Moore’s Law Trends in Software Development: Implications for Effort and Schedule

Jo Ann Lane  
San Diego State University  
jalane@mail.sdsu.edu

Justin Zhang  
Canyon Crest High School  
San Diego, CA
Moore’s Law observations*

- **Hardware:**
  - The power of microchips doubles approximately every two years
  - Example: Intel chip evolution over 50 years
    - 3,500 times more performance
    - 90,000 times more energy efficient
    - 60,000 times lower in cost

- **Moore’s Law non-linear acceleration also applies to**
  - Memory units
  - Networks
  - Software
  - Sensors
  - Data

Typical Software "Applications" Today

- Retail Management Systems
- Phone Apps
- Healthcare
- Driverless Cars
- Transportation Management
- National Security
- Home Security

- eCommerce
- Reporting & Analytics
- Loyalty
- Mobile Point of Sale
- Digital Passes
- Point of Sale
Software sizes for various systems*

- Google (Maps, Search, Photos, Android, etc.): 2 billion SLOC
- Car
  - Chevy Volt: 10 million SLOC
  - High-end car: 100 million SLOC
  - Autonomous: MIT guesses ~110 million SLOC plus TBs of data
- Facebook (including backend code): 62 million SLOC
- Microsoft Office 2013: 40+ million SLOC
- Linux kernel (v 4.9.2): 18+ million
- iPhone apps: ~50,000 SLOC

* http://www.visualcapitalist.com/millions-lines-of-code/
Software Environments

- Operate as
  - System of systems
  - Family of systems

- Composed of huge legacy systems

- Embedded to control hardware or perform functions previously performed by hardware

- Often provided as
  - Open source software
  - Configurable software products
How industry is achieving Moore’s Law accelerations in software development

- Significant reuse of open source software
  - OpenStack
  - .NET
  - GitHub repository applications
- APIs to support abstraction and reuse
- Big data and data analysis (e.g., Hadoop)

Allow for fast time-to-market and quick responses to changing worldwide situations....
Development processes to support significant reuse

- Analysis of alternative(s)
  - Compatibility/interoperability
  - % requirements met
  - Requirements not met
  - Cost (licenses/maintenance/etc.)
  - Technical debt
  - Documentation
  - Test materials
  - ...

- Architecture design/enhancement
- Refactoring/integration of alternatives
- Agile methods to evolve system
- Continual testing
• Multi-team
• Multi-vendor
• Multi-contractor
Scoping “new development” for cost models

- Are there really any 100% new software systems?
- How to count reuse
  - Whole software system?
  - All software in a system of systems?
  - Only components that are “touched” in an upgrade cycle?
  - Does size of existing software base really matter if it is continually upgraded by people familiar with the software?
New software vs. software evolution

Some statistics
- 80/20 rule for software development
  - 80% brownfield
  - 20% greenfield
- 80/20 rule for brownfield development
  - 75% adaptive/perfective
  - 4% preventative
  - 21% corrective

Some observations
- New software learning curves can be high for
  - New processes/tools
  - Developers new to organization
- Learning curves can be high for
  - Software that has not been “touched” in recent times
  - Changing development staff over time
Implications for software cost estimation

- Most work today focuses on:
  - Evolution of existing
  - Adaptation of existing
  - Reuse of existing

- Key cost model areas for evolution/adaptation/reuse:
  - Reuse factors
  - Interoperability
    - Reuse components
    - System of systems
  - Refactoring
  - Technical debt

- Other types of software development to consider:
  - Big data analysis
  - Embedded software

- New roles for cost models:
  - Analysis of alternatives
    - Architectures
    - Reusable software
    - Level of trust
  - Data capture, storage, analysis, and access
Implications for software engineering education

- Analysis of reuse alternatives
- Technical debt assessments
- Modification of existing software from unknown author
- Refactoring
- Big data management and analysis
- Security analysis/engineering/re-engineering
Thoughts and questions going forward

- Software time to market, scope, size, and vulnerabilities are causing us to rethink software engineering practices
- What are the impacts for software development cost models
- Our challenge is to anticipate the software and systems engineering cost model needs of tomorrow
- Will yesterday’s software development information give us what we need for the next generation software cost model?