Building, Deploying, and Living With IT Systems

LBSC 690: Jordan Boyd-Graber

University of Maryland

November 19, 2012

COLLEGE OF INFORMATION STUDIES

Adapted from Jimmy Lin's Slides

Today's Topics

- The system life cycle
- The open source model
- Cloud computing

Take-Away Messages

- Not “what are the right answers,” but “what are the right questions”
- There is no right answer
  - It all depends on the exact circumstances
  - It’s all about tradeoffs

Outline

1. Designing Systems

2. Open Source and TCO

3. Privacy

4. Cloud Computing

The System Life Cycle

- Analysis and Design: How do we know what to build?
- Implementation: How do we actually build it?
- Maintenance: How do we keep it running?

User-Centered Design

- As opposed to what?
- Understanding user needs
  - Who are the present and future users?
  - How can you understand their needs?
- Understanding the use context
  - How does the particular need relate to broader user activities?
  - How does software fit into the picture?

Some Library Activities

- Acquisition
- Cataloging
- Reference
- Circulation, interlibrary loan, reserves
- Recall, fines, . . .
- Budget, facilities schedules, payroll, . . .

Important Questions

- Where does information originate?
  - Beware of “chicken and egg” problems
- What components already exist?
  - Sometimes it’s easier to start with a clean slate
- Which components should be automated?
  - Some things are easier to do without computers

Important Questions

- Which components should be integrated?
  - Pick your poison: centralization vs. decentralization
  - Implications for privacy, security, etc.
- How will technology impact human processes?
  - Technology is not neutral
- How can we take advantage of the community?
  - Web 2.0, Library 2.0, etc.

Requirements

- Availability
  - Mean Time Between Failures (MTBF)
  - Mean Time To Repair (MTTR)
- Capacity
  - Number of users (typical and maximum)
  - Response time
- Flexibility
  - Upgrade path
  - Interoperability with other applications

Decisions, Decisions. . .

- Off-the-shelf applications vs. custom-developed
- “Best-of-breed” vs. integrated system

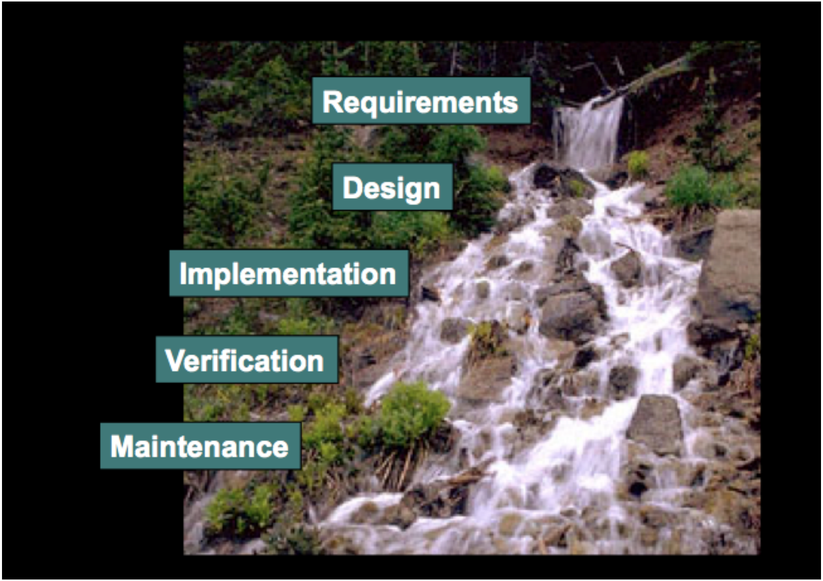
More Decisions: Architectures

- Desktop applications
  - What we normally think of as software
- Batch processing (e.g., recall notices)
  - Save it up and do it all at once
- Client-Server (e.g., Web)
  - Some functions done centrally, others locally
- Peer-to-Peer (e.g., Kazaa)
  - All data and computation are distributed

The Waterfall Model

- Key insight: upfront investment in design
  - An hour of design can save a week of debugging!
- Five stages:
  - Requirements: figure out what the software is supposed to do
  - Design: figure out how the software will accomplish the tasks
  - Implementation: actually build the software
  - Verification: makes sure that it works
  - Maintenance: makes sure that it keeps working

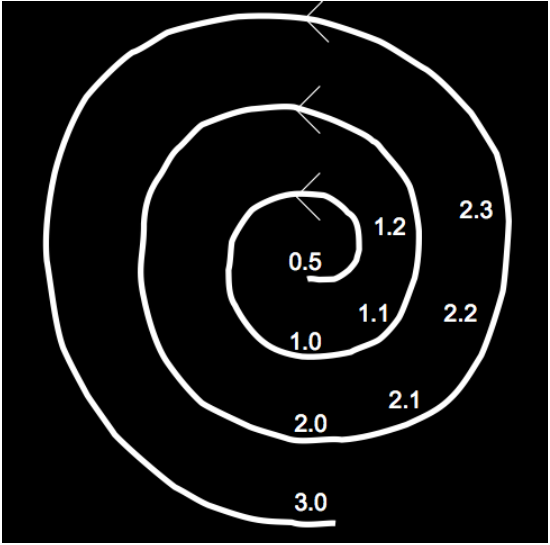
The Waterfall Model



The Spiral Model

- Build what you think you need
  - Perhaps using the waterfall model
- Get a few users to help you debug it
  - First an “alpha” release, then a “beta” release
- Release it as a product (version 1.0)
  - Make small changes as needed (1.1, 1.2, .)
- Save big changes for a major new release
  - Often based on a total redesign (2.0, 3.0,)

The Spiral Model



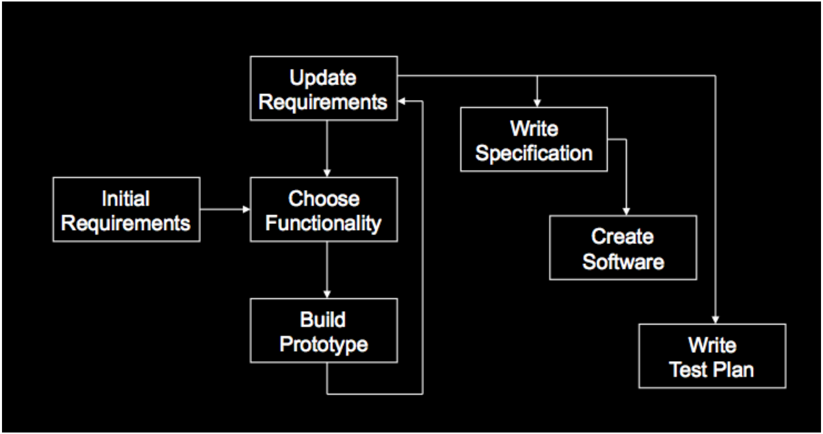
Unpleasant Realities

- The waterfall model doesn't work well
  - Requirements usually incomplete or incorrect
- The spiral model is expensive
  - Redesign leads to recoding and retesting

A Hybrid Model

- Goal: explore requirements
  - Without building the complete product
- Start with part of the functionality
  - That will (hopefully) yield significant insight
- Build a prototype
  - Focus on core functionality
- Use the prototype to refine the requirements
- Repeat the process, expanding functionality

A Hybrid Model



Management Issues

- Operating costs
  - Staff time
  - Physical resources (space, cooling, power)
  - Periodic maintenance
  - Equipment replacement
  - Retrospective conversion
- Moving from "legacy systems"
  - Even converting electronic information is expensive!
- Incremental improvements
  - No piece of software is perfect
- Legal constraints

Management Issues

- Management information
  - Usage logs, audit trails, etc.
  - Often easy to collect, difficult to analyze
- Training
  - Staff
  - Users
- Privacy, security, access control
- Backup and disaster recovery
  - Periodicity, storage location

TCO

- TCO = “Total cost of ownership”
- Hardware and software isn’t the only cost!
- Other (hidden) costs:
  - Planning, installation, integration
  - Disruption and migration
  - Ongoing support and maintenance
  - Training (of staff and end users)

Legal Requirements

- Sarbanes-Oxley (2007) - Corporations must retain and release more information
- PATRIOT (2001) - Government can monitor web and e-mail
- Gramm-Leach-Bliley (1999) - Financial companies
- Digital Millennium Copyright Act (1998) - Illegal to circumvent copyright mechanisms
- Health Insurance Portability and Accountability Act (1996, HIPAA) - Rules for handling health information, establishes rules for transferring health records electronically

Outline

1 Designing Systems

2 Open Source and TCO

3 Privacy

4 Cloud Computing

What is open source?

- Proprietary vs. open source software
- Open source used to be a crackpot idea:
  - Bill Gates on GNU/Linux (3/24/1999): "I don't really think in the commercial market, we'll see it in any significant way."
  - MS 10-Q quarterly filing (1/31/2004): "The popularization of the open source movement continues to pose a significant challenge to the company's business model"
- Open source
  - For tree hugging hippies?
  - Make love, not war?

Basic Definitions

- What is a program?
- What is source code?
- What is object/executable code (binaries)?

Proprietary Software

- Distribution in machine-readable binaries only
- Payment for a license
    - Grants certain usage rights
    - Restrictions on copying, further distribution, modification
- Analogy: buying a car . . .
    - With the hood welded shut
    - That only you can drive
    - That you can't change the rims on

Software is copyrighted

- Copyright is a legal monopoly granted by the government for a *limited time* to promote the arts
- Law gives redress to copyright holders whose work has been infringed
- Software costs nothing to copy - protects the livelihood of those who write software

Open Source Principles

- Free distribution and redistribution
  - “Free as in speech, not as in beer”
- Source code availability
- Provisions for derived works

Open Source vs. Proprietary

- Who gets the idea to develop the software?
- Who actually develops the software?
- How much does it cost?
- Who can make changes?

Distinction: Free vs. Open

- “Free” software is not just open source
- Open source means you can view the code of a program and use it **without charge**
- “Free” software means that if you distribute a program, you must also distribute the source
- What’s the difference?

Distinction: Free vs. Open

- “Free” software is not just open source
- Open source means you can view the code of a program and use it **without charge**
- “Free” software means that if you distribute a program, you must also distribute the source
- What’s the difference?
- Example of Open Source: Apache License
  1. Free to use code, can use in closed-source products
- Example of Free License: Gnu Public License (viral?)
  1. Free to use code, but if you distribute program, must distribute code too

Examples of Open Source Software

Task	Proprietary	Open
OS	Windows	GNU/Linux
IM	AIM	Adium
Browser	Internet Explorer	Firefox
Image Editor	Photoshop	GIMP
Web Server	IIS	Apache
Database	Oracle	MySQL
Office Suite	Microsoft Office	Open Office / LibreOffice

Open Source: Pros

- Peer-reviewed code
- Dynamic community
- Iterative releases, rapid bug fixes
- Released by engineers, not marketing people
- High quality
- No vendor lock-in
- Simplified licensed management

Pros in Detail

- Peer-reviewed code
  - Everyone gets to inspect the code
  - More eyes, fewer bugs
- Dynamic community
  - Community consists of coders, testers, debuggers, users, etc.
  - Any person can have multiple roles
  - Both volunteers and paid by companies
  - Volunteers are highly-motivated to work on something that interests them

Pros in Detail

- Iterative releases, rapid bug fixes
  - Anyone can fix bugs
  - Bugs rapidly fixed when found
  - Distribution of “patches”
- Released by engineers, not marketing people
  - Stable versions ready only when they really are ready
  - Not dictated by marketing deadlines
  - High quality

Pros in Detail

- No vendor lock-in
  - Lock in: dependence on a specific program from a specific vendor
  - Putting content in MS Word ties you to Microsoft forever
  - Open formats: can use a variety of systems
- Simplified licensed management
  - Can install any number of copies
  - No risk of illegal copies or license audits
  - No anti-piracy measures (e.g. CD keys, product activation)
  - No need to pay for perpetual upgrades
  - Doesn't eliminate software management, of course

Cons of Open Source

- Dead-end software
- Fragmentation
- Developed by engineers, often for engineers
- Community development model
- Inability to point fingers

Cons in Detail

- Dead-end software
  - Development depends on community dynamics: What happens when the community loses interest?
  - How is this different from the vendor dropping support for a product? At least the source code is available
- Fragmentation
  - Code might "fork" into multiple versions: incompatibilities develop
  - In practice, rarely happens

Cons in Detail

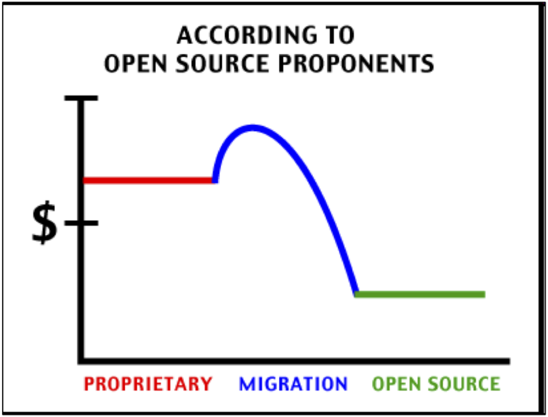
- Developed by engineers, often for engineers
  - My favorite "pet feature"
  - Engineers are not your typical users!
- Community development model
  - Cannot simply dictate the development process
  - Must build consensus and support within the community
- Inability to point fingers
  - Who do you call up and yell at when things go wrong?
  - Buy a support contract from a vendor!

IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPUs, UP FROM THE OLD LIMIT OF 1,024.
DO YOU HAVE SUPPORT FOR SMOOTH FULL-SCREEN FLASH VIDEO YET?
NO, BUT WHO USES *THAT*?

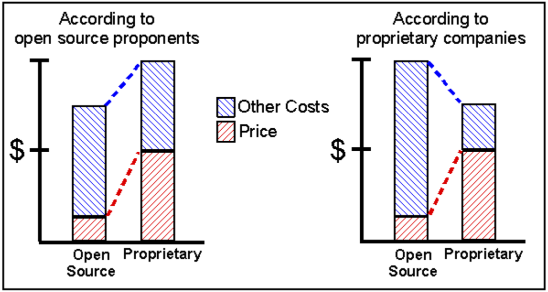
Open Source Business Models

- Support Sellers
- Loss Leader
- Widget Frosting
- Accessorizing

It comes down to cost. . .



The TCO Debate



Is open source right for you?

- Do you have access to the necessary expertise?
- Do you have buy-in from the stakeholders?
- Are you willing to retool your processes?
- Are you willing to retrain staff and users?
- Are you prepared for a period of disruption?
- Have you thought through these issues?

Open source isn't just about software

- Creative commons movement
- "Copyleft"
- Consistent with our Remix culture (Lawrence Lessig)
- Various usage regimes
    - How you can change it
    - If you have to acknowledge
    - If you charge for it
- **Within** copyright regime

Outline

1 Designing Systems

2 Open Source and TCO

3 Privacy

4 Cloud Computing

Why Privacy is a Good Thing

- Build trust in your service
- Protect your users from identity theft / embarassment
- Protect yourself from legal action
- Philosophical / ethical reasons

Why Privacy is not always a Good Thing

- More data leads to better service
  - “collaborative filtering”
  - Targeted advertising/deals
- Transparency and accountability
- People aren’t good at keeping track of their own information

The Good: Netflix Challenge

- “sanitized” data
- Improved predictions of algorithms on what movies you’d like
- Created an invaluable real-world dataset for researchers

The Bad: AOL Search Queries

- Collection of searches given to AOL
- “landscapers in Lilburn, GA”, “shadow lake subdifivsion gwinnett county” revealed user id 4417749 was Thelma Arnold, a 62-year-old widow
- Also revealed other information about users

Best Practices

- Have a privacy statement and policy
- Develop retention policies and audit them
- Allow users to export / expunge their data
- Consider external privacy audits
- Minimize the data you collect

Outline

1 Designing Systems

2 Open Source and TCO

3 Privacy

4 Cloud Computing

What is Cloud Computing?

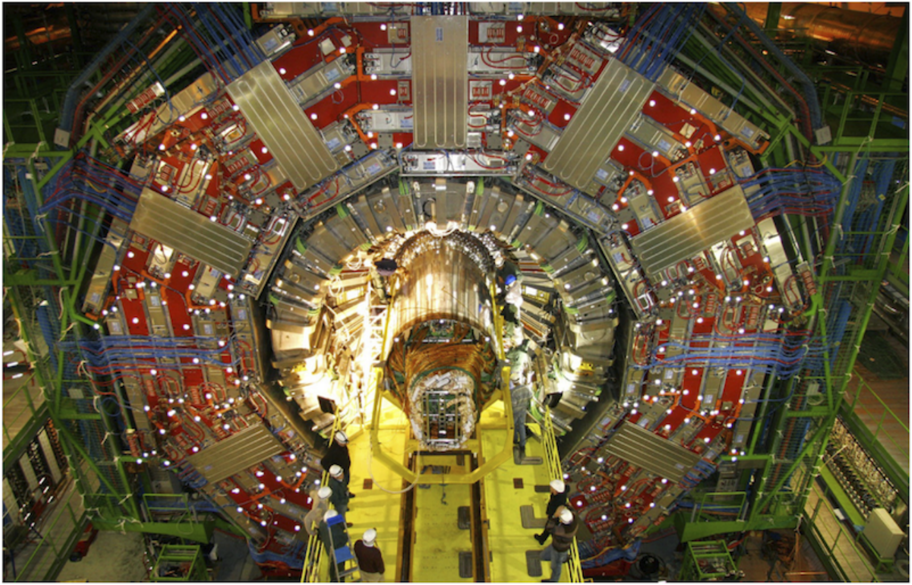
- Web-scale problems
- Large data centers
- Different models of computing
- Highly-interactive Web applications

Web-Scale Problems

- Characteristics:
  - Definitely data-intensive
  - May also be processing intensive
- Examples:
  - Crawling, indexing, searching, mining the Web
  - "Post-genomics" life sciences research
  - Other scientific data (physics, astronomers, etc.)
  - Sensor networks
  - Web 2.0 applications

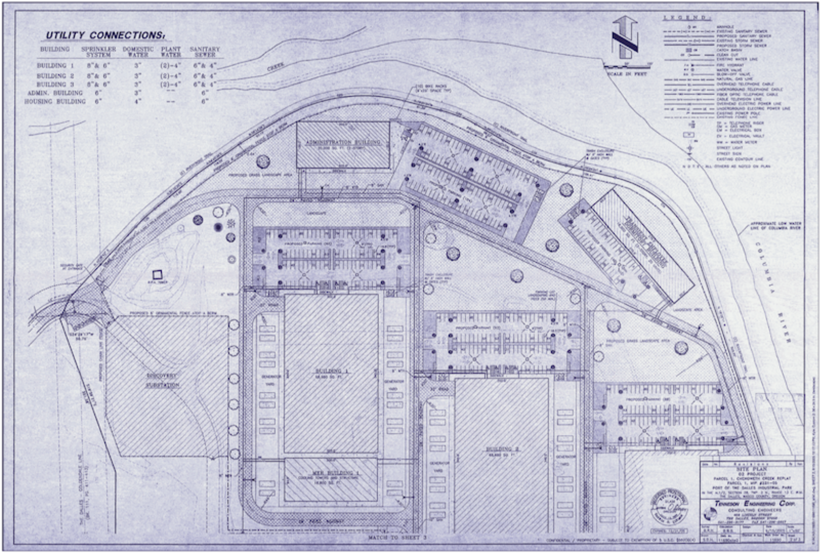
How much data?

- Wayback Machine has 2 PB + 20 TB/month (2006)
- Google processes 20 PB a day (2008)
- “all words ever spoken by human beings”   5 EB
- NOAA has 1 PB climate data (2007)
- CERN’s LHC will generate 15 PB a year (2008)



Large Data Centers

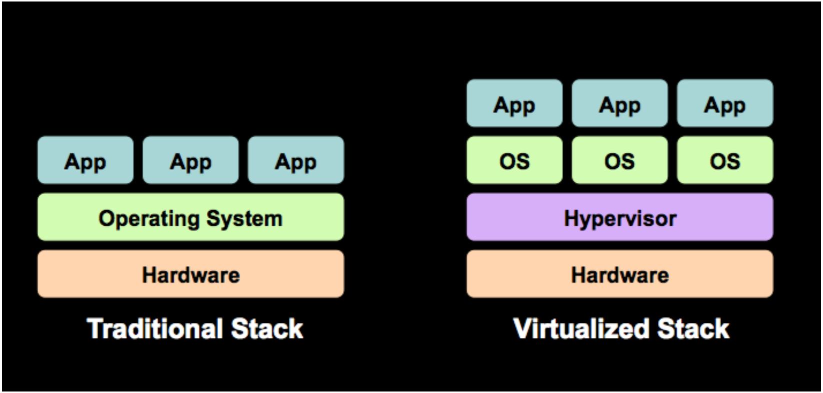
- Web-scale problems? Throw more machines at it!
- Clear trend: centralization of computing resources in large data centers
  - Necessary ingredients: fiber, juice, and space
  - What do Oregon, Iceland, and abandoned mines have in common?
- Important Issues:
  - Redundancy
  - Efficiency
  - Utilization
  - Management



UTILITY CONNECTIONS:

BUILDING	SPRINKLER SYSTEM	DOMESTIC WATER	PLANT WATER	SANITARY SEWER
BUILDING 1	8"& 6"	3"	(2)-4"	6"& 4"
BUILDING 2	8"& 6"	3"	(2)-4"	6"& 4"
BUILDING 3	8"& 6"	3"	(2)-4"	6"& 4"
ADMIN. BUILDING	6"	3"	--	6"
HOUSING BUILDING	6"	4"	--	6"

Key Technology: Virtualization



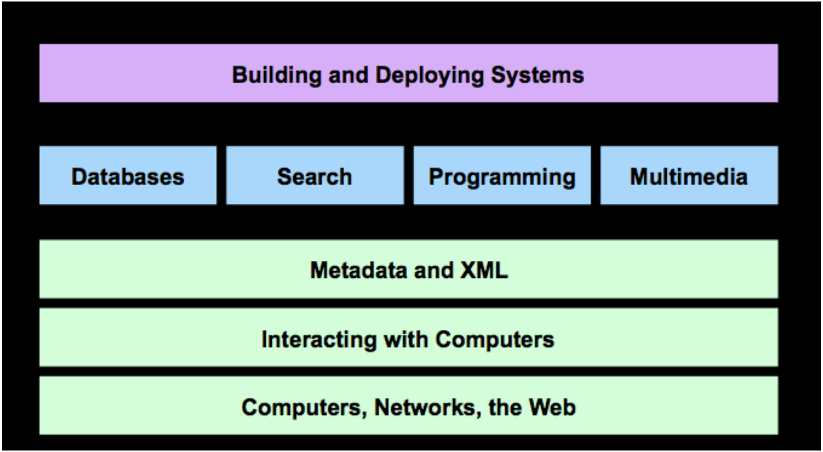
Different Computing Models

- Utility computing
- Why buy machines when you can rent cycles?
  - Examples: Amazon's EC2, GoGrid, AppNexus
- Platform as a Service (PaaS)
  - Give me nice API and take care of the implementation
  - Example: Google App Engine
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail

Web Applications

- What is the nature of software applications?
- From the desktop to the browser
    - Rise of Web-based applications
    - Examples: Google Maps, Facebook
- How do we deliver highly-interactive Web-based applications?
    - Ajax (Asynchronous JavaScript and XML)

The Grand Plan



Building and Deploying Systems

Databases | Search | Programming | Multimedia

Metadata and XML

Interacting with Computers

Computers, Networks, the Web

Discussion Question

Privacy in a Library

- What information do you need and what should you ask for?
- How long should you keep each?

Discussion Question

Privacy in a Library

- What information do you need and what should you ask for?
- How long should you keep each?
- What if you wanted to suggest media to a user? Do you need to keep all of their data forever?

Discussion Question

Privacy in a Library

- What information do you need and what should you ask for?
- How long should you keep each?
- What if you wanted to suggest media to a user? Do you need to keep all of their data forever?
- What if you wanted to allow users to pay fines online?