Chapter 31

- **Emerging Trends in Software Engineering**

Slide Set to accompany

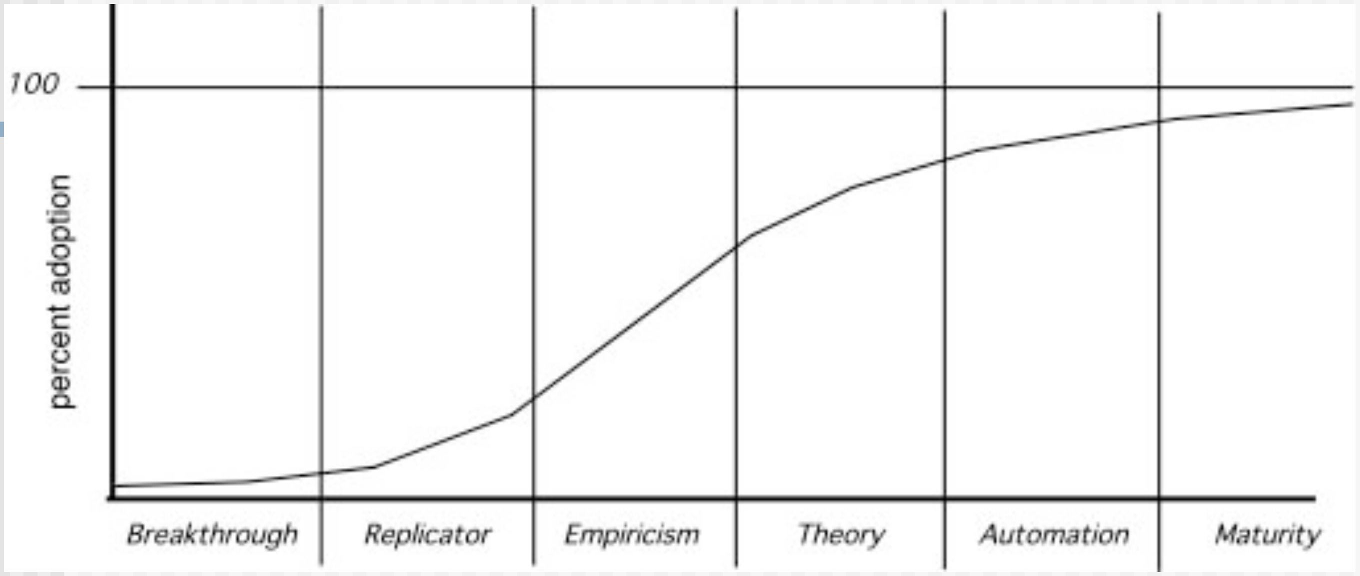
Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

Trends

- Challenges we face when trying to isolate meaningful technology trends:
  - ***What Factors Determine the Success of a Trend?***
  - ***What Lifecycle Does a Trend Follow?***
  - ***How Early Can a Successful Trend be Identified?***
  - ***What Aspects of Evolution are Controllable?***
- Ray Kurzweil [Kur06] argues that technological evolution is similar to biological evolution, but occurs at a rate that is orders of magnitude faster.
  - Evolution (whether biological or technological) occurs as a result of positive feedback—“the more capable methods resulting from one stage of evolutionary progress are used to create the next stage.” [Kur06]

Technology Innovation Lifecycle



Breakthrough: Viable solution to a problem is discovered
Replicator: Solution is reproducible, gains wider usage
Empiricism: Rules governing solution use are devised
Theory: Broader theory of usage, applicability
Automation: Tools which can automatically use/deploy/work with the solution (becomes a technology)
Maturity: Widespread use of the technology

Observing SE Trends

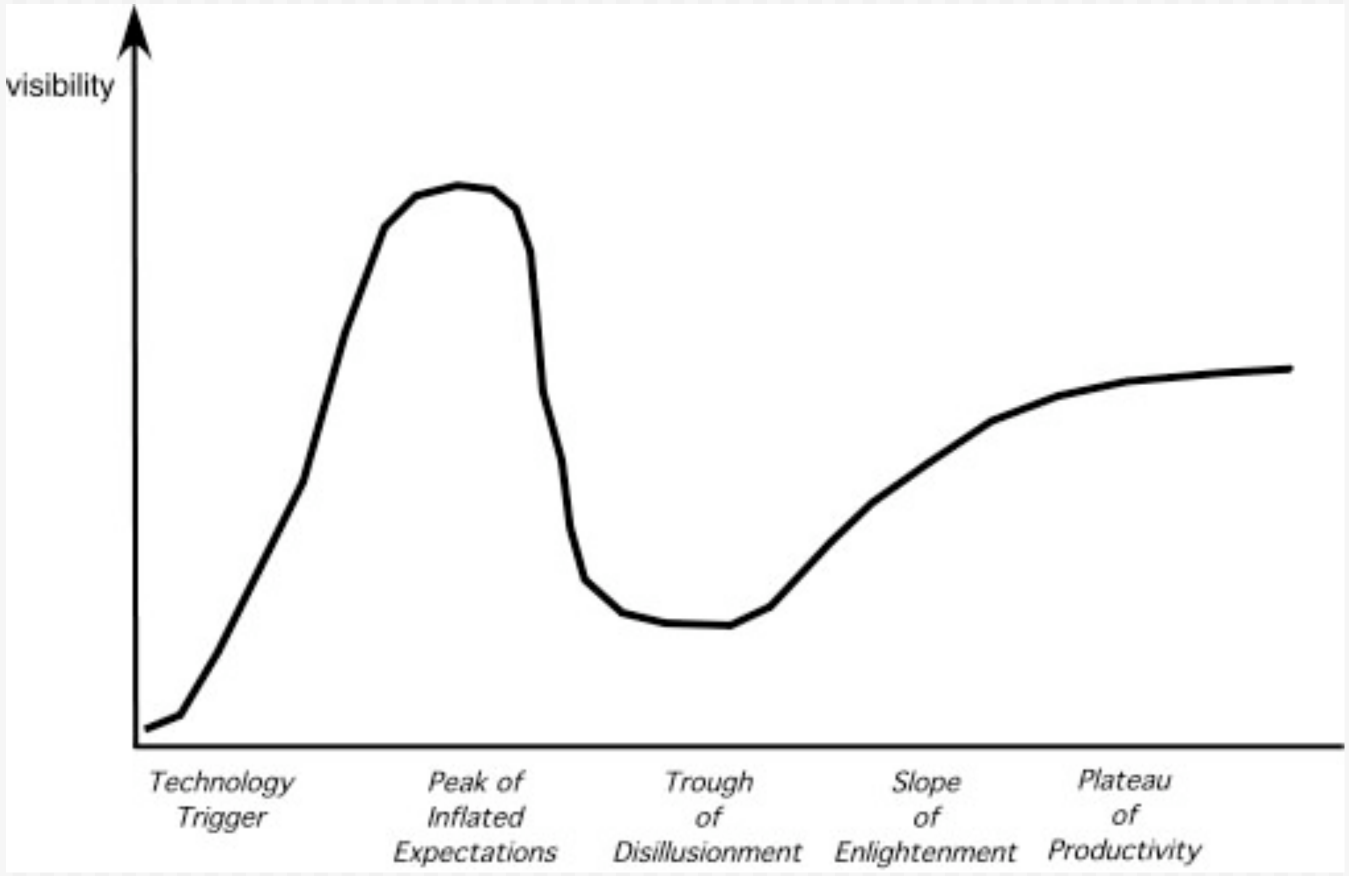
- Barry Boehm [Boe08] suggests that "software engineers [will] face the often formidable challenges of dealing with rapid change, uncertainty and emergence, dependability, diversity, and interdependence, but they also have opportunities to make significant contributions that will make a difference for the better."
- But what of more modest, short-term innovations, tools, and methods?

The Hype Cycle

- *technology trigger*—a research breakthrough or launch of an innovative new product that leads to media coverage and public enthusiasm
- *peak of inflated expectations*—over-enthusiasm and overly optimistic projections of impact based on limited, but well-publicized successes
- *disillusionment*— overly optimistic projections of impact are not met and critics begin the drumbeat; the technology becomes unfashionable among the cognoscenti
- *slope of enlightenment*—growing usage by a wide variety of companies leads to a better understanding of the technology's true potential; off the shelf methods and tools emerge to support the technology
- *plateau of productivity*—real world benefits are now obvious and usage penetrates a significant percentage of the potential market

Gartner Group [Gar08]

The Hype Cycle



Soft Trends

- *Connectivity and collaboration* (enabled by high bandwidth communication) has already led to a software teams that do not occupy the same physical space (telecommuting and part-time employment in a local context).
- *Globalization* leads to a diverse workforce (in terms of language, culture, problem resolution, management philosophy, communication priorities, and person-to-person interaction).
- *An aging population* implies that many experienced software engineers and managers will be leaving the field over the coming decade. The software engineering community must respond with viable mechanisms that capture the knowledge of these aging managers and technologists
- *Consumer spending in emerging economies* will double to well over $9 trillion. [Pet06] There is little doubt that a non-trivial percentage of this spending will be applied to products and services that have a digital component—that are software-based or software-driven.

Managing Complexity

- *In the relatively near future, systems requiring over 1 billion LOC will begin to emerge*
  - Consider the interfaces for a billion LOC system
    - both to the outside world
    - to other interoperable systems
    - to the Internet (or its successor), and
    - to the millions of internal components that must all work together to make this computing monster operate successfully.
  - Is there a reliable way to ensure that all of these connections will allow information to flow properly?
  - Consider the project itself.
  - Consider the number of people (and their locations) who will be doing the work
  - Consider the engineering challenge.
  - Consider the challenge of quality assurance.

Open-World Software

- Concepts such as *ambient intelligence, context-aware applications,* and *pervasive/ubiquitous computing*—all focus on integrating software-based systems into an environment far broader than anything to date
- "open-world software"—software that is designed to adapt to a continually changing environment 'by self-organizing its structure and self-adapting its behavior." [Bar06]

Emergent Requirements

- As systems become more complex, requirements will emerge as everyone involved in the engineering and construction of a complex system learns more about it, the environment in which it is to reside, and the users who will interact with it.
- This reality implies a number of software engineering trends.
  - process models must be designed to embrace change and adopt the basic tenets of the agile philosophy (Chapter 3).
  - methods that yield engineering models (e.g., requirements and design models) must be used judiciously because those models will change repeatedly as more knowledge about the system is acquired
  - tools that support both process and methods must make adaptation and change easy.

Software Building Blocks

- the software engineering community attempts to capture past knowledge and reuse proven solutions, but a significant percentage of the software that is built today continues to be built "from scratch."
  - Part of the reason for this is a continuing desire (by stakeholders and software engineering practitioners) for "unique solutions."
- "merchant software"—software building blocks designed specifically for a unique application domain (e.g., VoIP devices).

Open Source

- *"Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in."* [OSO08]
- The term *open source* when applied to computer software, implies that software engineering work products (models, source code, test suites) are open to the public and can be reviewed and extended (with controls) by anyone with interest and permission.

Process Trends

- *As SPI frameworks evolve, they will emphasize "strategies that focus on goal orientation and product innovation."* [Con02]
- *Because software engineers have a good sense of where the process is weak, process changes should generally be driven by their needs and should start form the bottom up.*
- *Automated software process technology (SPT) will move away from global process management (broad-based support of the entire software process) to focus on those aspects of the software process that can best benefit from automation.*
- *Greater emphasis will be placed on the return-on-investment of SPI activities.*
- *As time passes, the software community may come to understand that expertise in sociology and anthropology may have as much of more to do with successful SPI as other, more technical disciplines.*
- *New modes of learning may facilitate the transition to a more effective software process.*

The Grand Challenge

- There is one trend that is undeniable—software-based systems will undoubtedly become bigger and more complex as time passes.
- It is the engineering of these large, complex systems, regardless of delivery platform or application domain, the poses the "grand challenge" [Bro06] for software engineers.
- Key approaches:
  - more effective distributed and collaborative software engineering philosophy
  - better requirements engineering approaches
  - a more robust approach to model-driven development, and
  - better software tools

Collaborative Development

- Today, software engineers collaborate across time zones and international boundaries, and every one of them must share information.
- The challenge over the next decade is to develop methods and tools that facilitate that collaboration.
- Critical success factors:
  - Shared goals
  - Shared culture
  - Shared process
  - Shared responsibility

Requirements Engineering

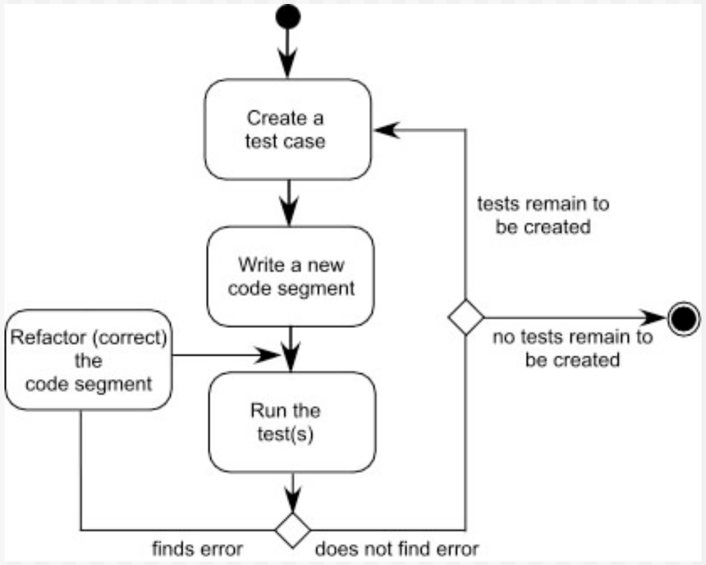
- To improve the manner in which requirements are defined, the software engineering community will likely implement three distinct sub-processes as RE is conducted [Gli07]:
  - improved knowledge acquisition and knowledge sharing that allows more complete understanding of application domain constraints and stakeholder needs
  - greater emphasis on iteration as requirements are defined
  - more effective communication and coordination tools that enable all stakeholders to collaborate effectively.

Model-Driven Development

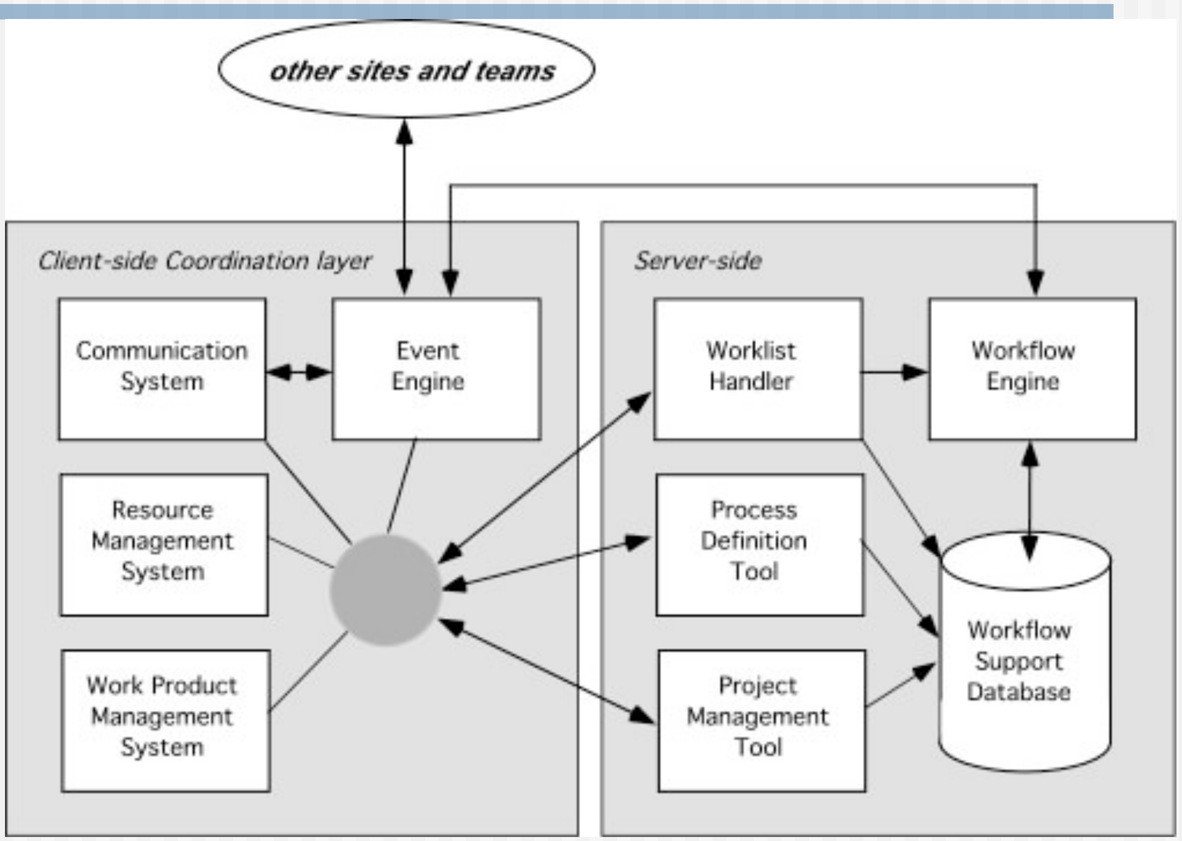
- couples domain-specific modeling languages with transformation engines and generators in a way that facilitates the representation of abstraction at high levels and then transforms it into lower levels [Sch06]
- *Domain-specific modeling languages* (DSMLs)
  - represent "application structure, behavior and requirements within particular application domains"
  - described with metamodels that "define the relationships among concepts in the domain and precisely specify the key semantics and constraints associated with these domain concepts." [Sch06]

Test-Driven Development

- In *test-driven development* (TDD), requirements for a software component serve as the basis for the creation of a series of test cases that exercise the interface and attempt to find errors in the data structures and functionality delivered by the component.
- TDD is not really a new technology but rather a trend that emphasizes the design of test cases *before* the creation of source code.continue to emphasize the importance of software architecture


Create a test case
Write a new code segment
Refactor (correct) the code segment
Run the test(s)
tests remain to be created
no tests remain to be created
finds error
does not find error

Tools Trends—SEE*



*Software Engineering Environment

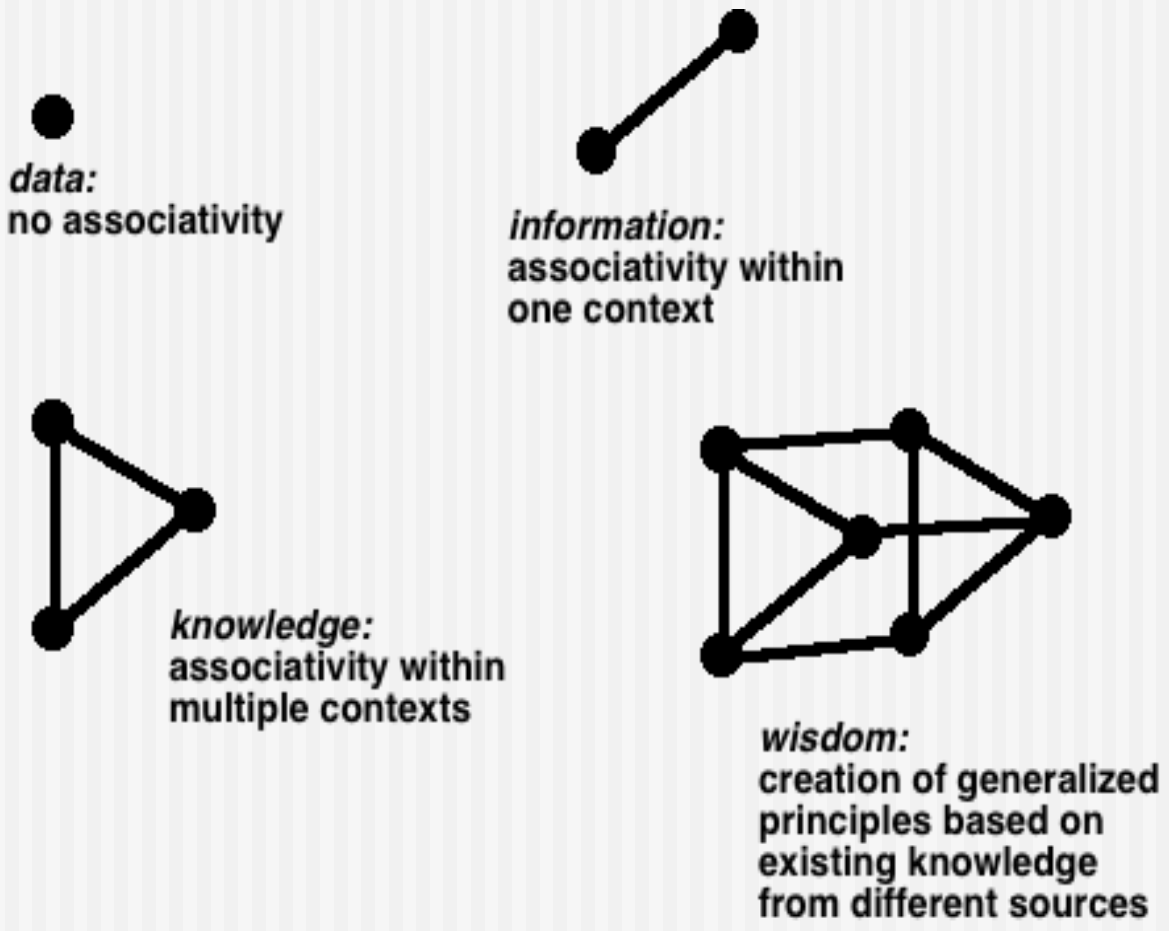
Importance of Software-Revisited

- In Chapter 1, software was characterized as a differentiator.
  - The function delivered by software differentiates products, systems, and services and provides competitive advantage in the marketplace.
- But software is more that a differentiator.
- The programs, documents, and data that are software help to generate the most important commodity that any individual, business, or government can acquire—information.

People - Building Systems

- Communication is changing
  - e.g., video conferencing
- Work patterns are changing
  - e.g., intelligent agents
- Knowledge acquisition is changing
  - e.g., data mining, the Web

An Information Spectrum

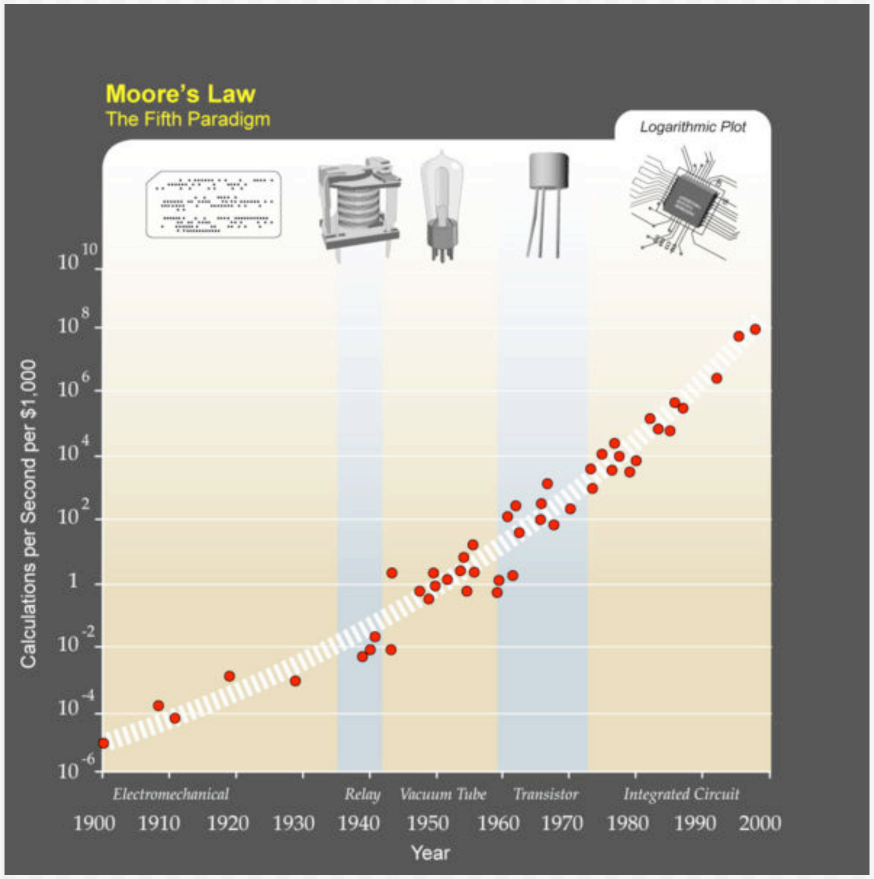

data:
no associativity
information:
associativity within one context
knowledge:
associativity within multiple contexts
wisdom:
creation of generalized principles based on existing knowledge from different sources

The Long View: Singularity

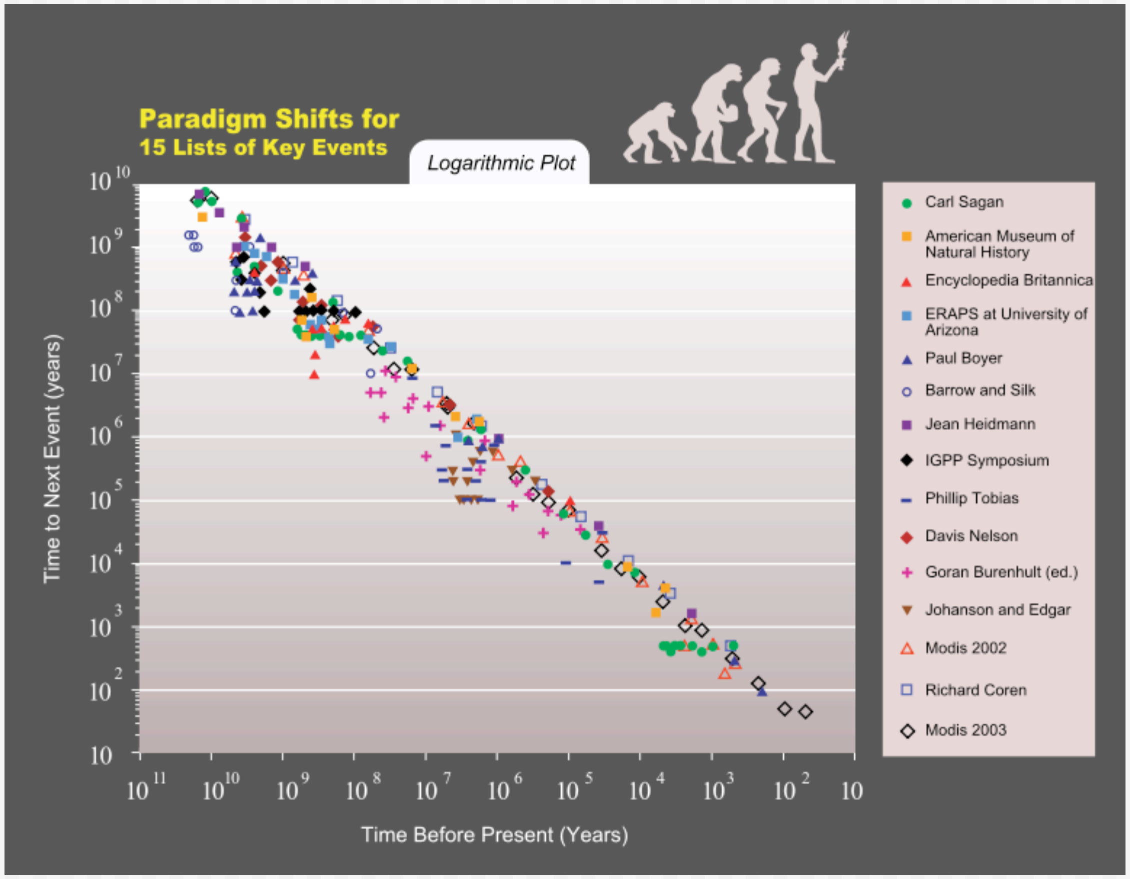
- Singularity:
  - The idea that machines will develop self-awareness, hyper-intelligence, and become super-human
  - See wikipedia –

    http://en.wikipedia.org/wiki/Technological_singularity
- Two conflicting views:
  - Ray Kurzweil—digital utopia and immortality
  - Bill Joy—digital distopia and extinction
- IEEE Report on the concept:
  - http://spectrum.ieee.org/biomedical/ethics/waiting-for-the-rapture

Explosion of Tech Evolution (Kurzweil's View)



Paradigm Shift Timeline (Kurzweil)



Conclusions

- *Engineering approach is essential*
  - Critical to major breakthroughs
  - Processes enable replication, improvement, quality
  - Best products follow best practices
- Personal efforts pay off
  - Know your own metrics well
  - Improve your own software quality
  - Reliable estimations + bug-free code + good processes = successful, highly paid software engineer