

Class 13 CS540

Gregg Vesonder
Stevens Institute of Technology
(© 2005 Gregg Vesonder)

Class 13

1

Roadmap- Class 13

- Clarifications from last class
- Log Book volunteer
- Brooks Chapter 15-Epilogue
- Formal methods
- Clean room design
- Software Factories
- Software Archeology
- Outsourcing
- Parnas
- Best Current Practices, OA&M, Life Cycle Software Engineering, Software project Reviews
- Next week: SCRUM, special topics, review

Class 13

2

Clarifications

- Apple and the mach kernel

Class 13

3

Logbook

- Canary in the coal mine, project indicators

Class 13

4

Brooks No Silver Bullet (1986)

- **Accidental vs. essential** (abstract conceptual structures).
Suggests:
 - Exploit mass market - buy vs. build, recurrent theme
 - Use rapid prototyping for establishing software requirements
 - Grow software organically (incrementally)
 - Identify and develop the great conceptual designers
- **Software projects as werewolves - turning from the familiar to the horrible**
- **There is no silver bullet**
 - Unfair to compare software and hardware
 - The essence are inherent difficulties, accidents not inherent. Essence is interlocking concepts of data sets + their relationships + algorithms + invocations of functions

Class 13

5

NSB -2

- **Hard part of software is specification, design and testing of the conceptual construct** not representing and testing its fidelity
- **Inherent properties include:**
 - **Complexity** - scaling software is not simply enlarging size of elements, rather increases # of different elements and their interaction
 - Math models will not work because you cannot abstract out the complexity - it is part of the essence
 - Complexity causes:
 - Difficulty communicating among team members
 - Difficulty in enumerating all possible states
 - Unvisualized states resulting in security trap doors
 - Difficulty in providing overview

Class 13

6

NSB-3

- Additional Inherent Properties:
 - Conformity - unlike physics with underlying laws, **software reflects the arbitrary complexity of human institutions** and systems and conformity to the interfaces of other systems
 - Changeability - software is constantly subjected to pressures to change:
 - Successful software is expanded
 - Exists longer than the machine vehicle for which it is written
 - Embedded in a changing cultural matrix of applications, machines, laws, organizations, ...
 - Invisibility - hard to visualize in a single space

Class 13

7

NSB-4

- Breakthroughs solved many **accidental** difficulties
 - High level languages
 - Time sharing and now the network

Class 13

8

Silver Bullet "Hopefuls"

- OO - has a chance
- ADA and other languages - forces training in modern design techniques
- AI - no common technologies (that has changed a bit)
- Expert Systems - separates application complexity from the program (very underused currently)
- Automatic programming, specification to code, sees little hope
- Graphical programming - screens too small, software hard to visualize, may work for tiny, application specific languages
- Program verification - does not save labor - "much of the essence of the program is debugging the specification"
- Environments and tools - helps to keep track of details, includes integrated project databases
- Workstations - great but no magic, although it does seem magical at times

Promising Approaches to CE

- Buy vs. Build - development of mass market is the most profound trend - also puts simple programming power in the hands of users, (simple programming is becoming a skill - just as people became literate ...)
- Requirements Refinement and Rapid Prototyping
- Incremental Development - **grow not build**, helps morale, instant feedback
- **Great Designers**, great designs come from great designers

Exciting Products - Brooks

- Great Designers - Yes
- UNIX
- APL
- Pascal
- Modula
- Smalltalk
- Fortran
- Committees - No
- Cobol
- PL/1
- Algol
- MVS/370
- MS-DOS

Great Designers

- Software is a creative process
- Ways to grow great designers (too little practiced these days):
 - Identify top designers early on
 - Career mentor them
 - Plot a career development path with apprenticeships and opportunities
 - Encourage peer interaction with other great designers

NSB Refired

- (review of critiques of NSB)
- Some tidbits:
 - On **complexity** - if you clean up human organizational structures you do simplify but complexity is also due to data structures and modules
 - Optimistic vs. Pessimistic vs. Realistic - I am a developer therefore I am innately optimistic!
 - Harel claimed conceptual stuff is topological and therefore graphical - Brooks agrees that you can represent software in a collection of representations but disagrees that they will be useful thought and design aids
 - Capers Jones - wrong to focus on productivity, **focus on Quality** and productivity will follow

Class 13

13

Larger Tidbits: OO

- Many OO concepts have a history but OO brings them together:
 - Enforces modularity
 - Emphasizes modularity and hides the details
 - Provides inheritance and a hierarchical structure
 - Strong abstract data typing with the benefit that a particular data type will only be manipulated by appropriate operations
- Why has OO grown so slowly (dated, Java changed things)
 - Parnas, because OO has been tied to complex tools rather than design
 - Brooks, upfront costs are substantial benefit back loading, slow to accrue benefits, 5th project goes real fast (java/web changed this)
 - Requires managerial courage

Class 13

14

Larger Tidbits: Reuse

- Programmers reuse their own code (and friends)
- Mathematical software is heavily reused but it is a special case
- DeMarco - big expense to reuse, Yourdon quotes factor of 2 (1.5 to 3).
- Large class libraries, vocabulary is an impediment, simply too many things. Helpful aids:
 - Examples of composed products (not JAVADOC)
 - Learn vocabulary by use
- After all the critiques, no change in his premise - **complexity is our business and it does limit us**

Chapter 18

- Summarizes all the chapters - good study aid

MMM after 20 Years

- Really addresses how people in teams make things
- Central argument - *conceptual integrity and the architect*
 - System must be conceptually coherent to the single mind of the user, yet designed by many minds
 - The architect forms and owns the public mental model of the product
 - Separate the architecture, the definition of the product as perceived by the user(s) from its implementation, this boundary is within the design task, user facing vs implementation facing
- Recursion of architects - large projects have a master architect and the master architect partitions the system into subsystems with minimal, easily designed interfaces, each having its own architect
- Having a system architect is the single most important step to achieving conceptual integrity

Class 13

17

MMM Retro - 2

- Second system effect
 - Mass market particularly susceptible to featuritis over releases, e.g., Microsoft Office
 - Potential cure - frequencies, list explicit guesses for the attributes of the prospective users
- WIMP (Windows, Icons, Menus, Pointing) interface
 - Dragging and dropping, overlaying windows, trash can, folders, ... follows from the office metaphor
 - Suggests two mice (not!) and marvels at keyboard chording .. The apple command key
 - Apple and Microsoft have made it standard for apps (with coercion)
 - Promotes speech (too slow, too linear)

Class 13

18

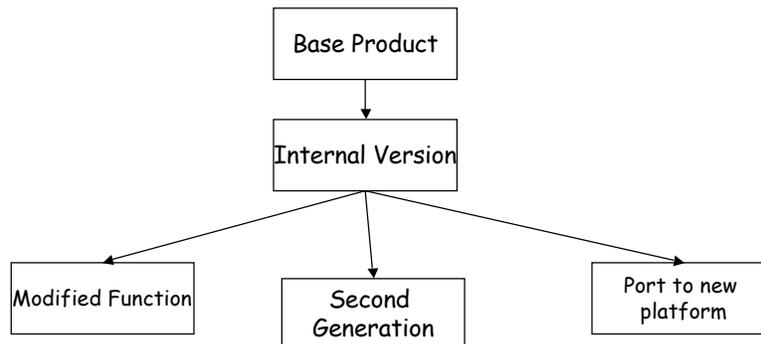
MMM Retro -3

- Felt he was too tainted with Waterfall model
- "The basic fallacy of the waterfall model is that it assumes projects go through the process only once" (but it did define the stages)
- Fallacies of the waterfall model: places testing at the end and assumes one builds the whole system at once
- Again promotes incremental model
 - Permits early user testing
 - Permits build to budget strategy, albeit with some functional shortfall - but what is there works
- Parnas families concept - designing software project as a family of related products - near the root less likely to change

Class 13

19

Parnas Family Concept



Class 13

20

MMM - Retro 3

- Microsoft recommends build every night - (we do)
- Rapid Prototyping technique - [Wizard of Oz](#)
- Parnas was right about information hiding
 - Programmers are most effective if shielded from the innards of modules not their own (all in moderation)
 - More robust in a design for change (but ...)
- How Mythical is the Man Month - hardly any projects succeed in 3/4ths of calculated time regardless of how many people are deployed

Class 13

21

MMM-Retro 4

- How true is Brooks's Law : adding manpower to a late software project makes it later - most looked for **remedies**
 - Add manpower as early as possible
 - Focus on strategies for incorporating new folks
 - Brooks: none have addressed the added communication links
- People are (almost) everything
 - Quality of people and their organization and management are the major factors, COCOMO model quality of people is strongest factor
 - Managers must enhance creativity
 - Delegating power down

Class 13

22

MMM-Retro 5

- Biggest surprise over 20 years - millions of computers
 - Creativity enhancer
 - Predicts rise in virtual environment systems
- Shrink wrap software
 - Focus on 5 (now 4) operating systems: MVS, DEC VMS, UNIX, Windows, MAC
 - Development stars in small companies

Class 13

23

Buy and Build - Shrinkwrap Software

- My UNIX experience
- Metaprogramming, e.g., HyperCard - web and HTML replaced this sort of thing
- Shrink wrapped products need to make products easy to incorporate in larger systems - has happened in strange ways: databases as meeting place, shells stitch things together
- 4 levels of users for shrinkwrap:
 - As is user
 - Metaprogrammer - templates of functions
 - External function writer - adds functions to applications
 - Metaprogrammer using several applications in a larger system

Class 13

24

Software Engineering Concerns

- Remain the same
- How to design and build a set of programs into a system
- How to design and build a program or a system into a robust, tested, documented and supported product
- How to maintain intellectual control over complexity in large doses

Formal Methods

- **Formality = degree of mathematical rigor during analysis and design**
 - Mathematically based techniques for describing system properties
 - Strive for consistency, completeness and lack of ambiguity - the usual suspects
 - Issues with less formal approaches: contradictions - statements at variance with each other, usually separated by substantial text in the requirements, ambiguities, vagueness, incompleteness, mixed levels of abstraction
- Mathematics provides a high level of validation

Formal Method Concepts

- Data invariant is a set of conditions that are true throughout the execution of the system that contains a collection of data
 - Example: constraint on table size in a symbol table = maxids and all items are unique names
- State - stored data that the system accesses and alters
- Operator - an action that reads or writes data to a State, e.g. adding and removing names to a symbol table. Operator is associated with two conditions:
 - Precondition - defines circumstances in which a particular operation is valid
 - Postcondition - defines what happens when an operator has completed its action defined by its affect on state
- Brainstorming techniques are useful to develop a data invariant for a complex function, e.g., bounds, restrictions and limitations

Class 13

27

Set Preliminaries

- Set - collection of objects and elements, all elements are unique and order is immaterial
- Cardinality - number of items in a set
- Sets are defined by enumerating elements or using a constructive set specification
 - $\{n:N \mid n < 3 \cdot n\}$
 - $n:N$ is signature, specifies range of values considered
 - $n < 3$ is predicate defines how set is constructed
 - n is term provides general form of the item of the set, when n is obvious it can be omitted
 - $\{0, 1, 2\}$

Class 13

28

More Set Prelims

$x \in X$ denotes membership in a set

$12 \in \{6, 1, 2, 12, 22\}$

$x \notin X$ is not a member of X

\emptyset is empty set

$\emptyset \cup A = A$ and $\emptyset \cap A = \emptyset$

union, cup

intersection, cap

contained in relationships

Union combines both sets with duplicates
eliminated

Intersection provides list of common
elements

29

Yet More Set Prelims

- \setminus is set difference operation removes elements of 2nd from 1st
- X is cross product, each of the elements of the 1st combined with each of the elements of the 2nd
- Powerset is collection of subsets of set
- Logical operators
- Universal quantification
- Sequence, elements are ordered, 1st element is domain, 2nd element is range
- Sequence operators: concatenation, head, tail, front, last

Class 13

30

Example: Block Handling

- Blocks of storage held on a file storage device
- State is collection of free blocks, collection of used blocks and queue of returned blocks
- Data Invariants:
 - No block will be marked as used and unused
 - All sets of blocks held in the queue will be subsets of the collection of currently used blocks
 - No elements of the queue will contain the same block numbers
 - Collection of used and unused blocks is the collection of blocks that make up files
 - The collection of {used, unused} blocks have no duplicate block number
- Operations: adds a collection of blocks to the end of queue, checks whether queue of block is empty

Class 13

31

Block Handling - 2

- Blocks = set of all block #s
- AllBlocks is the set of blocks between 1 and MaxBlocks
- 2 sets, Used & Free
- Invariants:
 - $Used \cap Free = \emptyset \wedge Used \cup Free = AllBlocks \wedge \dots$
- Operation, remove a block from head of queue
 - $\# BlockQueue > 0$: precondition
 - Postcondition: head must be removed and placed in Free and queue adjusted to show removal
 - $Used' = Used \setminus head\ BlockQueue \wedge Free' = Free \cup head\ BlockQueue \wedge BlockQueue' = tail\ BlockQueue$

Class 13

32

Formal Specification Language

- Three components:
 - Syntax defining specific notation of specification
 - Semantics to help define a universe of objects used to describe the system
 - Set of relations that define the rules indicating which objects properly satisfy the specification
- Syntax is usually derived from formal set theory
- Semantics abstraction usually are things such as states, state transitions, ...

10 Commandments of Formal Methods

- Choose the appropriate notation - that is good match for application
- Formalize but not overformalize - not necessary for every (most) aspects of the system
- Estimate costs - formal methods have high startup, training costs
- Know a formal guru - for consultation
- Keep your standard development methods
- Document sufficiently including natural language commentary

10 Commandments (cont'd)

- Maintain quality standards - FM do not supplant quality efforts
- Do not be dogmatic - you will still get bugs
- Test, test and test again - does not replace testing
- Reuse

Cleanrooms

- Cleanroom techniques - build correctness into software as it is being developed
- Instead of analysis, design, code, test and debug
- Write code increments correctly the first time and verify correctness before testing - Do it right the first time
- Verification is through mathematical techniques
- Testing certifies software reliability

Steps in Cleanroom Design

- Analysis and design uses a box structure notation
 - Box encapsulates some aspect of the system (or entire system) at a level of abstraction
 - Correctness verification is done once the box structure design is completed
 - This is done in place of unit testing
- Testing is done by defining set of usage scenarios and probability of use for each scenario, then defining random tests that conform to the probabilities
- Error records from testing are analyzed to compute reliability
- Heavily incremental approach

Software Factories

- Applying factory techniques to software development emphasizing process, measurement and reuse (Toshiba's view)
- Software Workbench - integrated environment that supports all workers in factory - includes programming, debugging, configuration, test, requirements, documentation, Project management, quality assurance and reuse
- Uses waterfall model
- Project is divided into unit workloads with daily and weekly status, tracking actual vs expected, provides feedback and identifies issues
- Heavy Quality emphasis
- Reuse is the single most critical issue in improving quality and productivity
- Quality circles - voluntary groups of works that focus on improving quality, process, ...

Software Archeology

- Trying to understand what was in the minds of software developers using only artifacts left behind
- Hampered by the fact that the artifacts were not created to communicate to the future
- Only part of what was originally created has been preserved
- Relics from different eras (versions) are intermingled
- Dynamic and static techniques
- Support for refactoring
- Browse the workshop!

Class 13

39

Outsourcing

- Should you outsource?
- Outsourcing strategies
- Outsourcing tactics/plan
- Cultural issues in outsourcing (and in general distributed development)

Class 13

40

Outsourcing Definition

- Contracting of various information system functions such as managing of data centers, operations, hardware support, software maintenance, network and even application development to outside service providers.

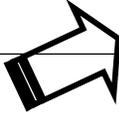
Should you outsource?

- "no one really likes outsourcing, even well run initiatives" (<http://www.cio.com>)
- Outsourcing in combination with offshoring can get more value and reduce staff
- It can be leveraged to improve process discipline
- But root causes will still remain - e.g., lousy diagnostics will not make a help desk any better
- Privacy, security and reliability concerns must be addressed

Outsourcing Strategy

(Kishore, et.al. 2003)

	Low Strategic Importance	High Strategic Importance
High Control by Service Provider	Reliance	Alliance
Low Control by Service Provider	Support	Alignment



Class 13

43

Outsourcing Plan

- Do a vendor assessment before contract is signed
 - Process, Technology, Operations audit (Reliance, Alliance)
 - Does vendor have privacy policy limiting data sharing
 - **Does the vendor subcontract?**
 - If live feed, proof that there are access controls, identity management and authentication in place
- Audit of privacy and security practices
- Senior executives must consent to sign a compliance pledge for privacy, security, process, ...
- Complete due diligence
- Outsourcing should not be treated as a simple IT contract but as relationship management

Class 13

44

Cultural Differences

- Exist in both collocation and virtual collocation
- Culture is acquired, it helps people read world signals, artifacts gestures. It molds the way people think and are motivated.
- Many kinds of culture - national, regional, organizational, avocational and generational
- Affects how projects are formed and managed

Class 13

45

Dimensions of Culture

- Hofstede (IBM employees)
 - Revering hierarchy
 - Individualism vs. Collectivism
 - Task (Japan, Germany, US) vs. Relationship (France, Russia, Netherlands) focused
 - Risk avoidance (Japan, Russia vs US, India)
 - Longterm orientation
- Hall
 - Space
 - Material goods
 - Friendship (transitory or lasting)
 - Time
 - Agreement
- Low context (US) vs. High context cultures

Class 13

46

Issues

- Team composition:
 - Short term teams difficult in high context cultures
 - Attribution of team mates - recognize skills (misattribution of dress in videos)
 - Motivation- money vs time off
- Teamwork
 - "microstructure of conversations" eye contact, tone of voice (soft, respect; loud, control)
 - Planning: buy in vs. authoritarian
 - Decision making: past, present, time/cost/quality
 - Argumentation styles: democracy versus authority
 - Use of time: abrupt meetings vs. slow chatty
 - Virtual collocation (high context need video, IM is an advantage (slowness issue))
 - Brainstorm - anonymity
 - Time of day: (Israel, Sunday-Thursday, French 35 hr week, Holidays)

Class 13

47

Steps in Collocation

- Awareness of cultural factors
- Awareness of specific cultural values
- Adjust to suit others, understanding is not enough
- Establish a management communication covenant detailing how the team will be managed and how the team will communicate - just as important as agreeing on development rules and conventions!

Class 13

48

UE - Undesired Events

- **Basis for exception handling**
- Always aspects of a program's execution environment that do not behave as we wish - if you will, defensive programming
 - Arises from the "normal" behavior of the real world
- Goal is to anticipate what can go wrong and make (the possibility for) accommodations in advance that do not mess with the structure
- Basis for variants of throw and catch

Class 13

49

More on UEs

- Differentiate from errors that should be corrected
- UEs demand an evolutionary approach - they evolve as experience with system increases
- Responses to UEs include:
 - Self diagnosis
 - Saving of partial results
 - Printing of diagnostic Information
 - Retrying
 - Use of alternate resources
 - Job cessation only occurs as a last resort
- Traps keep code for UE separate from code for modules, lexical separation of normal use, from detection and correction procedures

Class 13

50

Classes of UEs to detect

- Limitations on values of parameters
- Capacity limits
- Requests for undefined information
- Restrictions on order of operations
- Detections of action which are likely to be unintentional
 - "open" when opened, but this could be overloaded by convention
- Errors of mechanism - failure of modules is more difficult than failure by applicability condition

Class 13

51

More on UEs

- Assign priorities to these traps usually more than one is active
- Try to consolidate UE routines, not one for one
- Eliminate some of the redundant checks as data is passed around for efficiency.
- Incidents, UEs that were expected and where recovery attempts were successful versus crashes

Class 13

52

And the Rest

- Best Practices
- Life cycle software engineering - refactoring
- OA&M
- Project reviews

Class 13

53

Thought Problems

- What is the most significant development in the software world since you became a software student/professional?

Class 13

54

References

- Larry Poneman, "Practice safe outsourcing," <http://www.darwinmag.com>
- Kishore, et.al. "Relationship perspective on IT outsourcing," *CACM*, December 2003
- Olson, J.S. and Olson, G.M. "Culture surprises in remote software development teams" *QUEUE*, December-January 2003.
- Brooks
- <http://www.visibleworkings.com/archeology>.