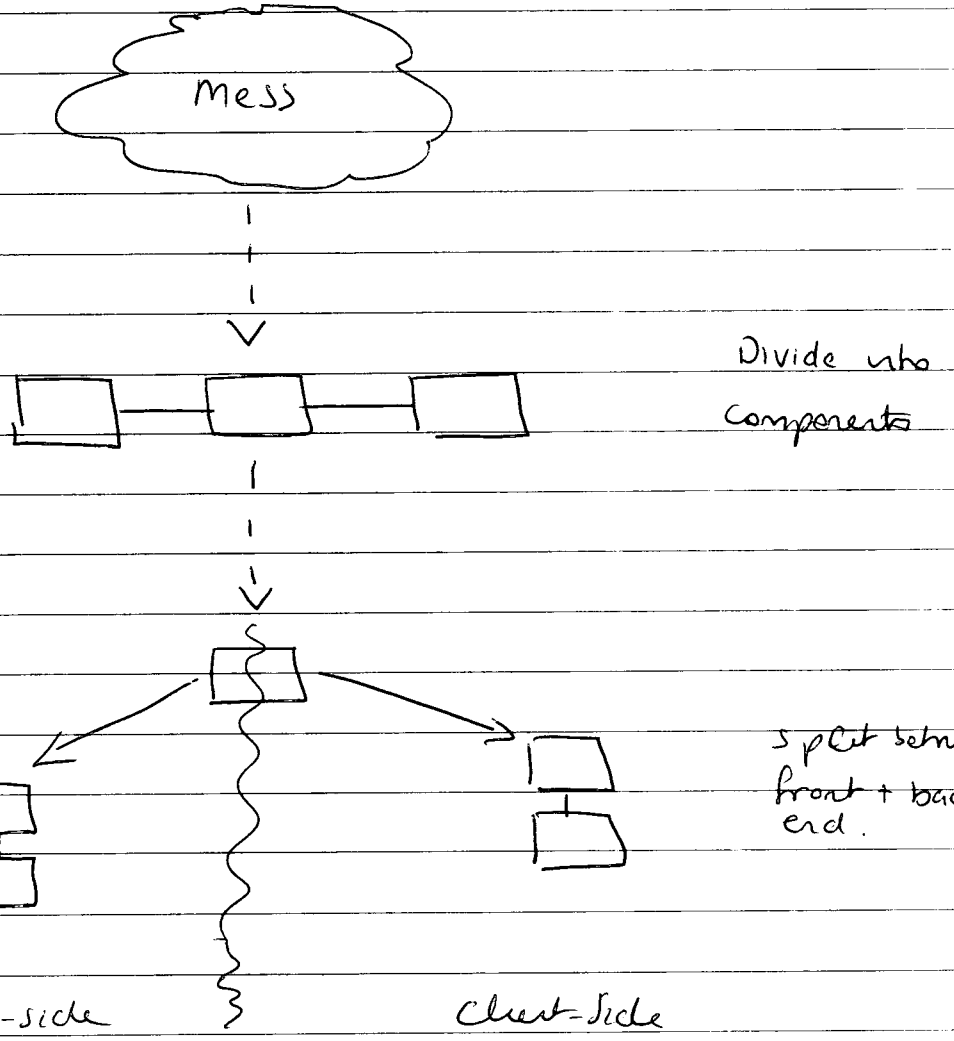
What is SA?

- Design - High Level Design
- Breaking the system down into components (blocks)
  - Handlers } ▷ Functions } Connected to design patterns
  - Methods

These are described as higher level than classes.

- You'd also have data, communication, control that happen between different components.
- MVC :- Model View Controller
  - Model :- Code / Data / State / Functions
  - View :- Interface functionality (kept separate from background/model)
  - Controller :- links Model & View





- Security / Secrecy
- Processing Load
- Communication Load
- Memory Load

} Reasons why things are kept in different areas

Cache Data + Performance
− Security

S.A. is all about Non-Functional Properties.
- Easy to understand, easy to modify.

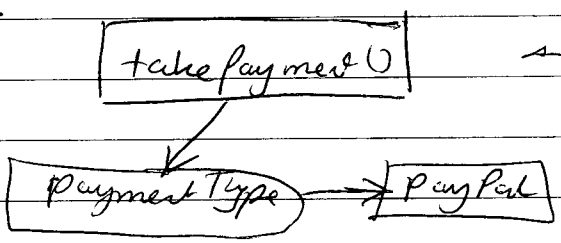
▲ Trade off = when one goes up / the other goes down.

✱ Parna's Criteria (David Parnas)
- Self-contained modules
- Put a design pattern to erase change if required.

○ GRASP - General Responsibility Assignment Software Patterns / Principles.

Protected Variation :- constrain how code will vary, allow changes elsewhere, but not everywhere.

Strategy :- have an abstract class, define sub-classes later.



← Abstract Class

Software Architecture

- Large scale components rather than lower level components
- Abstract level of design
- Reuse good design idioms
- Component based re-engineering
- System evaluation pre-implementation — can be done first.

Not only what but why

CS 4504 Thursday 20th February 2014

✸ Larman Principles
- Protected Variation
  - → Limit the amount of scope for change.
  - → "Leave scope but limit it."
  - → Force people to define what can be changed.
  - → Maintainability

△ KWIC - Key Word in Context Case study.

- Tale of two cities
- City of God
- God's Book

Tale ..
Two Cities ..
Cities

City ..
God ..

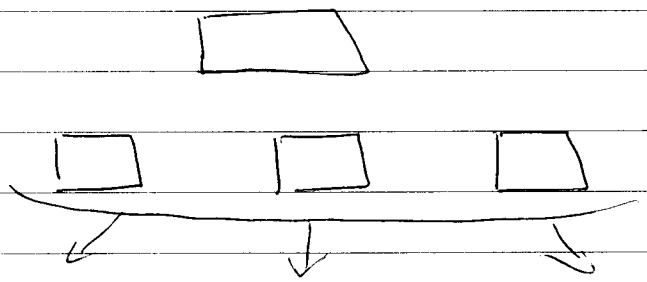
Rotate words and index first word.

✸ Model - View Controller (MVC)

ExamQ? - Know pro's & con's

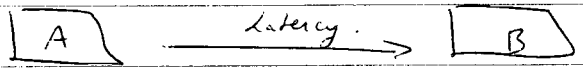
△ Distributed Systems

Distributed Software Engineering



Disadvantages

- local vs Distributed
  - → Costs
    - → Speed
    - → Security
    - → latency



functionality / Data — functionality / Data

- Keeping some things on the client
  - → perform some tasks there
  - → verify between server / client is bad.
  - → Compute locally
  - → Performance
  - → Cost
  - → Risk

TRADE - OFF
Increase A - Decrease B

Communication Cost.

Distributed Systems are generally internet based, but there are fixed distributed systems for example access control, traffic systems and fire systems.

CAN

Embedded Distributed Systems

One way of achieving transparency is to have some level of indirection.

CS4504 Tuesday. 11th March 2014

Scaling Out - change code / spread load out.
split up databases / modules.

Scale Up - buy more machines / memory
to store more data and keep going
on and on.

Website - high scalability
→ Real Life Architectures.

Shard - Splitting up your database into
places where you think it is required.
De-normalise the database - spread data
out to optimise - reduce latency
and balance load across the shards.

× C: Strong Consistency
✓ VIP → A: High Availability
P: Partition Tolerance
↳ Integrity of system properties is ensured.

The above are desired properties but only
two are achievable for example A & P
but low or no C.

Brewers CAP Theorem !!!

Size of data
- sharding
- partitioning

Latency
- Replicate database
- Caching

Consistency
- enforce consistency — timestamps