Emerging Trends in Software Engineering

presented by
Roger S. Pressman, Ph.D.
R.S. Pressman & Associates, Inc.
Boca Raton, Florida USA
January, 2009

Predictions

- "One of the things that I think we have learned is that we should all be very careful about making predictions about the future." *Bill Clinton, 42nd President of the USA*
- *For example:*
  - "I think there is a world market for maybe five computers." Thomas Watson, chairman of IBM, 1943
  - "There is no reason anyone would want a computer in their home." Ken Olson, President, Chairman of Digital Equipment Corp., 1977
  - "640K ought to be enough for anybody." Bill Gates, chairman of Microsoft, 1981

The Big Picture

Software intensive systems (SIS) have become the foundation of virtually every modern technology.

- Software content in virtually every product and service will continue to grow—in some cases dramatically
- Software must be demonstrably safe, secure, and reliable
- Requirements will emerge as systems evolve
- Interoperability and "networkability" will become dominant as "mash-ups" become the norm
- *A "smart world" demands better, more reliable software*

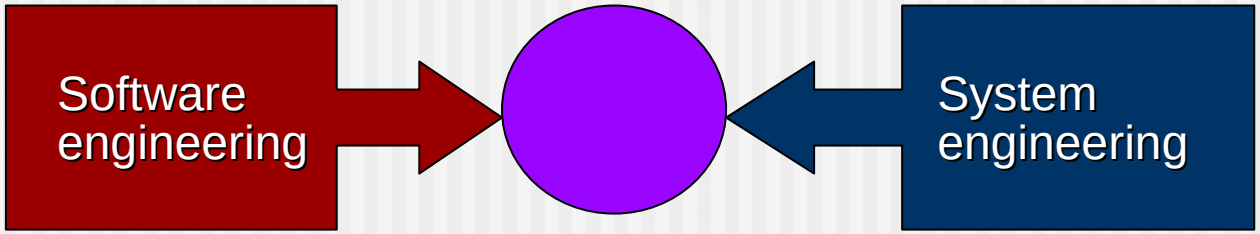
A Harsh Reality

- The challenges facing software engineers will get no easier as we move into the second decade of the 21st century
- New process models, methods, languages, and tools will emerge,

But …

- *There is no silver bullet!*

Software Intensive Systems (SIS)



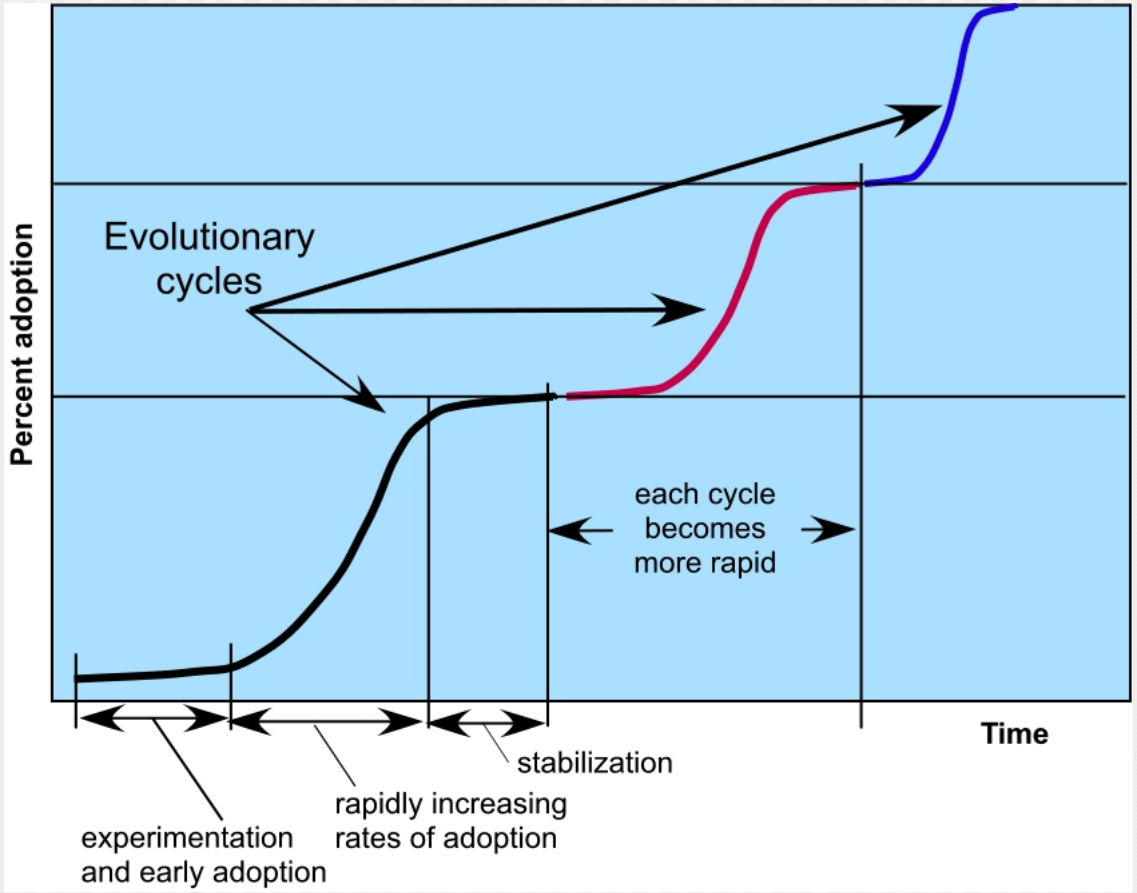
- Increasing integration of software engineering and system engineering activities
- Increasing emphasis on users and end-value
- Increasing SIS criticality and dependability
- Increasing need to manage rapid change
- Increasing project/product globalization and need for interoperability
- Increasing size and complexity
- Increasing software “autonomy” (apps that independently evaluate their environment and choose a course of action that is appropriate)

Barry Boehm

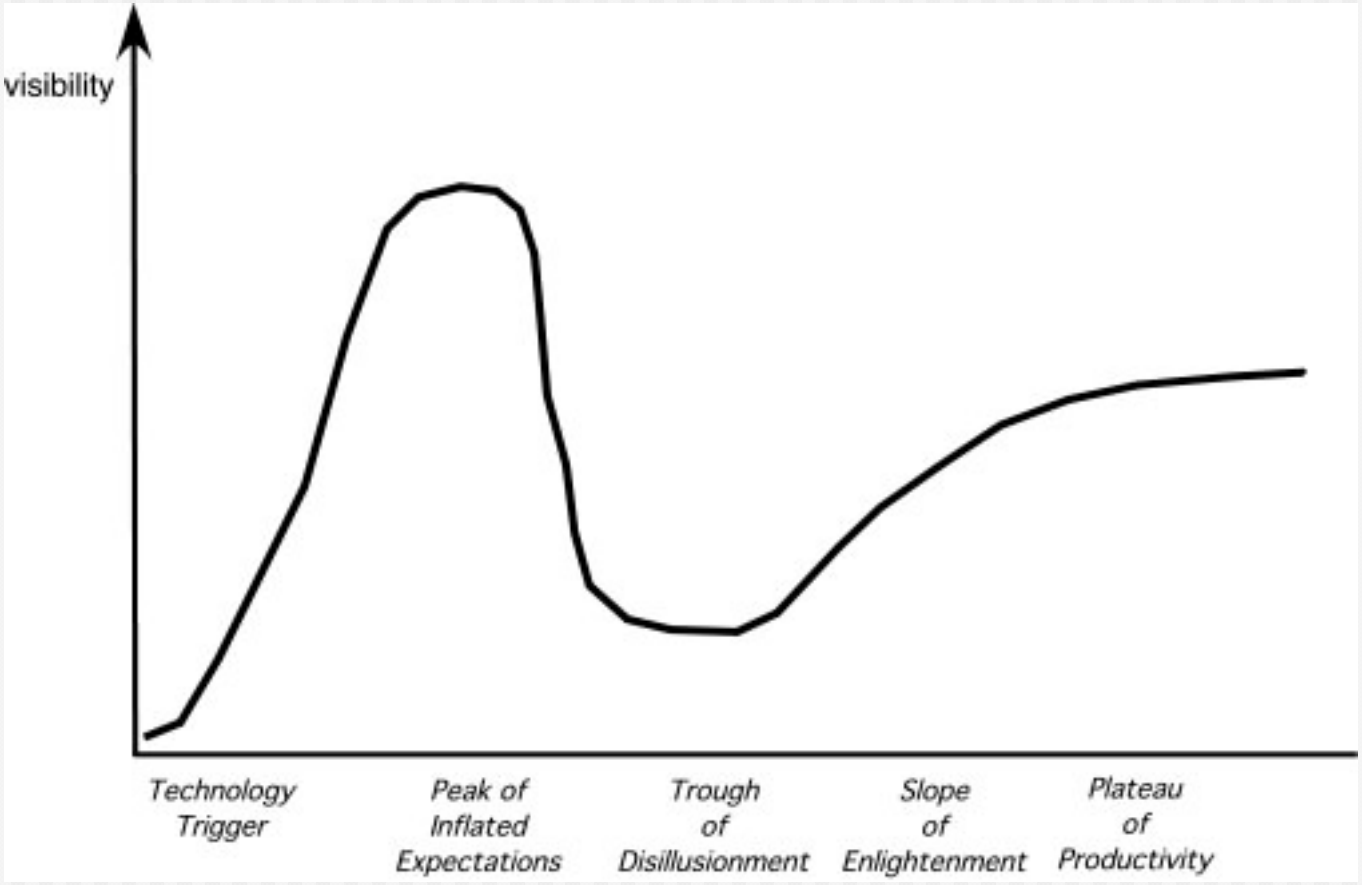
Technology “Evolution”

- Ray Kurzweil argues that technological evolution is similar to biological evolution, but occurs at a rate that is orders of magnitude faster.
  - “ … the more capable methods resulting from one stage of evolutionary progress are used to create the next stage.” [Kurweil, R., *The Singularity is Near,* 2005]
- What are “the more capable [process and] methods [and tools]” for software engineering?

Idealized Technology Innovation Lifecycle



The Hype Cycle



Gartner Group, 2008

Software Engineering Trends

- Soft Trends
  - The broad characteristics of the new systems we build (and test), e.g.,
    - Emergent requirements
    - Autonomy of action
  - The anthropological and sociological characteristics of the new generation of people who do software engineering work
- Hard Trends
  - The technical aspects of next generation software intensive systems
  - The technical directions that software engineering process, methods, and tools will take

Soft Trends - I

- *Connectivity and collaboration* (enabled by high bandwidth communication)
  - 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
  - We need viable mechanisms that capture the knowledge of these aging managers and technologists
- *Consumer spending in emerging economies* will double to well over $9 trillion.
  - Some of this spending will be applied to products and services that have a digital component that is software-based

Software Trends - II

- People and Teams
  - As systems grow in size, teams grow in number, geographical distribution, and culture
  - As systems grow in complexity, team interfaces become pivotal to success
  - As systems become pervasive, teams must manage emergent requirements
  - As systems become more open, *what is a team?*

Managing Size & Complexity - I

- *In the relatively near future, systems requiring over 1 billion LOC will begin to emerge*
  - Consider the interfaces for a billion LOC system
    - 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.
  - How do we manage the project, the teams, the communication among software engineers?

Managing Size & Complexity - II

From a testing perspective:

- Is there a viable way to validate requirements?
- Is there a reasonable way to conduct technical reviews and communicate their results across hundreds of teams?
- Is there a reliable way to ensure that all of the interfaces will allow information to flow properly?
- Is there a realistic strategy for conducting tests, tracking errors, and performing debugging?
- Is there an efficient way to perform regression tests?
- How do you conduct performance tests, stress tests, and security tests when the system is huge?

Pervasive Computing (PvC)

- Concepts such as *ambient intelligence, context-aware applications,* and *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." *

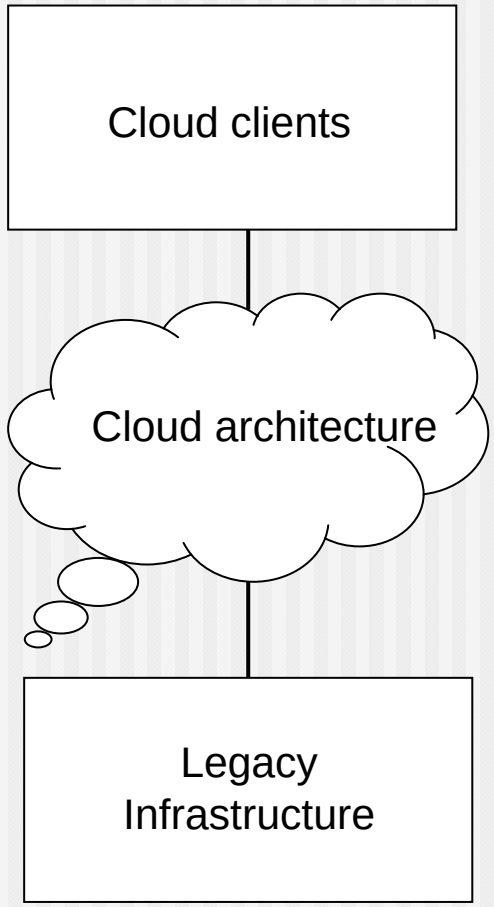
* Baresi, L., E. DiNitto, and C. Ghezzi, "Toward Open-World Software: Issues and Challenges," *IEEE Computer,* October 2006.

Pervasive Computing (PvC)

- First stage (PvC-1) *[today]*
  - Device mobility and ad hoc networking
  - Simple context awareness
  - *Soon:* smart objects implemented in devices that have the potential to communicate with one another
- Second stage (PvC-2) *[over the next decade]*
  - Mobile user profiles that can be recognized by other objects
  - Smart objects will respond to other objects based on situational characteristics
- *Testing issues*
  - Considerable environmental variation
  - Complex communication issues
  - Adaptive processing requirements

Cloud Computing

- "Cloud Computing is a paradigm in which information is permanently stored in servers on the Internet and cached temporarily on clients that include desktops, entertainment centers, table computers, notebooks, wall computers, handhelds, sensors, monitors, etc." *IEEE Internet Computing*
- Provides software as a service (SaaS)
- Device and location independence enables users to access systems regardless of their location or device
- Multi-tenancy enables sharing of resources (and costs) among a large pool of users
- Demands reliability, scalability, security, sustainability (Green IT)



Emergent Requirements

- As systems become more complex, requirements will emerge as everyone learns more about:
  - The system's interoperable elements
  - The environment in which it is to reside, and
  - The objects that interact with it.
- This reality implies a number of software engineering trends.
  - Software process agility—embracing change
  - Judicious use of modeling methods—models will change repeatedly as more knowledge about the system is acquired
  - Tools that make adaptation and change easier

Software Building Blocks

- "merchant software"—software building blocks designed specifically for a unique application domain (e.g., VoIP devices).
- The software engineering issue:
  - **"Component trust"**
- *Testing Issues:*
  - Evaluating/Certifying a component validation suite
  - Establishing trust using **component testing**
  - Integrating using **deployment testing**
  - Validating using **system testing**

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."* *
- 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.

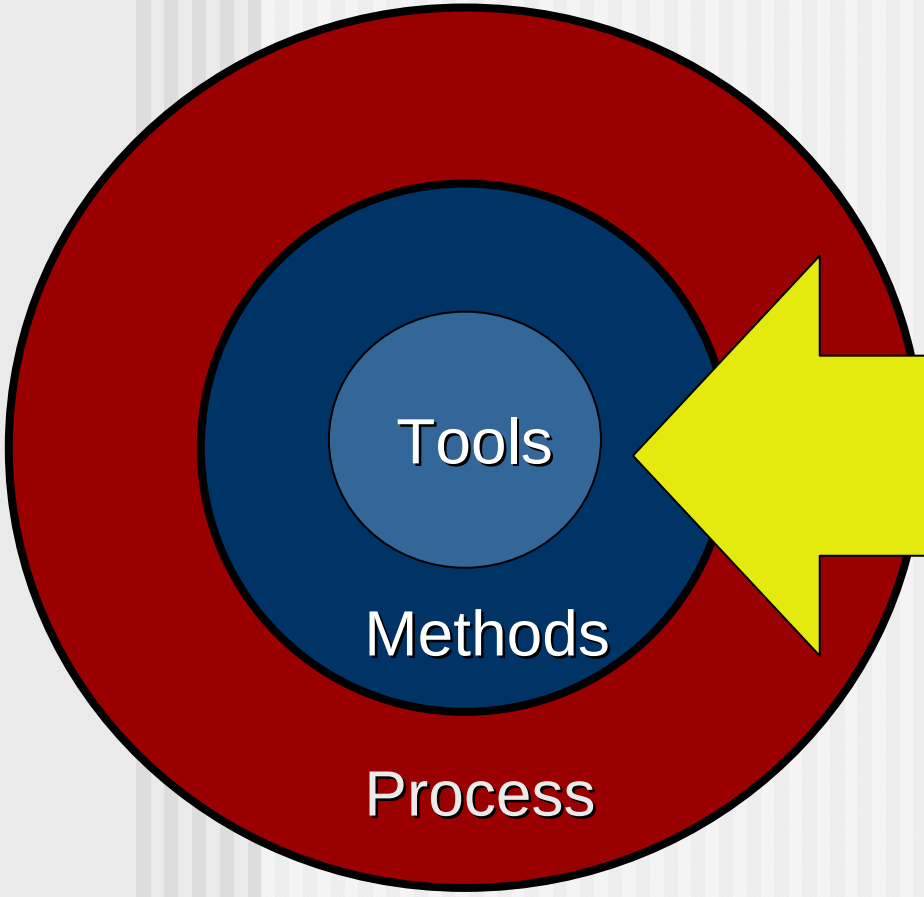
* OpenSource.org, 2008, available at www.opensource.org/.

Testing Open Source

- Functionality is added regularly as the software evolves
  - *It must be tested asap*
- Refactoring occurs continually
  - *The behavior of the software after refactoring must be tested*
- Application code and test code (suites) must evolve simultaneously ("co-evolve")
- To manage testing in an open source environment, we must examine the application in three different "dimensions" *
  - The change history view
  - The growth history view
  - The coverage/evolution view

* Zaidman, A., et al, "On how developers test open source software systems" available at citeseerx.ist.psu.edu, 2007.

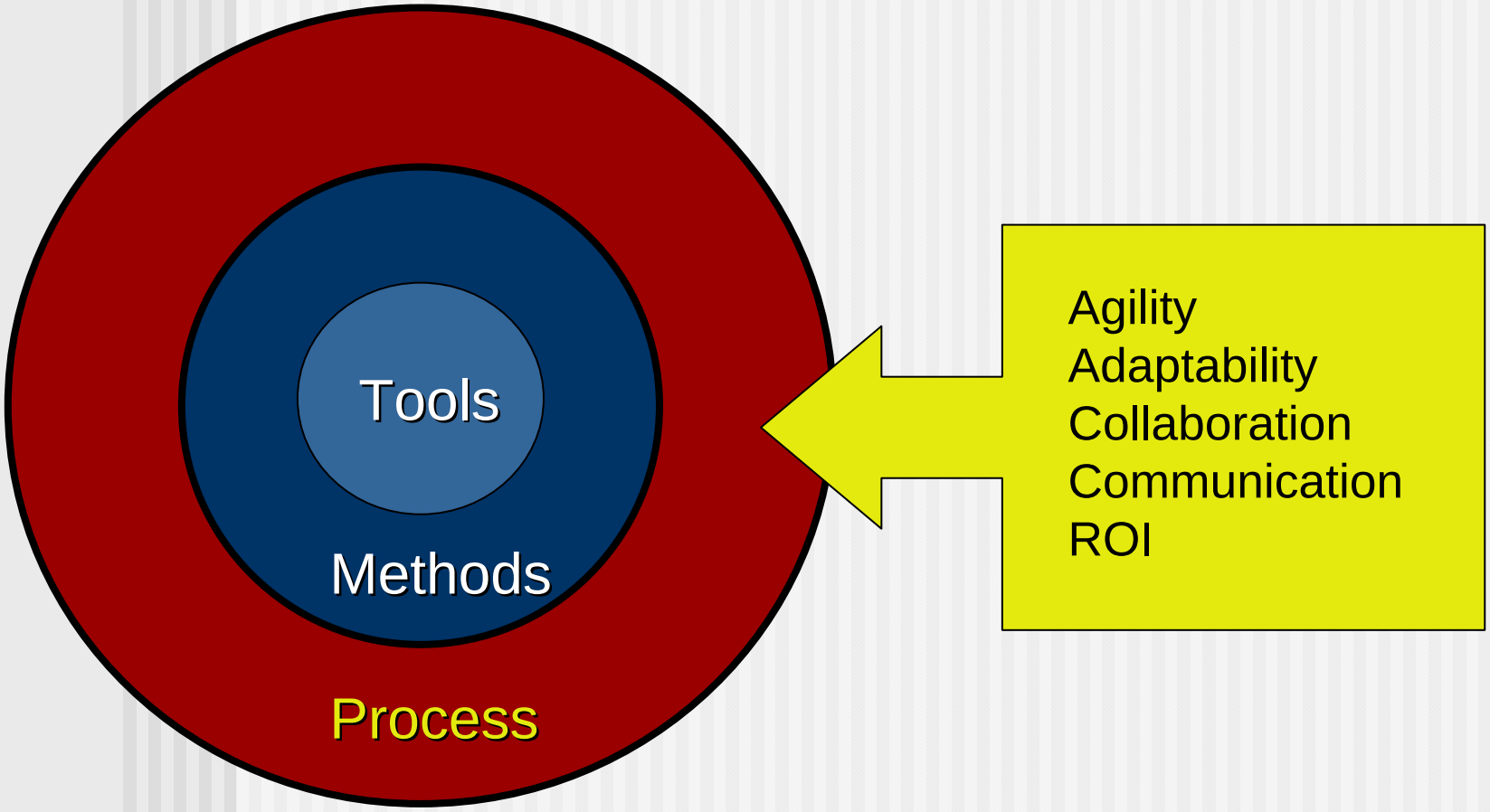
Software Engineering Trends



Ivar Jacobson identifies four major trends:

- Reusable components—stop reinventing the wheel
- More up-front testing—test-driven development
- Platform transparency—reduce today's programming complexity
- UML "all the way down"—model-driven development

SE Trends—Process



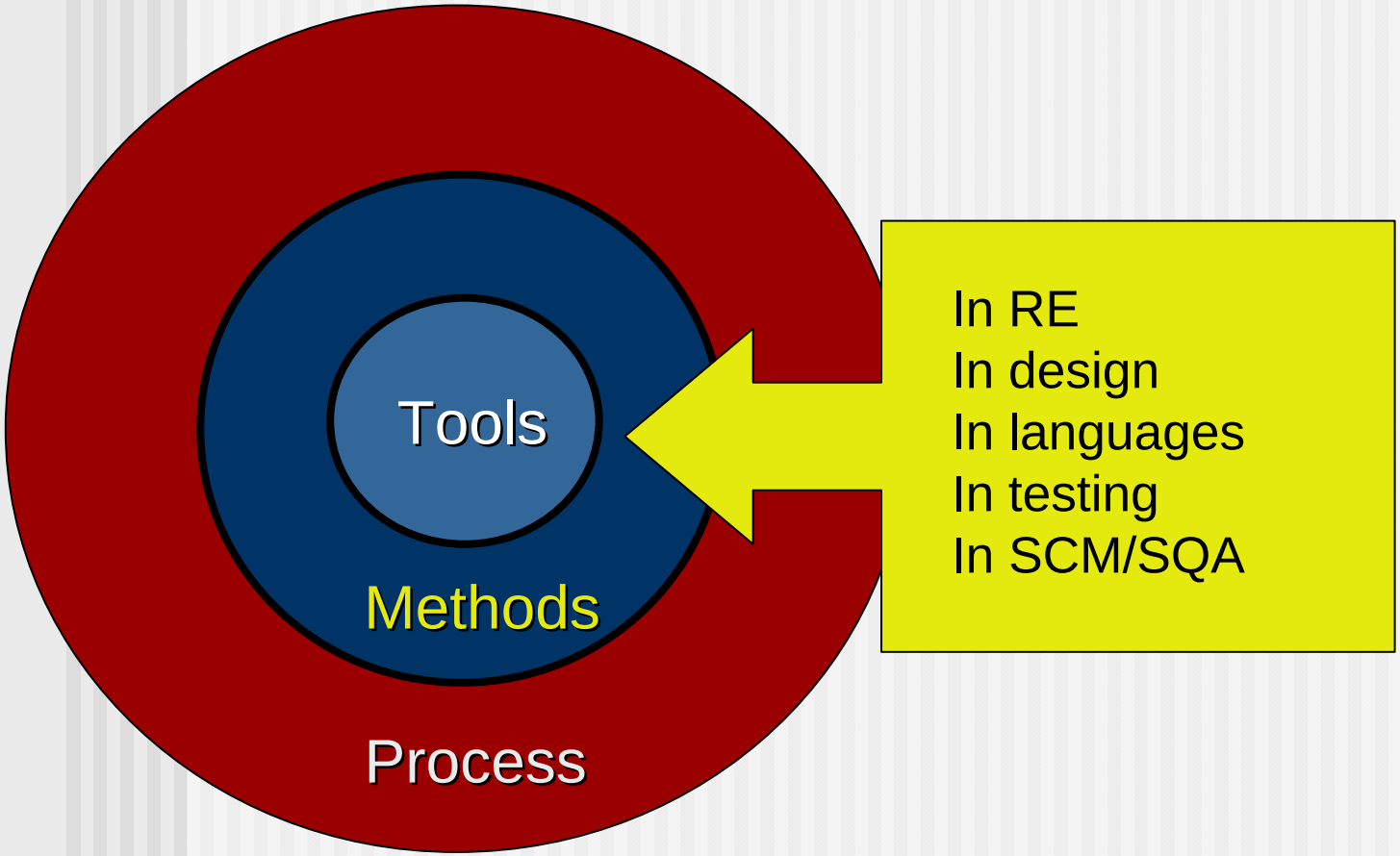
Process Trends

- SPI frameworks will emphasize “strategies that focus on goal orientation and product innovation.” [Con02]
- Process changes will be driven by the needs of practitioners and should start from the bottom up.
- Greater emphasis will be placed on the return-on-investment of SPI activities.
- Expertise in sociology and anthropology may have as much or 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.
- 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.

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 project goals
  - Shared project culture
  - Shared process
  - Shared responsibility

SE Trends—Methods



Methods: *The Grand Challenge*

- The engineering of large, complex systems, regardless of delivery platform or application domain, the poses the "grand challenge" * for software engineers.
- Key approaches:
  - more effective distributed and collaborative software engineering process models
  - better requirements engineering approaches
  - a more robust approach to model-driven development, and
  - better software tools

* Broy. M., "The Grand Challenge," *IEEE Computer,* Oct., 2006.

Requirements Engineering

- The software engineering community will likely implement three distinct sub-processes as RE is conducted:*
  - 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.

* Glinz, M., and R. Wieringa, "Stakeholders in Requirements Engineering," *IEEE Software,* March–April 2007.

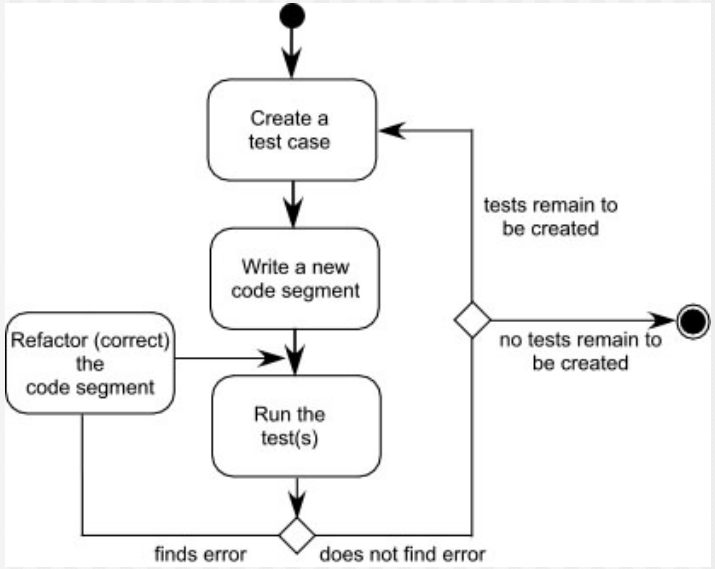
Model-Driven Development

- MDD 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
- *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.” *

* Schmidt, D.,”Model-Driven Engineering,” *IEEE Computer,* February 2006.

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. Continues to emphasize the importance of software architecture.

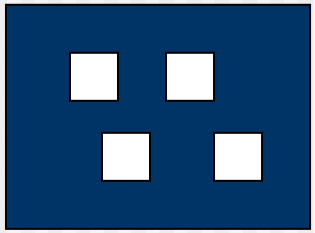


Service-Oriented Development

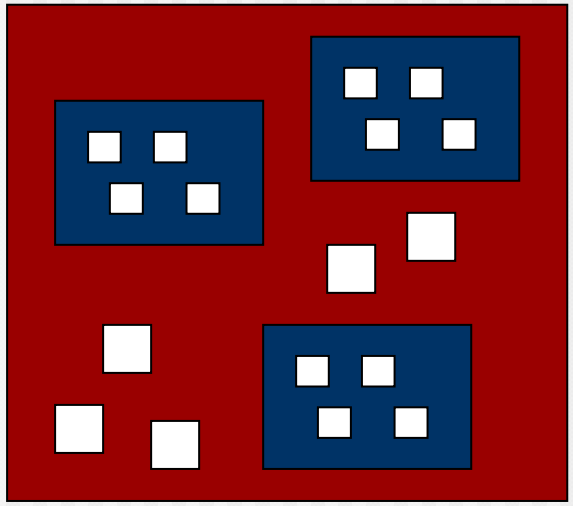
- Strives to create models that can be understood by individuals with diverse levels of business and technical understanding.
- The service-oriented modeling paradigm advocates taking a holistic view of the analysis, design, and architecture of all 'Software Entities' in an organization.
- Service-oriented modeling encourages viewing software entities as 'assets' (service-oriented assets), and refers to these assets collectively as 'services'.

Atomic service

Composite service

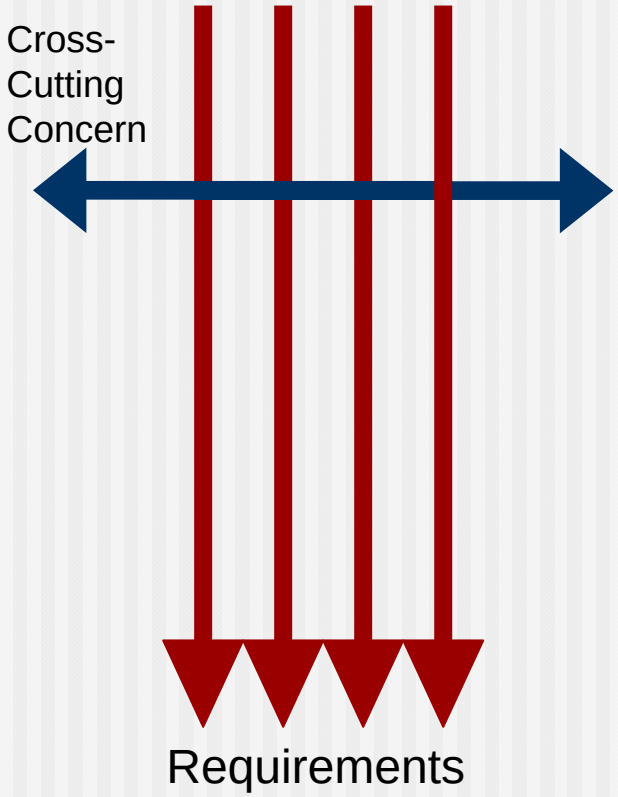


Service Cluster

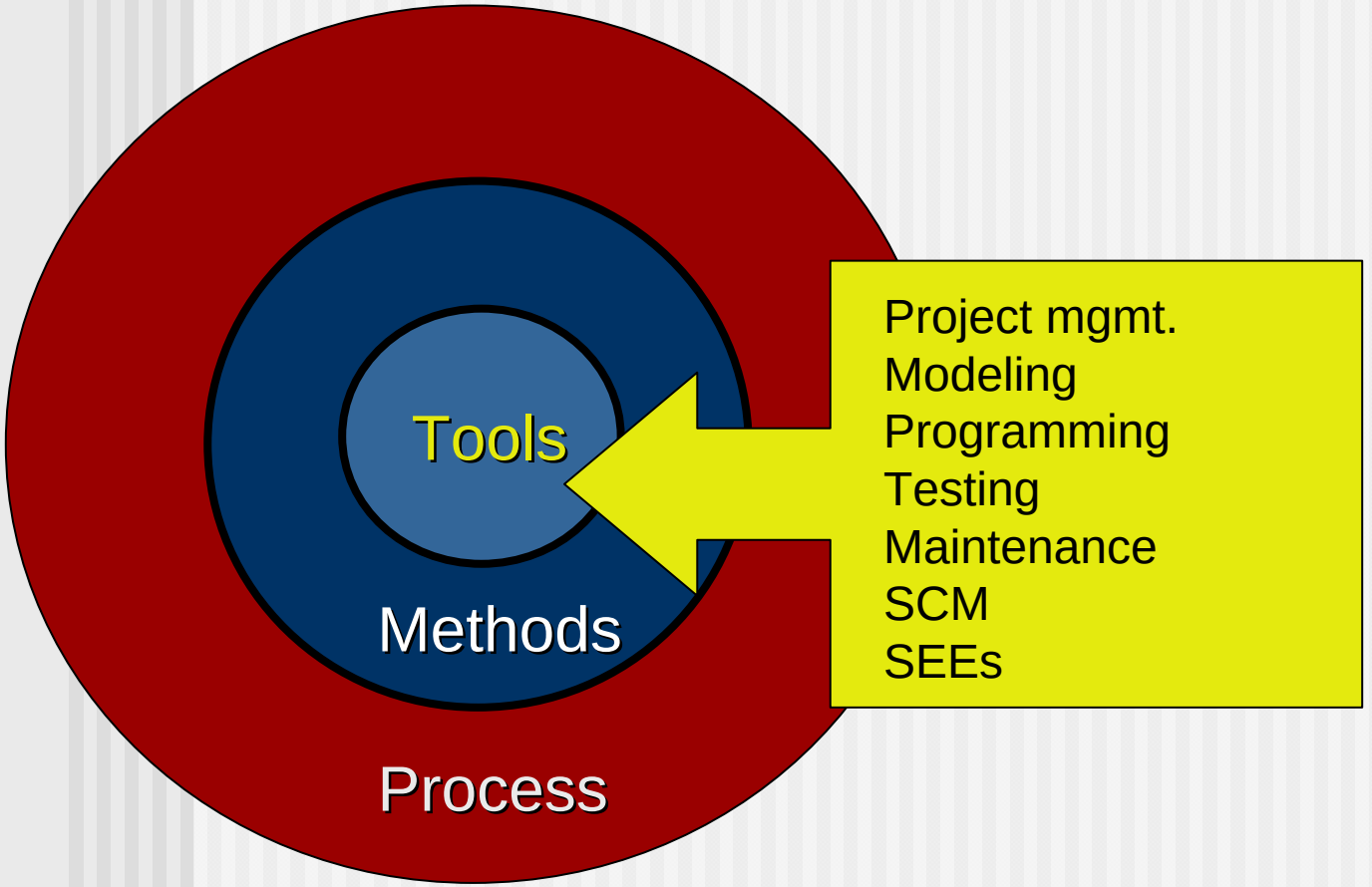


Aspect-Oriented Development

- *Concerns*—customer required properties or areas of technical interest
- *Cross cutting concerns*—concerns cut across multiple system functions, features, and information
- AOD provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*—mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern



SE Trends—Tools



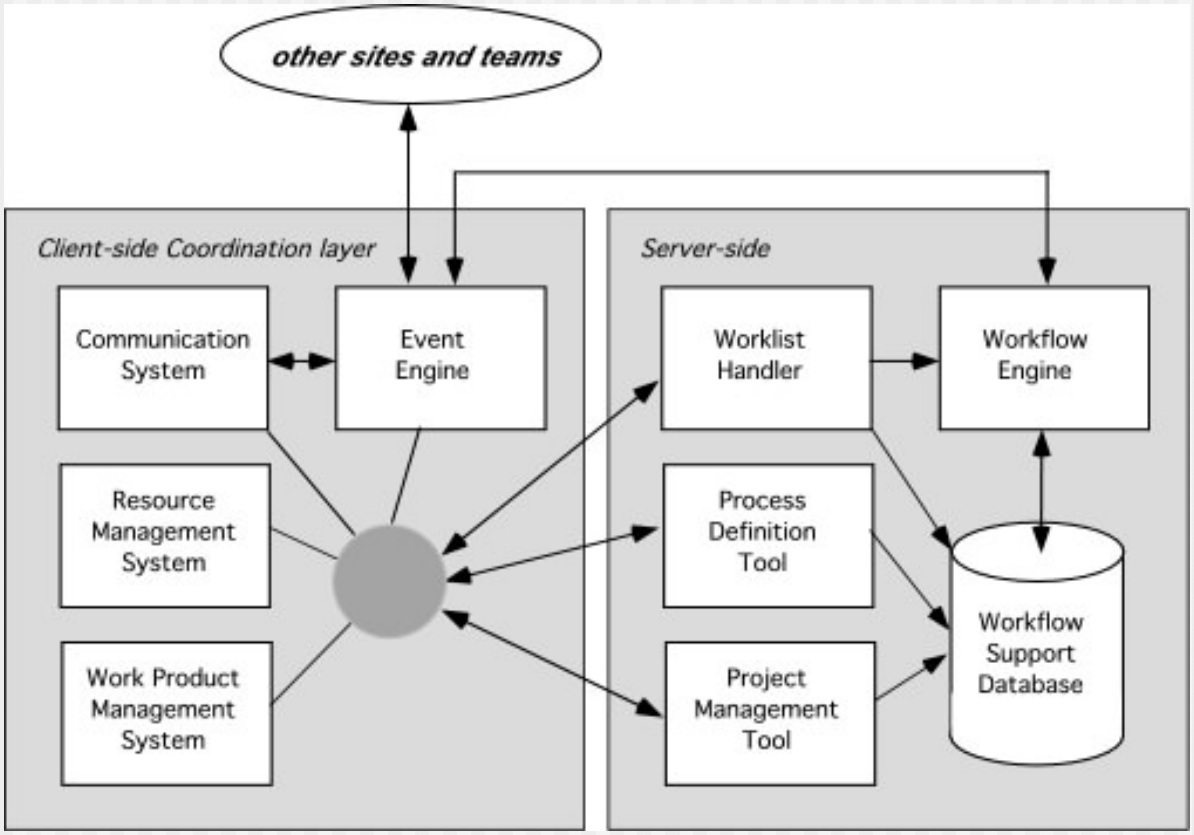
Tools Trends—RE and Modeling

- Requirements engineering tools will combine voice recognition input with “text mining” to extract requirements from informal information sources
- As pervasive computing becomes commonplace, design modeling tools must allow the designer to consider the architecture and behavior of the software *and* the physical properties of the devices on which the software resides

Tools Trends—Testing

- As test-driven development approaches gain momentum, tools for selecting test cases based on requirements and/or models must be developed. In addition, the software engineering community needs better tools for:
  - Launching selected tests in multiple device environments
  - Assessing the test outcome (sometimes called a “test oracle”)
  - Evaluating the impact of failures *and* determining the root cause of the failure
  - Determining whether the testing regime is sufficient

Tools Trends—SEE



The Road Ahead

- "We're driving faster and faster into the future, trying to steer by using only the rear-view mirror." *Marshall McLuhan*
- "Artificial intelligence is about at the same place the personal computer industry was in 1978." *BBC News*
- Prediction by consensus: *similar to* www.hubdub.com
- "…the best way to predict the future is to invent it." *Alan Kay*

Additional Information

- A new edition of my book covers trends in detail:
  - **Pressman R.S., *Software Engineering: A Practitioner's Approach,* McGraw-Hill, 2009** (available in late January, 2009)
  - Links to many Software Engineering Trends resources can be found at: **www.rspa.com/spi**
  - This presentation can be downloaded from:
    - **www.rspa.com/download/JaSSTKeynote.ppt**