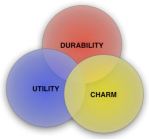
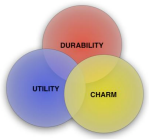


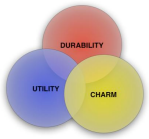

# Class 13 SSW 565

Gregg Vesonder
Stevens Institute of Technology



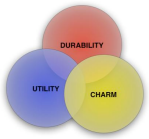

# Roadmap

- Logbook
  - Me
- Open source licensing
- Systems Thinking
- Outsourcing
- Wrap up
- The Final
- Stay in touch with the blog and the web site!



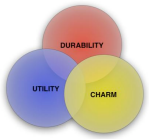

# Key Dates

- Final – tonight hand in by Thursday, July 23rd



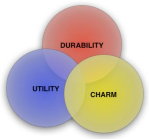

# Clarification

- Overloading is one type of Polymorphism
  - Polymorphism - process objects differently depending on their data type or class
  - Overloading (parametric) (number and type of arguments) --many methods same class vs overriding (inheritance based) a superclass method -- many methods different classes.



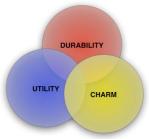

# Clarification: IPv6

- 2001:0db8:85a3:0000:0000:8a2e:0370:7334
  - $2^{128}(3.4x10^{38})$ vs $2^{32}$ >>> mass of earth in grams
- Very different
- Point to point
- Dual stacked for a long time
- From tunneled IPv6 to tunneled IPv4 – DSTM, Dual Stacked transition mechanism



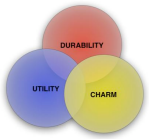

# Logbook

- Recurring Motifs



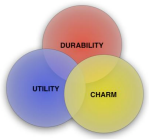

# Other Relevant Topics

- A potpourri of topics:
  - Open source including licensing
  - Systems Thinking ala Weinberg
  - Functional Programming
  - Street architecture, design and coding



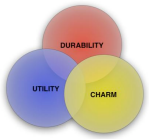

# Open Source Software Cathedral and the Bazaar

- Cathedral - commercial software world, bazaar - linux and open source
- Key names:
  - Richard Stallman - emacs, gnu, Free Software Foundation
  - Linus Torvalds, linux, open source process, open source license (General Public License (GPL), BSD, Perl's Artistic License)
  - Larry Wall - PERL



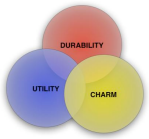

# Flavor of Open Source

- Torvalds style - release early and often, delegate be very open - VERY developer centric!
- Axiom 1 - Every (?) good work of software begins by scratching a developer's itch (does not conform to numbers in the paper)
- 2- Good programmers know what to write, great ones know what to rewrite and reuse
  - Constructive laziness
- 3-Plan to throw one away, you will anyhow
  - You do not understand problem until after first time you implement
- 4- If you have the right attitude interesting problems will find you
- 5- When you lose interest in a program, your last duty is to hand it off to a competent successor



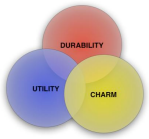

# Flavor of OS - 2

- 6- treating your users as codevelopers is your least hassle way to rapid code improvement and effective debugging
  - If you have that luxury
  - Torvalds "too lazy to fail"
- 7- release early, often and listen to customers
  - Released a new linux kernel in the early days more than once a day!
- 8- given a large enough beta tester and code developer base, almost every problem will be characterized quickly and the fix obvious to someone
  - Linus' Law "Given enough eyeballs, all bugs are shallow"
  - Delphi effect
  - Debugging is parallelizable
  - Brooks: more users find more bugs
  - Non source aware users do not provide great bug reports



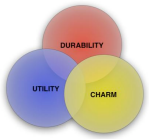

# Flavor of OS 3

- Team: Usually 1-3 core developers, beta testers and contributors in the 100s
- 9- If you treat your beta testers as if they're your most valuable resource, they become it
- 10- the next best thing to having good ideas is recognizing good ideas from users. Sometimes the latter is better
- 11-often the most striking and innovative solutions come from realizing your concept of the problem was wrong
- 12 - Perfection (in design) is achieved not when there is nothing to add, but rather when there is nothing to take away, Antoine de Saint-Exupery
  - Debugging is not only parallelizable, so is development and exploration of the design space!



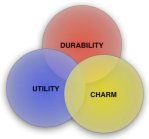

# Flavor of OS 4

- 13 - any tool should be useful in the expected way but a truly great tool lends itself to uses you never expected
- 14 -International flavor of participants is a plus in globalization (extract)
- 15-To solve an interesting problem, start by finding a problem that is interesting to you



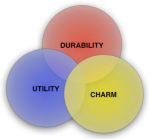

# Preconditions for the Bazaar style

- Hard to originate code in this style, test, debug, improve yes
  - You need to have something running - attractor
  - It has to run and convince others that it can evolve into something neat in reasonable time
- Leader/coordinator of Open Source project does not need to be a great designer but needs to recognize good ideas from other folks:
  - Robust and simple rather than cute and complicated
  - Community's internal market based on reputation exerts subtle pressure in self-selecting competent leaders
  - A bazaar leader must have good people and communication skills



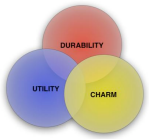

# Social context for Open Source

- Evolution of software in the presence of a large, active community of users
- Programmer time is not fungible - small number of codevelopers obeys Brooks communications links
- Programmers cannot be territorial about code, encourage others to look for bugs and improvement XXP!
- "while coding remains an essentially solitary activity, the really great hacks come from harnessing the material and brain power of entire communities"
- Internet helped
- Development of leadership style and set of cooperative customs
- Utility function of linux hackers is maximizing in their own eyes satisfaction and reputation among peers and users
- Boring is essential - software and documentation
- Open source is fun - joy as an asset
- Not so easy as to be boring, not too hard to achieve!



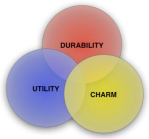

# Homesteading the Noosphere

- Open Source culture: zealotry varies, hostility to commercial software varies
- Open Source must protect an unconditional right of any party to modify (and redistribute modified versions of) open source software
- Taboos of Open Source:
  - Strong social pressure against forking projects
  - Distributing changes to a project without cooperation of moderators is frowned upon
  - Removing a person's name from the project history is not done without the person's explicit consent



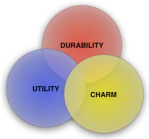

# Ownership in Open Source

- Owner of the software project is the person who has the exclusive right to distribute modified versions
- How to own:
  - Start the project - homesteading
  - Have ownership handed to you - deed transfer
  - Observe that a project needs work and owner has lost interest- try to find owner to get to have ownership handed to you - "adverse possession, moves on and improves"



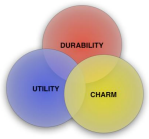

# Hacker culture as a gift culture

- Excludes crackers and warez d00dz - different culture
- Not what you control but what you give away - reputation among peers
- Along with sheer joy of making something work
- Craftsmanship model as a corollary - still linked to reputation
- In hacker culture status is based on critical judgment of peers



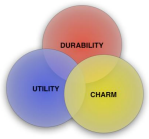

# On Reputation

- Ego is despised, yet the whole system runs on it!
- One's work is one's statement, no one attacks one's technical competence, emacs bugs not Stallman's bugs
- More prestige in founding a project than working on an existing one
- More prestige for innovative rather than me too
- Being carried in a major distribution (Red Hat, SuSE) is prestigious
- Continued devotion to hard, boring work (debug, write doc) is more praiseworthy than fun and easy hacks.



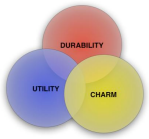

# Governance

- Project leader - codevelopers - contributors
- Apache has a voting committee
- PERL has rotating dictatorship among codevelopers



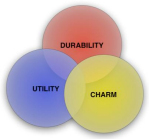

# Open Source Resources

- www.opensource.org - jump off point, Halloween papers
- SourceForge.net - project pages
- Freshmeat.net - products and product announcements
- Slashdot.org - fun




# Microsoft's response

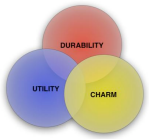
- The Halloween documents - response to Open Source, quotes Raymond
- Considers Open Source a threat especially in server market (H II - also in desktop)
- Commercial quality can be achieved!
- To target Open Source, you target a process not a product
- Open source is long term credible - you cannot FUD it!
- Implementation provides a high visibility showcase for OS
- Linux has done well in mission critical commercial environments
- Linux can win as long as services and protocols are commodities




# Open Source Licensing

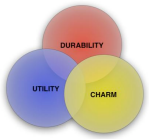
http://developer.kde.org/documentation/licensing/licenses_summary.html

| license | Proprietary sfw linking | Distribution | Re-distribution with changes | GNU GPL compatible |
|---|---|---|---|---|
| GPL | Not allowed | Not allowed | Only with GNU GPL | yes |
| Apple public | allowed | allowed | Only with apple | no |
| Apache | allowed | allowed | Allowed so long as apache is not in name | no |
| BSD | allowed | allowed | allowed | Yes (modified) |
| MIT(X11), W3C | allowed | allowed | allowed | yes |
| Sun public | allowed | allowed | Only under Sun public | no |



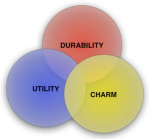

# Systems Thinking

- Weinberg's articles of faith (p.22):
  - There's no consistent substitute for a thorough understanding of your problem, though sometimes people get lucky
  - There's no solution that applies to every problem, and what may be the best approach in one circumstance may be precisely the worst in another
  - There are many useful approaches that work on more than one problem, so it pays to become familiar with what has worked before.
  - The trick to problem solving is not just "know how" but "know when" -- which lets you adapt solution method to problem not vice versa (but Kobiyashi Maru)
  - No matter how much you know how or know when, there are some problems that wouldn't yield to present knowledge, and some aspects of the problem that nobody currently understand, so humility is always in order.



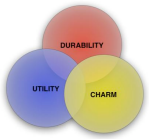

# More on Systems

- holistic approach
- Before you can count anything you have to know something
  - the count on sesame street
  - To assess, size what is in a system? Job control (scripts and the like, OA&M)? Test data/suites? Existing files, aka raw data (conversions)? Documentation (official and unofficial)? Brainware/Wetware? Explicit training? Implicit training (cubicle sheets)? Rest of Organization? Rest of World?



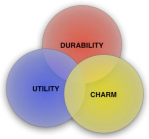

# Even more on systems

- There is no one truth about what happened
  - Ways of knowing
  - Epistemology - The branch of philosophy that studies the nature of knowledge, its presuppositions and foundations, and its extent and validity. American Heritage Dictionary
  - History is an example – Gutenberg did not invent moveable type and the printing press, he just made it all come together
- General vs. specific systems thinking
  - "It is true that horses have legs and tables have legs; but it is still better not to eat on horseback or put saddles on the table." – p41



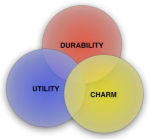

# Observing Systems

- Or observing simulations of systems, or scoping out the application (environment, current way of accomplishing task, …) part of what architect and designer should do
- "Merely looking at the sick is not observing" - Florence Nightingale
- Lessons for observers:
  - No reason to expect that a system will be fair to observers (observers change the system)
  - There may be a gold mine of information lying about, if only they know the language in which to ask
  - Systems often intentionally conceal information from would-be observers, because it may not serve the system's purpose to have too much known about it. - the model vs. the reality



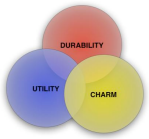

# Observing Systems - 2

- "Things are the way they are because they got that way" - it is, what it is
  - At the time it may have been a good idea
  - The developer is probably now the manager - you have to work effectively with the team - that does not mean solicitous, but not abrasive either
- "Study for understanding not for criticism"
  - True understanding is hard enough
- "Search for what is good in the old system" - it is there
- Beware the railroad paradox
- Rosenthal effect - the tendency of results to conform to the observer's expectations planaria and related stuff



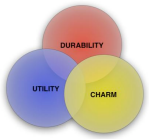

# On Design

- Learning to design is learning to generate and evaluate models
  - Biological evolution
  - Strike a balance between variation and selection
- Thing and process - both static and dynamic views
- Design for understanding
- Maintain not only form but also spirit, maintian maintainability
- Being (arch), Behaving (operating), Becoming (maintenance)



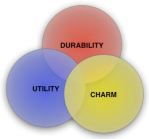

# After OO

- http://www.americanscientist.org/Issues/Comsci03/03-03Hayes.html
- Brian Hayes "The Post-OOP Paradigm"
- Has issues - but nice survey:
  - 50's languages
  - 60's-70's spaghetti -> structure
  - 80's to ??'s -> OO



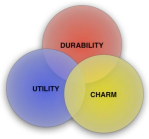

# Post-OOP trends

- Aspect Oriented Programming
- Generative programming - for constrained domains
- Patterns - programmers are like carpenters or stonemasons - stewards of  a body of knowledge gained by experience and passed along by tradition and apprenticeship
- (there are more:  functional programming, script based programming, …)
- Postmodernism from World Book encyclopedia - "example, a Postmodern society is more decentralized, fragmented, and impermanent than a Modernist society. " - not a bad description of today!



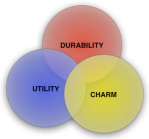

# Aspect Oriented Programming

- Gregor Kiczales - http://aosd.net
- Object orientation has limitations - has difficulty with "global constraints and pandemic behaviors"
- Concerns, properties or areas of interest of a system
  - Security & QoS
  - Caching and buffering
  - Fault tolerance mechanisms
- These scattered concerns are realized as first class elements and are cut out of the object structure
- Aspects - "mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern"
- AOP systems must provide ability to combine aspects and code into a system



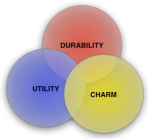

# AOP-2

- AOP use implicit invocation for invoking behavior
- 2 qualities of AOP
  - Quantification- one can write unitary and separate statements that have effect in many non-local places in a programming system
  - Obliviousness - that places these quantifications applied do not have to be prepared to receive them
- AOP focuses in one place behaviors that would be throughout the code
- Technology of combining programs and aspects: join points - where aspects interact with rest of system and aspect parameterization - customization of aspects



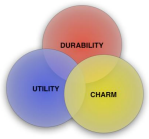

# Software Architecture Challenges

- Rediscover architecture of legacy systems or shape and rediscover as in refactoring, refactoring as active architecture rediscovery
- Architecture Archaeology
  - Many systems have no documented architecture
  - Many systems show no relationship between the architecture and the documentation
  - Many systems documentation is out of phase



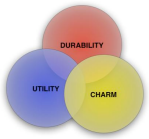

# Architecture Archaeology

- Refactoring
- Awareness of existing styles and patterns in architecture and identify areas of structural similarity that may be targets for reengineering
  - What is structural similarity?
- Move to modern patterns and off the shelf software once understanding is achieved.
  - Recall the wrapper issues -- is it worth saving/wrapping?
- Research topic includes attempts to automatically recognize or human-in the-loop recognize patterns



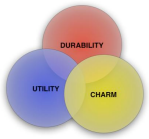

# Functional Programming

- Everything is a function.
- Entire program is a function which receives program input and delivers program's output as its result
- Functional Programming has
  - No assignment statement, variables once provided a value, never change
  - No side effects
  - Recursion heavily used
- Languages ml, caml, haskell …




# FP-2

- Use of different variants of the basic mapping function
  - (map function sequence) for example python has map, reduce and filter
  - Helps you to avoid assignment
- You can do some of this in today's languages, especially using recursion, therefore it is a design technique
  - Of course the FP folks want you to use their languages and be completely pure
- Example:
  - power (m, n)
  - if n <=0 then 1
  - else m * power(m, n-1)

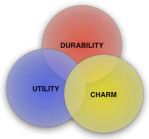



# FP-3

- In lisp:

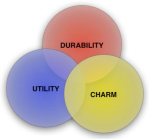
```
(defun gtvpower (m n)
     (if (<= n 0) 1
         (* m (gtvpower m (- n 1)))))
```

I have barely scratched the surface!



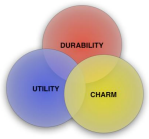

# FP- the Glue

- Supports lazy evaluation -- functions are only evaluated when it is necessary
- Simple functions are glued together to make more complex functions



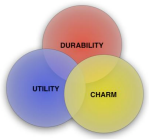

# Outsourcing

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



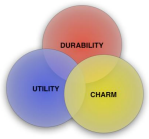

# Outsourcing Definition

- 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?

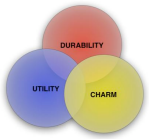
- "no one really likes outsourcing, even well run initiatives" (http://www.cio.com )
- Outsourcing in combination with offshoring can get more value and reduce staff
- It can be leveraged to improve process discipline
- But root causes will still remain - e.g., lousy diagnostics will not make a help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




# Outsourcing Strategy✓

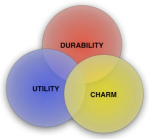
(Kishore, et.al. 2003)

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



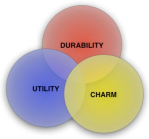

# Outsourcing Plan✓

- 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



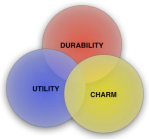

# Cultural Differences

- 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



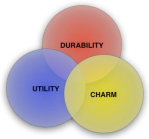

# Dimensions of Culture

- 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)
  - Long term orientation
- Hall
  - Space
  - Material goods
  - Friendship (transitory or lasting)
  - Time
  - Agreement
- Low context (US) vs. High context cultures



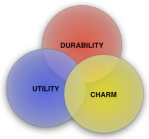

# Issues

- 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. authoritarion
  - 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)



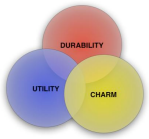

# Steps in Collocation

- 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 communicates - just as important as agreeing on development rules and conventions!

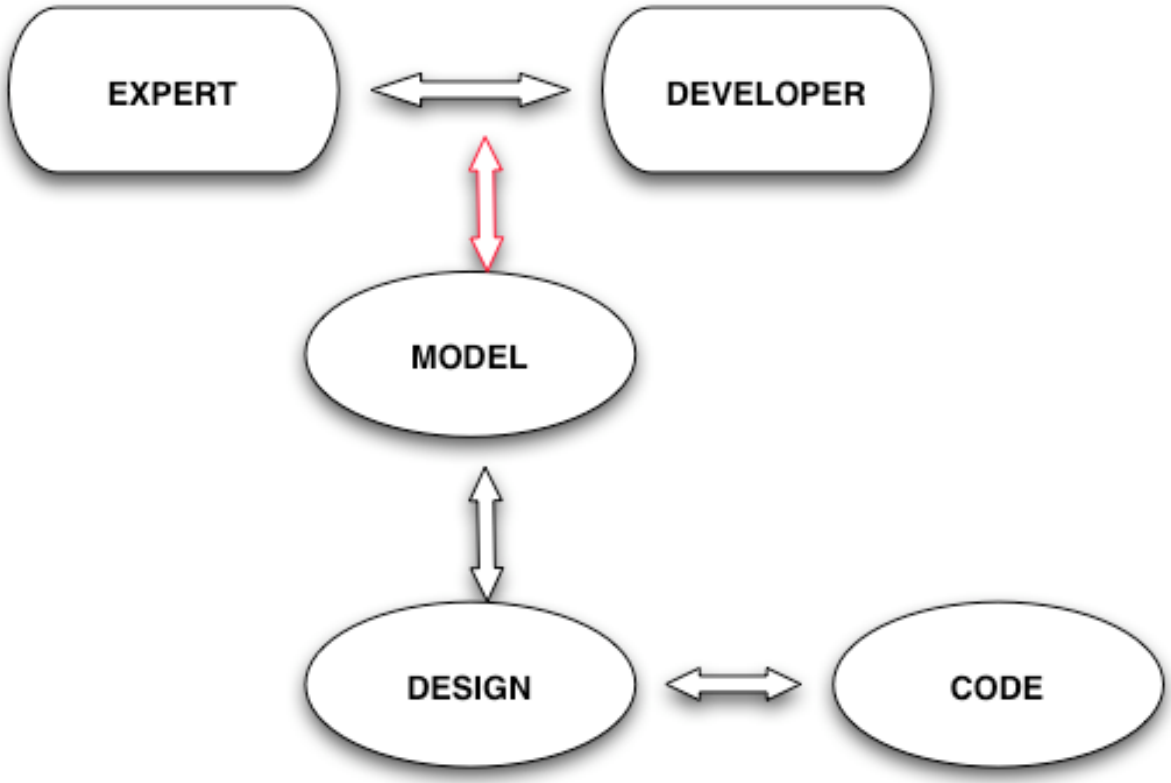


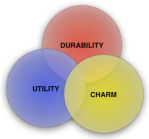

# Street A, D and P

- Someday a book
- View systems admins, they are in the trenches
- Important information is in the scripting language that crops up - still main program with subroutine but OO is creeping in - Python
- Reusing systems not just code - veneer, wrapping is important
- Retiring systems is difficult - better look at a clump!
- Every architect, developer and programmer marks their territory
- What is the process - what is the real process
- Look at the work environment - live there if you can
- Examine the information food chain - is it being used or is it preserving the organization
- Information technology usually requires a dramatic change in the work place
- Know business rules, beware business rules, become part of business rules - "new" arch and design from bottom up
- The web


DURABILITY
UTILITY
CHARM

# 30,000 Feet

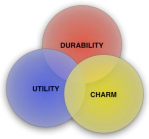

EXPERT
DEVELOPER
MODEL
DESIGN
CODE




# The Three Topics

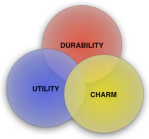
- Architecture
- Design
- Refactoring

THANKS!!!!!



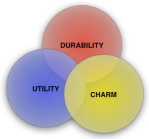

# In Summary

- Keep current
- Collect data
- Reflect - log books
- Give back to field
- Check my web site every now and again
- Appreciate insights (will post with attribution and permission) and new sources
- Fun! - it is always easy, but it is there in any task in this field
- And keep in mind ….

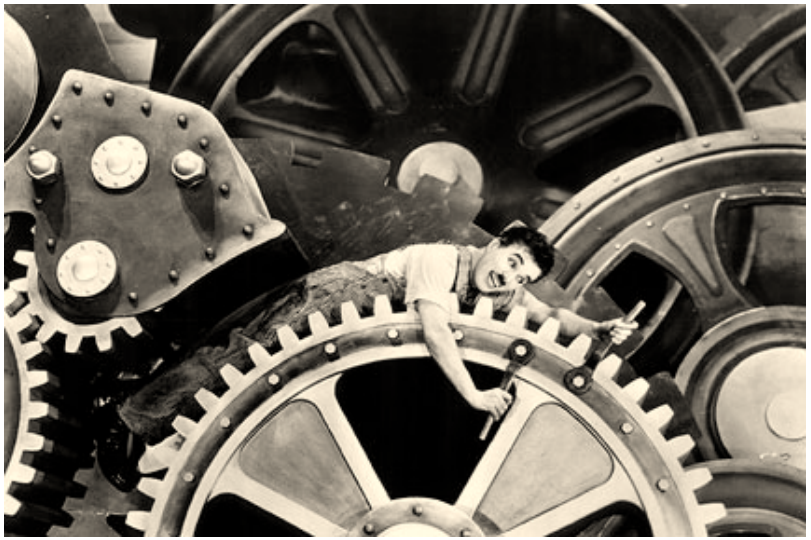


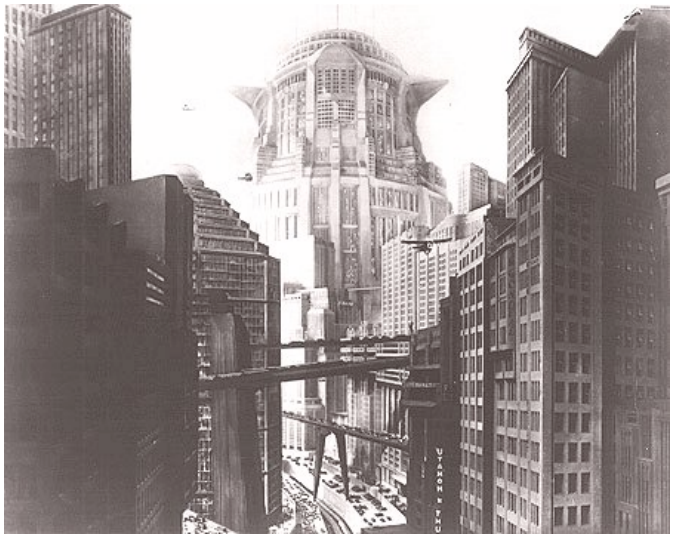

# One More Thing

- It is about people! It should be about joy.

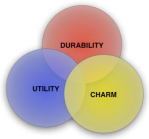



# Dystopia/Utopia/Other



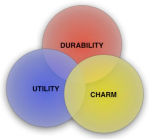


Result &
Workplace



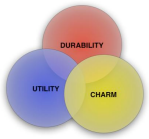

# Review 1

- eXtreme Programming
- Domain Model
- Ubiquitous Language
- Layered Architecture
- Smart UI what it is, advantages, …
- Entities, Value Objects and Services
- Identity
- Modules
- Non objects - rules
- Entity life cycle
- Aggregates (factory)
- Services
- Repositories/Reconstitution
- Refactoring to a deeper model
- Knowledge crunching
- Specification
- Supple Design
- Intention Revealing Interfaces
- Conceptual contours
- Deep Models + Supple Design



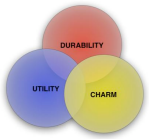

# Review - 2

- Angles of Attack, Divide and Conquer, Unclear - spin off teams, especially in Multiple Bounded Contexts
- Context, Distillation
  - Context map
  - Bounded Context
- Large Scale Structure, 3 themes:
  - context (consistency), Distillation (reduce clutter, focus) large scale structure
- Multiple Models
- Shared Kernel
- Upstream/Downstream
  - Customer/supplier
  - Conformist
  - Anti-corruption layer
  - Separate ways
- Large scale structure
- Responsibility layers
- Meta Knowledge, reflection




# Review - 3

- Refactoring
  - Bad smells
  - Stages (all)
  - Performance
  - Common refactorings
  - Target of code refactoring - think small, different from Big Refactorings
  - When not to refactor
  - Issues with public interfaces
  - Big Refactorings
- Patterns: structural, creational or behavioral
- Special topics of last lectures




# Other References


- Weinberg, G.M. Rethinking systems analysis and design, 1988, Dorset House Publishing, ISBN:0-932633-08-0, may be out of print but there are more than a few Weinberg books that press these points
- http://www.cert.org/archive/pdf/2004eCrimeWatchSummary.pdf
- 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.