CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS346: SOFTWARE ENGINEERING

Advanced Topics in Software Engineering

DEPSTAR

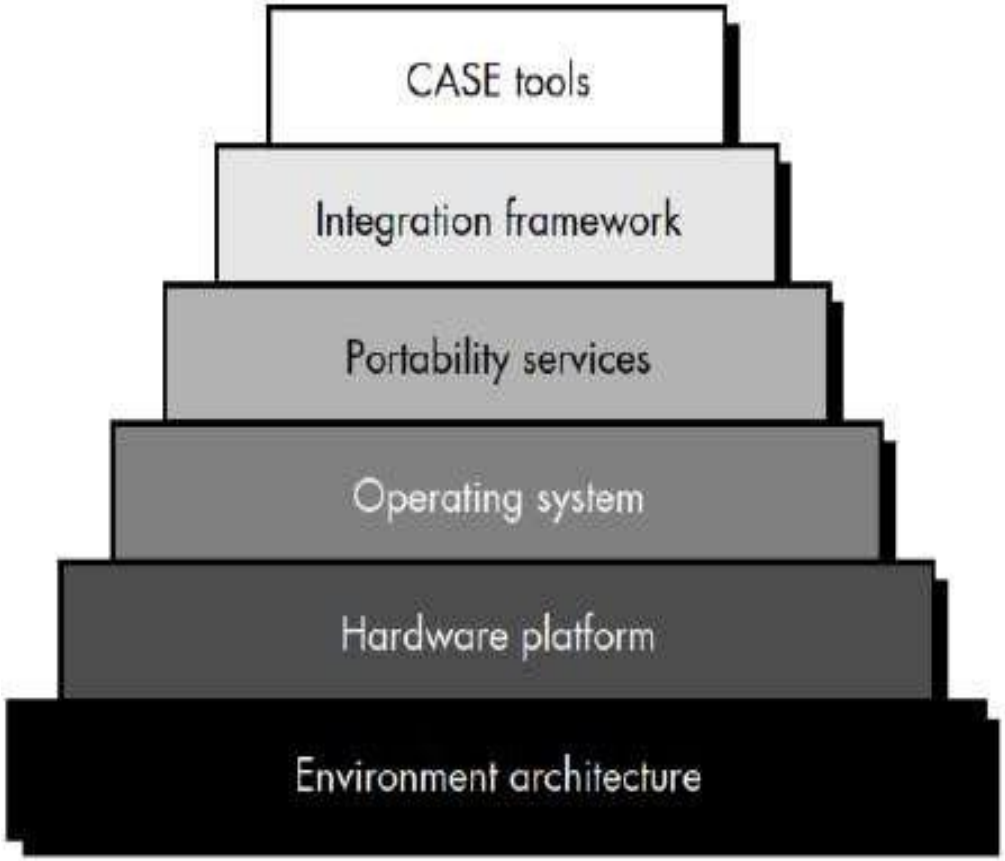
Content

- Component-Based Software Engineering, Client/Server Software Engineering, Web Engineering, Reengineering, Computer-Aided Software Engineering
- Software Process Improvement
- Emerging Trends in Software Engineering

Computer- Aided Software Engineering (CASE)

- Computer-aided software engineering (CASE) tools assist software engineering managers and practitioners in every activity associated with the software process.
- They automate project management activities, manage all work products produced throughout the process, and assist engineers in their analysis, design, coding and test work.
- CASE tools can be integrated within a sophisticated environment. CASE provides the software engineer with the ability to automate manual activities and to improve engineering insight.

CASE Tools Building Block



- The *environment architecture*, composed of the hardware platform and system support (including networking software, database management, and object management services), lays the ground work for CASE. But the CASE environment itself demands other building blocks.
- A set of *portability services* provides a bridge between CASE tools and their integration framework and the environment architecture.
- The *integration framework* is a collection of specialized programs that enables individual CASE tools to communicate with one another, to create a project database, and to exhibit the same look and feel to the end-user.
- Portability services allow CASE tools and their integration framework to migrate across different *hardware platforms and operating systems* without significant adaptive maintenance.

Contd...

- ThebuildingblocksdepictedinFigurerepresentacomprehensivefoundationfortheintegrationofCASEtools. However, most CASE tools in use today have not been constructed using all these building blocks.
- In fact, some CASE tools remain "point solutions." That is, a tool is used to assist in a particular software engineering activity (e.g., analysis modeling) but does not directly communicate with other tools, is not tied into a project database, is not part of an integrated CASE environment (I-CASE).
- Although this situation is not ideal, a CASE tool can be used quite effectively, even if it is a point solution

A Taxonomy of Case Tools

- CASE tools can be classified by function, by their role as instruments for managers or technical people, by their use in the various steps of the software engineering process, by the environment architecture (hardware and software) that supports them, or even by their origin or cost.
- The taxonomy presented here uses function as a primary criterion:

Business process engineering tools	Analysis and design tools
Process modeling and management tools	PRO/SIM (prototyping and simulation) tools
Project planning tools	Interface design and development tools
Risk analysis tools	Prototyping tools
Project management tools	Programming tools
Requirements tracing tools	Web development tools
Metrics and management tools	Integration and testing tools
Documentation tools	Static analysis tools
System software tools	Dynamic analysis tools
Quality assurance tools	Test management tools
Database management tools	Client/server testing tools
Software configuration management tools	Reengineering tools

Component-Based Software Engineering

- Component-based software engineering (CBSE) is a process that emphasizes the design and construction of computer-based systems using reusable software "components."
- CBSE seems quite similar to conventional or object-oriented software engineering.
- The process begins when a software team establishes requirements for the system to be built using conventional requirements elicitation techniques.

Activities

- **Componentqualification.**Systemrequirementsandarchitecturedefinethecomponentstha twillberequired. Reusable components (whether COTS or in-house) are normally identified by the characteristics of their interfaces.
- **Component adaptation.** Software architecture represents design patterns that are composed of components (units of functionality), connections, and coordination.
- In essence the architecture defines the design rules for all components, identifying modes of connection and coordination.
- **Component composition.** Architectural style again plays a key role in the way in which software components are integrated to form a working system.
- **Component update.** When systems are implemented with COTS components, update is complicated by the imposition of a third party (i.e., the organization that developed the reusable component may be outside the Immediate control of the software engineering organization).

Various Components

- Component—a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture.
- Run-time software component a dynamic bindable package of one or more programs managed a unit and accessed through documented interfaces that can be discovered in runtime.
- Softwarecomponent—aunitofcompositionwithcontractuallyspecifiedandexplicitcontextdependencies
- only.
- Business component—the software implementation of an "autonomous" business concept or business process.
- Qualified components—assessed by software engineers to ensure that not only functionality, but performance, reliability, usability, and other quality factors follow to the requirements of the system or product to be built.
- Adapted components—adapted to modify(also called mask or wrap)unwanted or undesirable characteristics.
- Assembled components—integrated into an architectural style and interconnected with an appropriate infrastructure that allows the components to be coordinated and managed effectively.
- Updated components—replacing existing software as new versions of components become available.

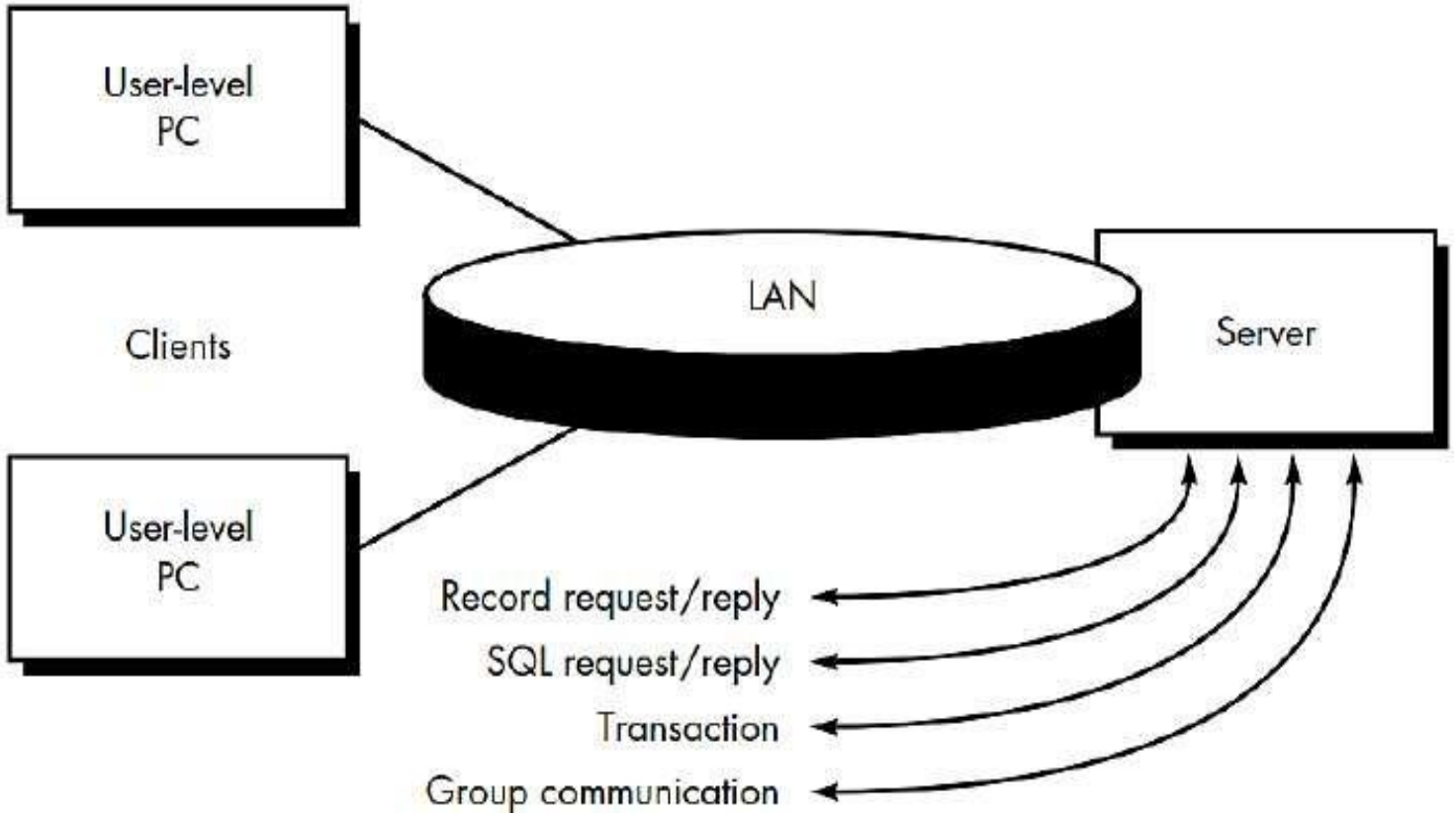
Client/Server Software Engineering

- There are different software process models were introduced.
- Although any of them could be adapted for use during the development of software for c/s systems, two approaches are most commonly used:
  - An evolutionary paradigm that makes use of event-based and/or object-oriented software engineering.
  - Component based software engineering that draws on a library of COTS and in-house software components.
- Client/server systems are developed using the classic software engineering activities analysis, design, construction, and testing—as the system evolves from a set of general business requirements to a collection of validated software components that have been implemented on client and server machines.

Software Components for Client/Server Systems

- Instead of viewing software as a monolithic application to be implemented on one machine, the software that is appropriate for a c/s architecture has several distinct subsystems that can be allocated to the client, the server, or distributed between both machines:
- User interaction/presentation subsystem. This subsystem implements all functions that are typically associated with a graphical user interface.
- Application subsystem: This subsystem implements the requirements defined by the application with in the context of the domain in which the application operates.
- Database management subsystem. This subsystem performs the data manipulation and management required by an application. Data manipulation and management may be as simple as the transfer of a record or as complex as the processing of sophisticated SQL transactions

Structure of Client/Server System



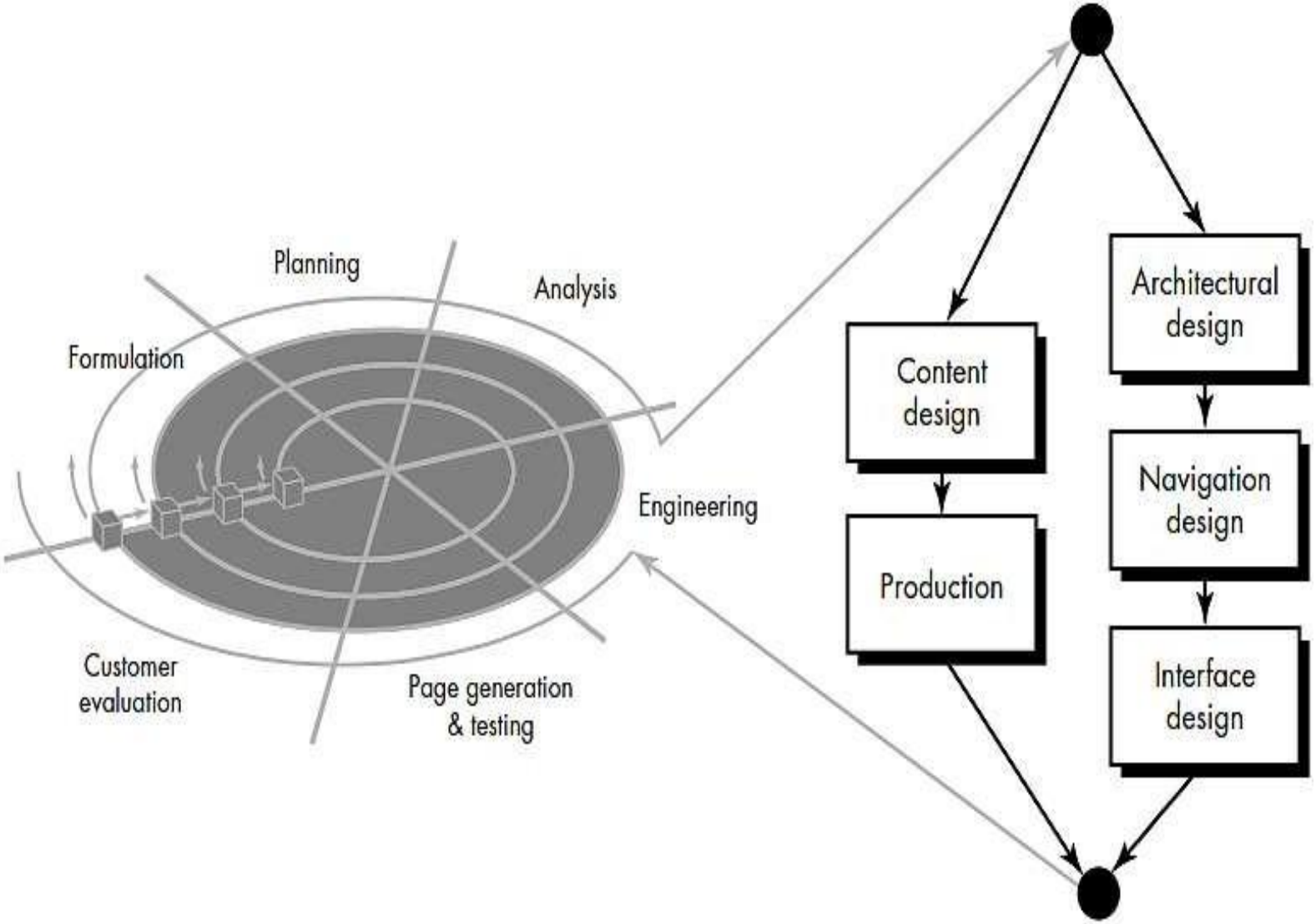
- **Fileservers :** The client requests specific records from a file. The server transmits these records to the client across the network.
- **Database servers:** The client sends structured query language (SQL) requests to the server. These are transmitted as messages across the network. The server processes the SQL request and finds the requested information, passing back the results only to the client.
- **Transaction servers:** The client sends are quest that invokes remote procedures at the server site. The remote procedures are a set of SQL statements. A transaction occurs when a request results in the execution of the remote procedure with the result transmitted back to the client.
- **Groupware servers:** When the server provides a set of applications that enable communication among clients (and the people using them)using text, images, bulletin boards, video, and other representations, a groupware architecture exists.

Web Engineering

- The characteristics of Web-based systems and applications have a profound influence on the WebEprocess.
- Immediacy and continuous evolution dictate an iterative, incremental process model that produces WebApp releases in rapid fire sequence.
- The network-intensive nature of applications in this domain suggests a population of users that is diverse (thereby making special demands on requirements elicitation and modeling) and an application architecture that can be highly specialized (thereby making demands on design).
- Because WebApps are often content driven with an emphasis on aesthetics, it is likely that parallel development activities will be scheduled within the WebE process and involve a team of both technical and non-technical people (e.g., copywriters, graphic designers).

Framework for Web Engineering

- As WebApps evolve from static, content-directed information sources to dynamic, user-directed application environments, the need to apply solid management and engineering principles grows in importance.
- To accomplish this, it is necessary to develop a WebE framework that encompasses an effective process model, populated by framework activities and engineering tasks.
- A process model for WebE is suggested in Figure on next slide


Planning
Analysis
Formulation
Engineering
Customer evaluation
Page generation & testing
Content design
Architectural design
Navigation design
Production
Interface design

- The WebE process begins with a *formulation*—an activity that identifies the goals and objectives of the WebApp and establishes the scope for the first increment.
- *Planning* estimates overall project cost, evaluates risks associated with the development effort, and defines a finely granulated development schedule for the initial WebApp increment, with a more coarsely granulated schedule for subsequent increments.
- *Analysis* establishes technical requirements for the WebApp and identifies the content items that will be incorporated.
- Requirements for graphic design (aesthetics) are also defined.
- The *engineering* activity incorporates two parallel tasks illustrated on the right side of Figure. *Content design and production* are tasks performed by nontechnical members of the WebEteam.
- The intent of these tasks is to design, produce, and/or acquire all text, graphics, audio, and video content that are to become integrated into the WebApp.

Contd…

- *Page generation* is a construction activity that makes heavy use of automated tools for WebApp creation.
- *Testing* exercises WebApp navigation; attempts to uncover errors in applets, scripts, and forms; and helps ensure that the WebApp will operate correctly in different environments (e.g., with different browsers).
- Each increment produced as part of the WebE process is reviewed during *customer evaluation*.
- This is the point at which changes are requested (scope extensions occur).
-

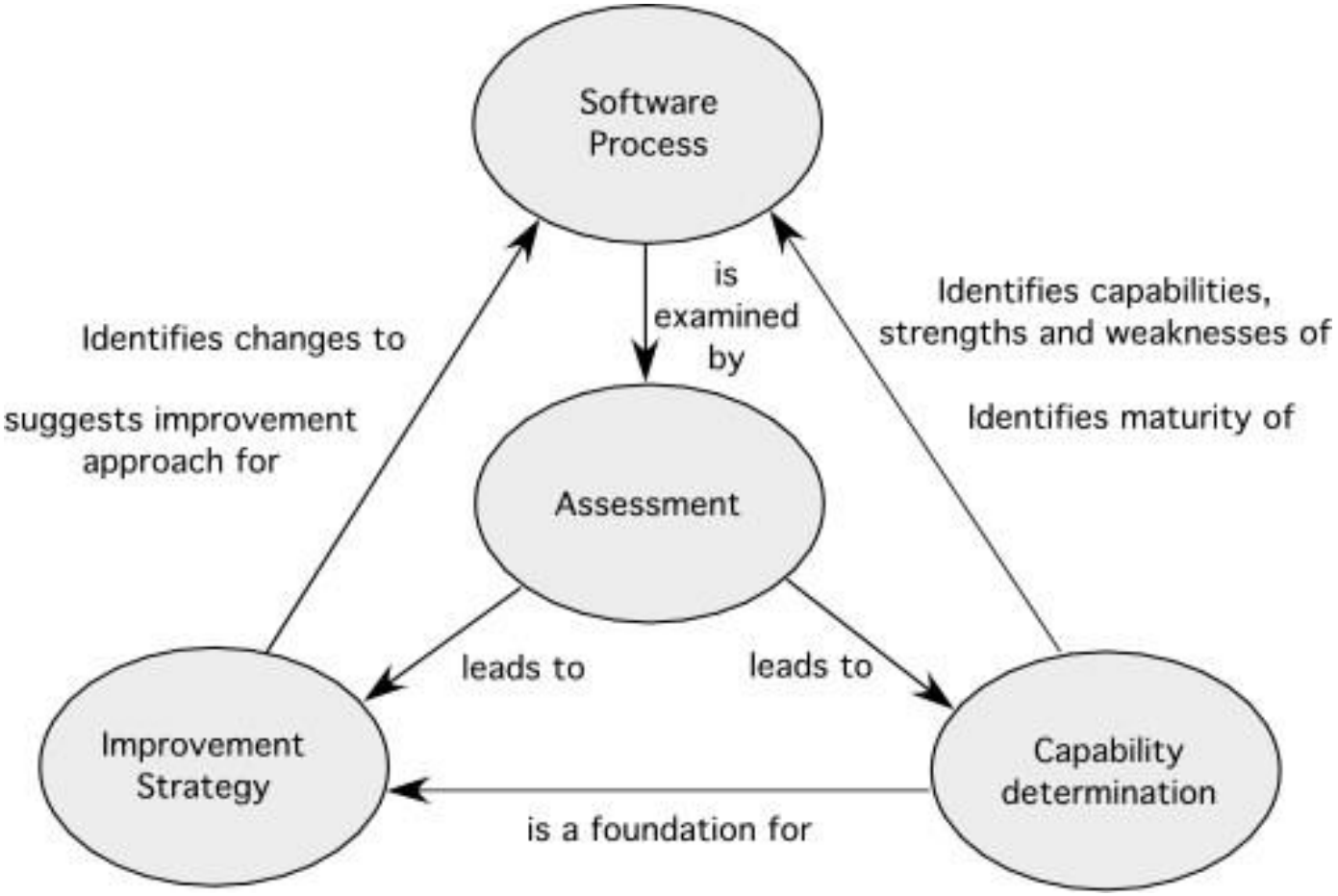
Software Process Improvement (SPI)

- SPI implies that
  - elements of an effective software process can be defined in an effective manner
  - an existing organizational approach to software development can be assessed against those elements, and
  - a meaningful strategy for improvement can be defined.
- The SPI strategy transforms the existing approach to software development into something that is more focused, more repeatable, and more reliable (in terms of the quality of the product produced and the timeliness of delivery).

SPI Framework

- a set of characteristics that must be present if an effective software process is to be achieved
- a method for assessing whether those characteristics are present
- a mechanism for summarizing the results of any assessment, and
- a strategy for assisting a software organization in implementing those process characteristics that have been found to be weak or missing.
- An SPI framework assesses the "maturity" of an organization's software process and provides a qualitative indication of a maturity level.

Elements of a SPI Framework



Constituencies

- Quality certifiers
  - Quality(Process) --> Quality(Product)
- Formalists
- Tool advocates
- Practitioners
- Reformers
- Ideologists

Maturity Models

- A *maturity model* is applied within the context of an SPI framework.
- The intent of the maturity model is to provide an overall indication of the "process maturity" exhibited by a software organization.
  - an indication of the quality of the software process, the degree to which practitioner's understand and apply the process,
  - the general state of software engineering practice.

Is SPI for Everyone?

- Can a small company initiate SPI activities and do it successfully?
  - Answer: a qualified "yes"
- It should come as no surprise that small organizations are more informal, apply fewer standard practices, and tend to be self-organizing.
  - SPI will be approved and implemented only after its proponents demonstrate *financial leverage.*

The SPI Process—I

- **Assessment and Gap Analysis**
  - *Assessment* examines a wide range of actions and tasks that will lead to a high quality process.
    - **Consistency.** Are important activities, actions and tasks applied consistently across all software projects and by all software teams?
    - **Sophistication.** Are management and technical actions performed with a level of sophistication that implies a thorough understanding of best practice?
    - **Acceptance.** Is the software process and software engineering practice widely accepted by management and technical staff?
    - **Commitment.** Has management committed the resources required to achieve consistency, sophistication and acceptance?
  - *Gap analysis*—The difference between local application and best practice represents a "gap" that offers opportunities for improvement.

Contd…

- *Education and Training*
- Three types of education and training should be conducted:
  - **Generic concepts and methods.** Directed toward both managers and practitioners, this category stresses both process and practice. The intent is to provide professionals with the intellectual tools they need to apply the software process effectively and to make rational decisions about improvements to the process.
  - **Specific technology and tools.** Directed primarily toward practitioners, this category stresses technologies and tools that have been adopted for local use. For example, if UML has been chosen for analysis and design modeling, a training curriculum for software engineering using UML would be established.
  - **Business communication and quality-related topics.** Directed toward all stakeholders, this category focuses on "soft" topics that help enable better communication among stakeholders and foster a greater quality focus.

Contd…

- **Selection and Justification**
  - choose the process model (Chapters 2 and 3) that best fits your organization, its stakeholders, and the software that you build
  - decide on the set of framework activities that will be applied, the major work products that will be produced and the quality assurance checkpoints that will enable your team to assess progress
  - develop a work breakdown for each framework activity (e.g., modeling), defining the task set that would be applied for a typical project
  - Once a choice is made, time and money must be expended to install it within an organization and these resource expenditures should be justified.

Contd…

- **Installation/Migration**
  - actually *software process redesign* (SPR) activities. Scacchi [Sca00] states that "SPR is concerned with identification, application, and refinement of new ways to dramatically improve and transform software processes."
  - three different process models are considered:
    - the existing ("as-is") process,
    - a transitional ("here-to-there") process, and
    - the target ("to be") process.

Contd…

- **Evaluation**
  - assesses the degree to which changes have been instantiated and adopted,
  - the degree to which such changes result in better software quality or other tangible process benefits, and
  - the overall status of the process and the organizational culture as SPI activities proceed
- From a qualitative point of view, past management and practitioner attitudes about the software process can be compared to attitudes polled after installation of process changes.

Risk Management for SPI

- manage risk at three key points in the SPI process [Sta97b]:
  - prior to the initiation of the SPI roadmap,
  - during the execution of SPI activities (assessment, education, selection, installation), and
  - during the evaluation activity that follows the instantiation of some process characteristic.
- In general, the following categories [Sta97b] can be identified for SPI risk factors:
    - budget and cost
    - content and deliverables culture
    - maintenance of SPI deliverables
    - mission and goals
    - organizational management and organizational stability
    - process stakeholders
    - schedule for SPI development
    - SPI development environment and process
    - SPI project management and SPI staff

Critical Success Factors

- The top five CSFs are [Ste99]:
  - Management commitment and support
  - Staff involvement
  - Process integration and understanding
  - A customized SPI strategy
  - A customized SPI strategy

The CMMI

- a comprehensive process meta-model that is predicated on a set of system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity
- a process meta-model in two different ways: (1) as a "continuous" model and (2) as a "staged" model
- defines each process area in terms of "specific goals" and the "specific practices" required to achieve these goals. *Specific goals* establish the characteristics that must exist if the activities implied by a process area are to be effective. *Specific practices* refine a goal into a set of process-related activities.

The People CMM

- "a roadmap for implementing workforce practices that continuously improve the capability of an organization's workforce." [Cur02]
- defines a set of five organizational maturity levels that provide an indication of the relative sophistication of workforce practices and processes

P-CMM Process Areas

Level	Focus	Process Areas
Optimized	*Continuous improvement*	Continuous workforce innovation Organizational performance alignment Continuous capability improvement
Predictable	*Quantifies and manages knowledge, skills and abilities*	Mentoring Organizational capability management Quantitative performance management Competency-based assets Empowered workgroups Competency integration
Defined	*Identifies and develops knowledge, skills and abilities*	Participatory culture Workgroup development Competency-based practices Career development Competency development Workforce planning Competency analysis
Managed	*Repeatable, basic people management practices*	Compensation Training and development Performance management Work environment Communication and coordination Staffing
Initial	*Inconsistent practices*	

Other SPI Frameworks

- **SPICE**— a international initiative to support the International Standard ISO/IEC 15504 for (Software) Process Assessment [ISO08]
- **Bootstrap**—a SPI framework for small and medium sized organizations that conforms to SPICE [Boo06],
- **PSP and TSP**—individual and team specific SPI frameworks ([Hum97], [Hum00]) that focus on process in-the-small, a more rigorous approach to software development coupled with measurement
- **TickIT**—an auditing method [Tic05] that assesses an organization compliance to ISO Standard 9001:2000

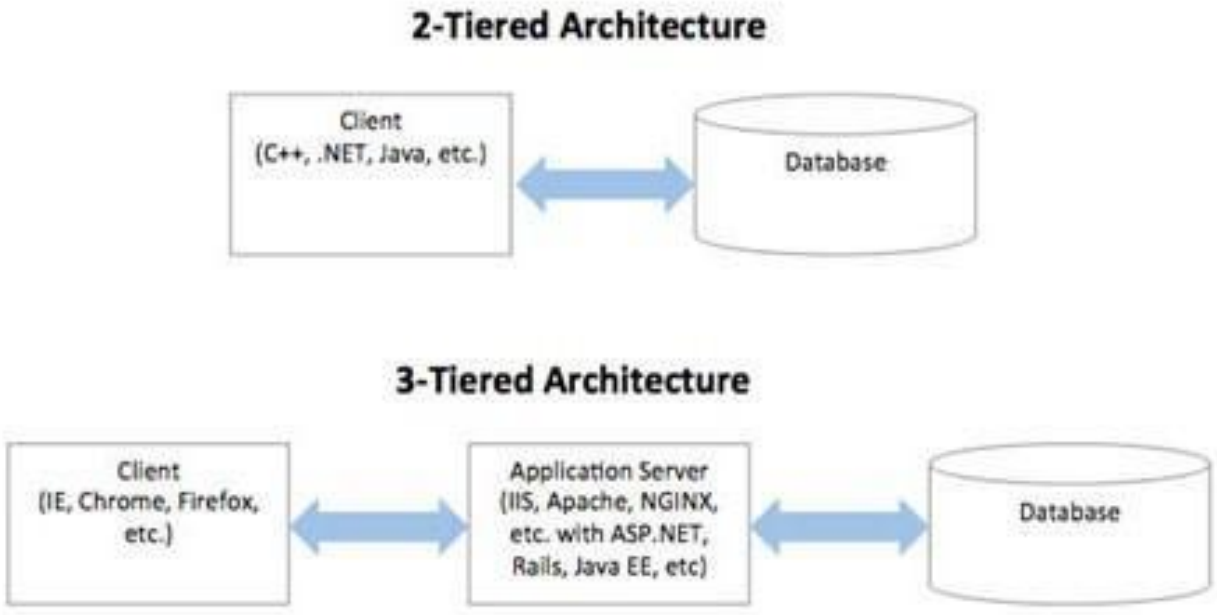
Emerging Trends in Software Engineering

- 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]

Contd…

- In response to the challenges faced, the following software engineering trends are becoming noticeable:
  - Client-Server Software
  - Service-Oriented Architecture (SOA)
  - Software as a service (SaaS)

Client-server Architectures



Contd…

- This application server is the middleware. The important activities are:
  - The middleware keeps track of the addresses of servers. Based on the client request, it can therefore easily locate the required server
  - It can translate between client and server formats of data and vice versa
- Two popular middleware standards:
  - Common Object Request Broker Architecture (CORBA)
  - COM/DOM

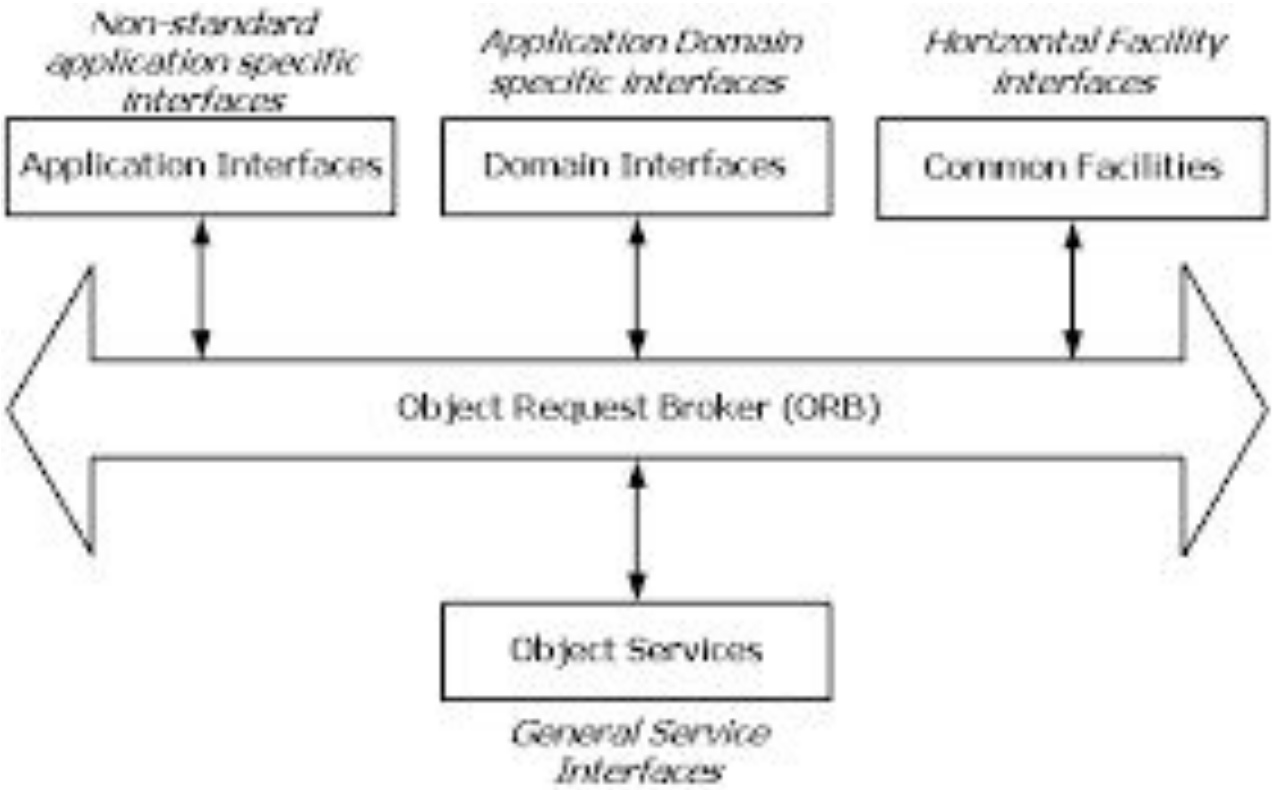
CORBA

- The **Common Object Request Broker Architecture** (**CORBA**) is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms.
- CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware.
- CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented.
- CORBA is an example of the distributed object paradigm.

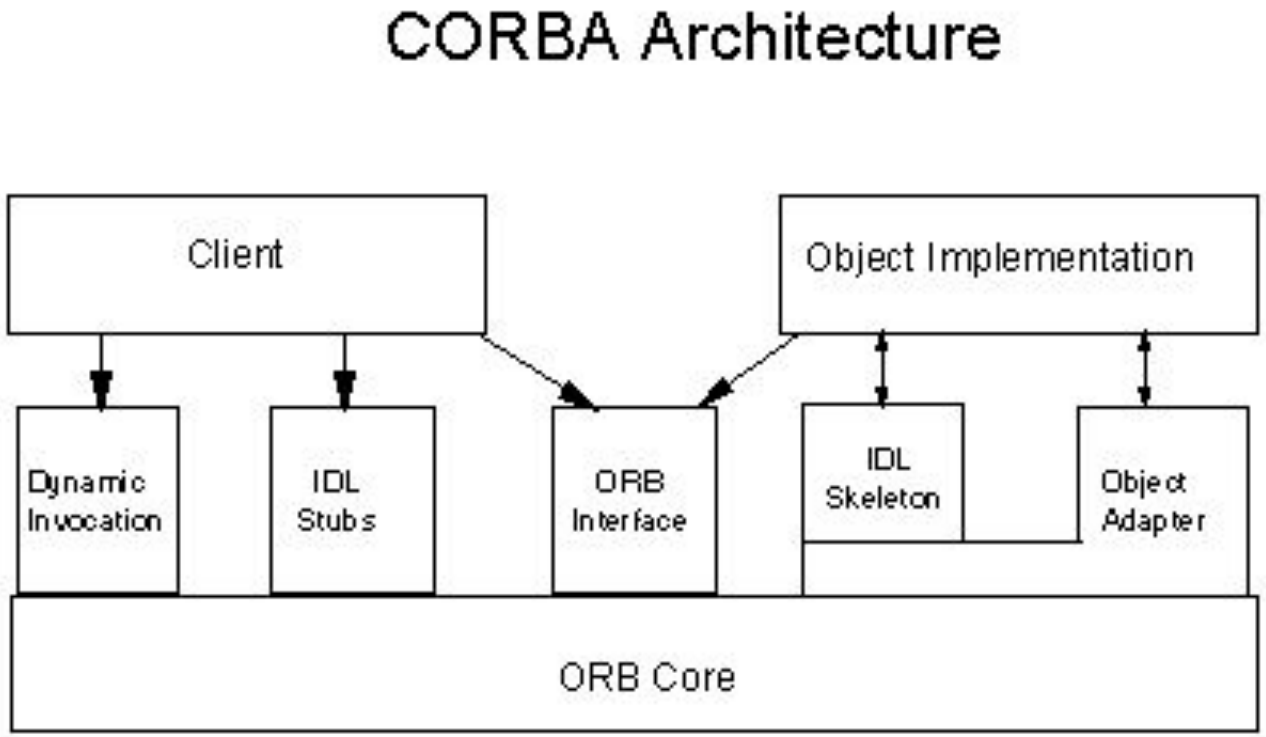
Contd…

- The CORBA specification dictates there shall be an ORB through which an application would interact with other objects. This is how it is implemented in practice:
- The application simply initializes the ORB, and accesses an internal *Object Adapter*, which maintains things like reference counting, object (and reference) instantiation policies, and object lifetime policies.
- The Object Adapter is used to register instances of the *generated code classes*. Generated code classes are the result of compiling the user IDL code, which translates the high-level interface definition into an OS- and language-specific class base for use by the user application. This step is necessary in order to enforce CORBA semantics and provide a clean user process for interfacing with the CORBA infrastructure.

CORBA Reference Model



CORBA ORB Architecture

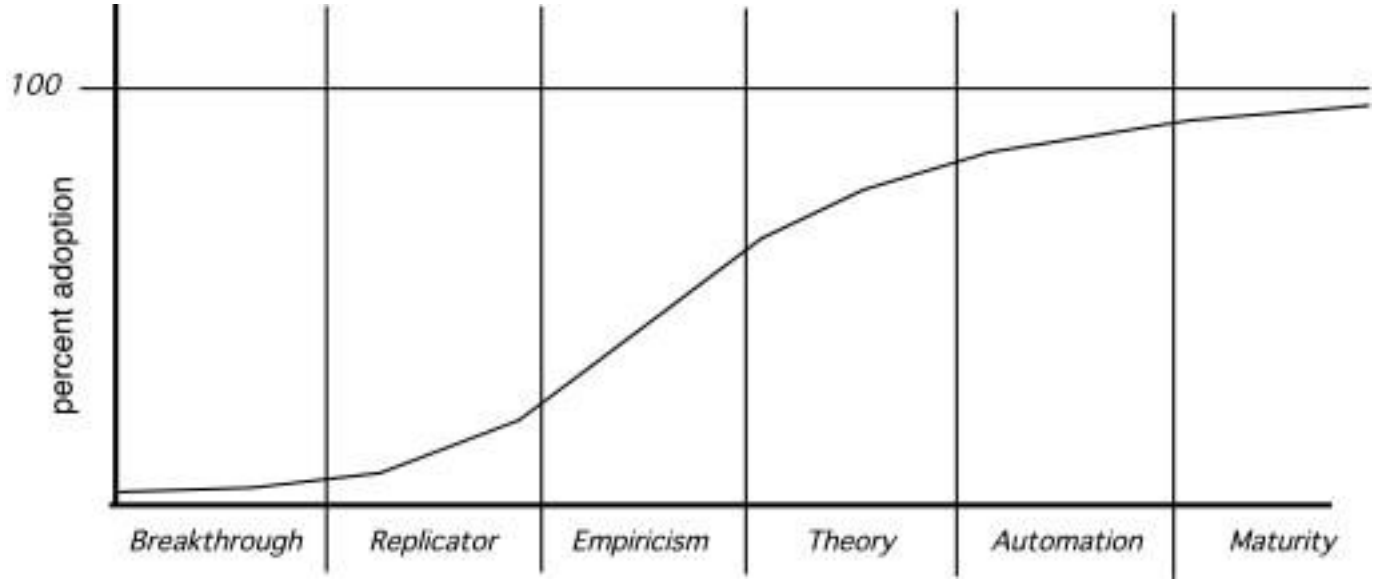


COM/DOM

- **Component Object Model (COM)**
  - Main idea is that different vendors can sell binary components.
  - It can be used to deploy component applications on a single computer.
  - The concepts are very similar to CORBA
  - The components are known as binary objects.
  - These can be generated used languages like Visual basic, visual C++ etc.
  - These languages have the necessary features to create COM components.
  - COM components are binary objects and they exist in the form of either .exe or .dll

- **Distributed Component Object Model (DOM):**
  - It is the extension of COM.
  - The restriction that clients and servers reside in the same computer is relaxed here.
  - So, DCOM can operate on networked computers.
  - Using DCOM, development is easy as compared to CORBA.
  - Much of the complexities are hidden from the programmer.

Technology Innovation Lifecycle



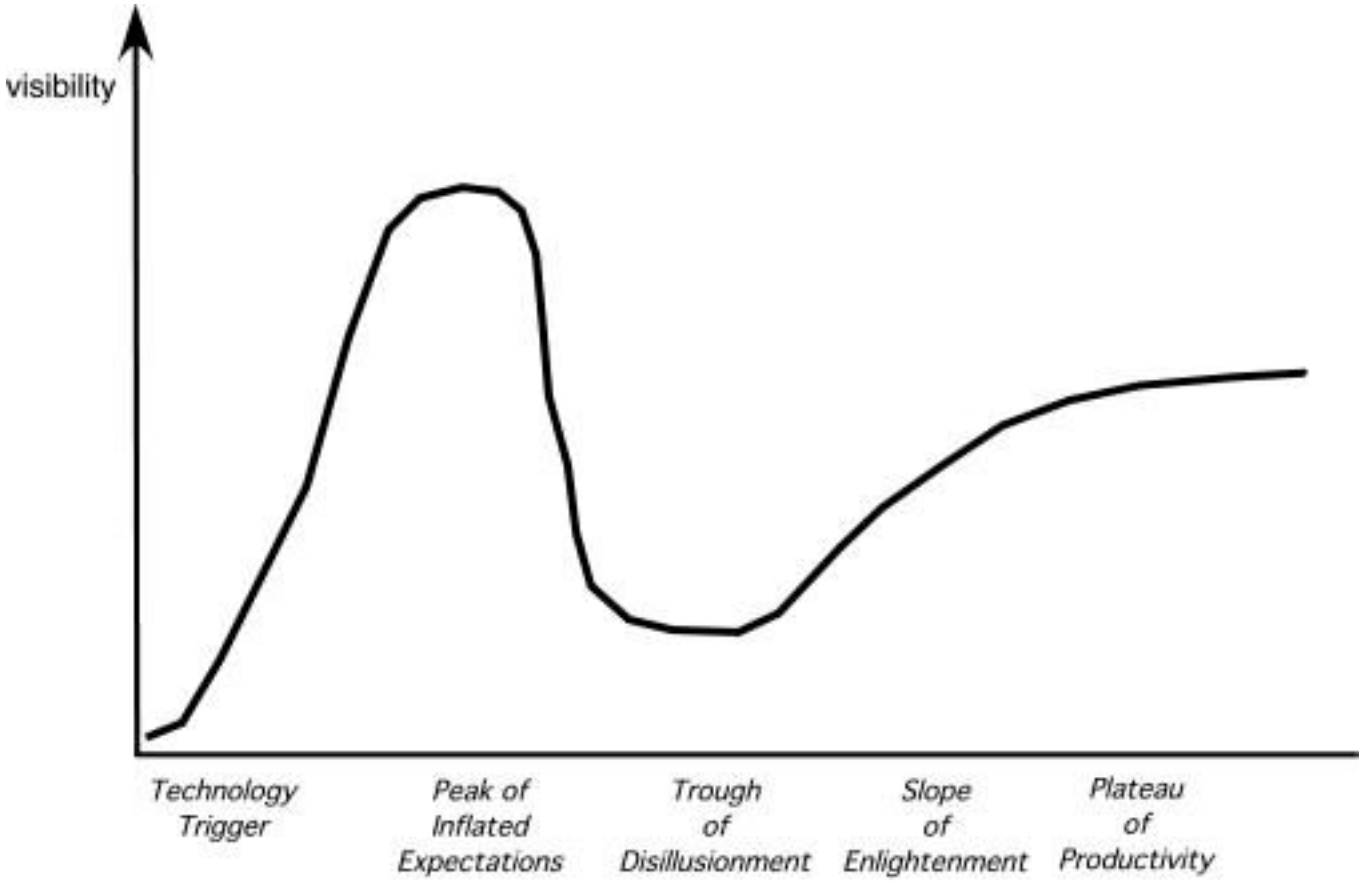
Observing SE Trends

- 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

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).

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

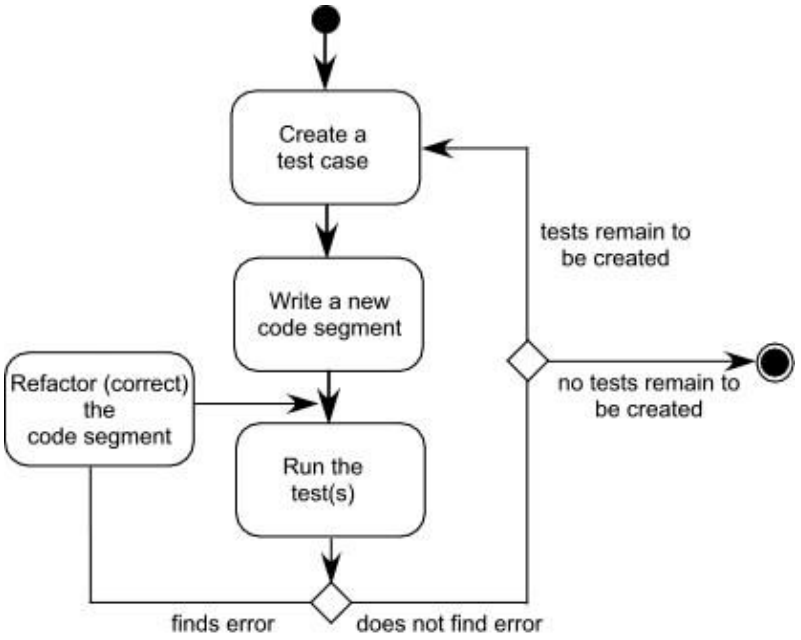
- Today, software engineers collaborate across time zones and international boundaries, and every one of them must share information.
- The challenge over the next decade is to develop methods and tools that facilitate that collaboration.
- Critical success factors:
  - Shared goals
  - Shared culture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

Model-Driven Development

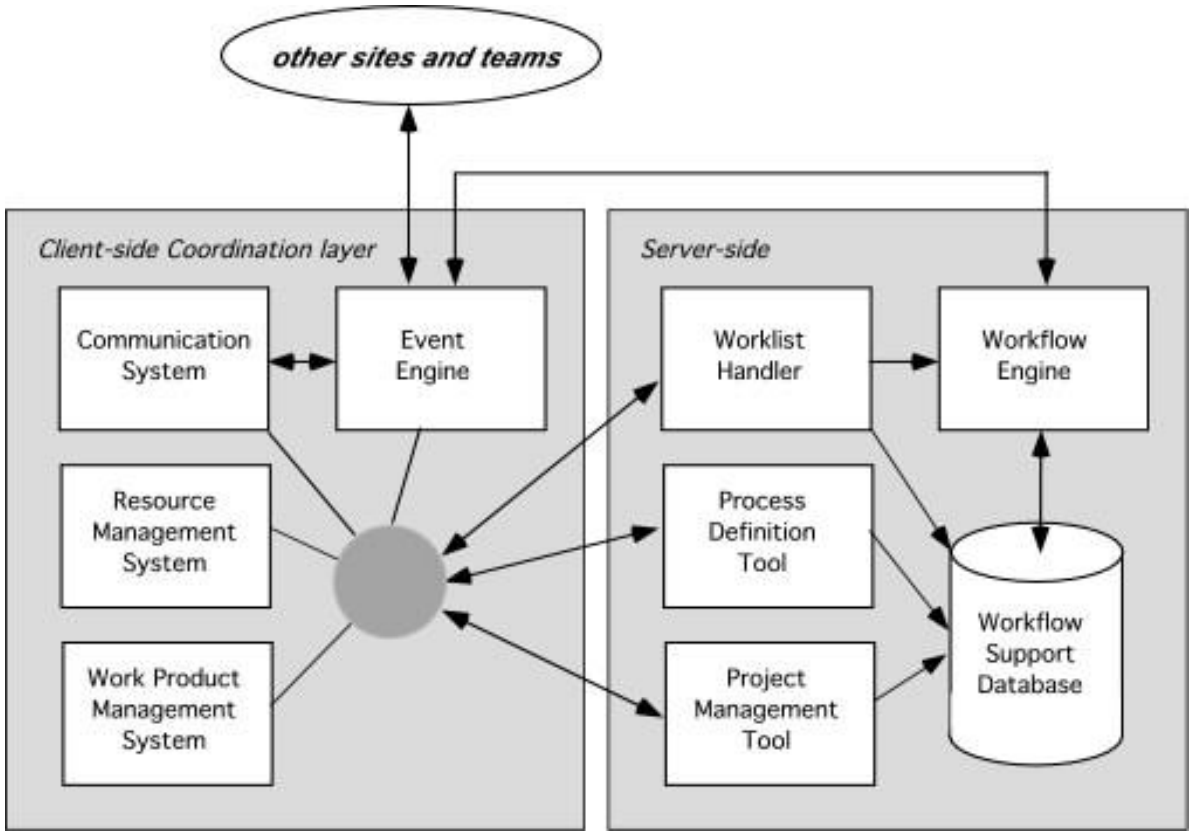
- couples domain-specific modeling languages with transformation engines and generators in a way that facilitates the representation of abstraction at high levels and then transforms it into lower levels [Sch06]
- *Domain-specific modeling languages* (DSMLs)
  - represent "application structure, behavior 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



Tools Trends—SEE




Thank you!