Software Engineering – Emerging Trends

12th August 2005
Kolkata
Agenda

- Introduction
- Setting the context
- Orientation of software engineering
- Discipline
- Overview of software processes
- Emerging methods – agile model
- Conclusions
Introduction

- **Definition of software engineering**
  - Branch of systems engineering
  - Development of large and complex software intensive systems
  - Focuses on real-world goals as well as services
  - Based on precise specification of system structure and behavior and the implementation of specifications
  - Activities required in order to develop an assurance that specifications and real-world goals have been met
  - It is also concerned with the processes, methods and tools for the development of software systems in an economic and timely manner.
Introduction

- **Objectives**
  - Bring out changes that are taking place in software engineering
  - Methods and processes used in software engineering
  - Highlight how technology changes in the last two decades has impacted the traditional approach.
Introduction

- **Presentation Structure**
  - Discuss changing context of software system development
  - Discuss changing orientation of software engineering
  - Evolution of this discipline
  - Analyze software development processes & methods
  - Then we focus on emerging models such as agile development
Context
Systems are rarely developed from scratch

System Development is now focused on

- Extension of preexisting systems
- Integration with ‘legacy’ infrastructure
- Embedded in complex, highly dynamic, decentralized organizations
- Required to support business & industrial processes, which are continually reorganized to meet changing consumer demands

Services provided by such systems satisfy the needs of diverse stakeholders.
Very few ‘bespoke’ software systems are being constructed.  
Instead, generic components are built & sold  
Development really focus on configuration and interoperability.  
Examples:  
- SAP  
- Oracle Applications  
- Siebel
Resultant systems are composed from autonomous, locally managed, heterogeneous components, which are required to provide complex services

- Examples
  - Open Source Products
  - APIs

- New & constantly changing business models relating to the provision of software & software enabled services resulting from Internet & eCommerce Technologies
Another paradigm shift is being seen, relating to ‘no coding’ required products.

- Examples:
  - Agentis
  - Skyway Builder

One major SDLC activity known as ‘Development’ / ‘Construction’ phase is likely to become very minimal.
Changing Context

Summary

- Overall setting is characterized by
  - Increasing business dependence on reliable software infrastructure
  - As well as rapid change and reconfiguration of business services
  - Resulting in rapid software development and frequent changes to the software infrastructure.
Orientation
Orientation Changes

- **Focus of software solution use**
  - Targeted towards ‘real industrial problems’ as compared to labor saving activities
  - Aiding in Decision support, business intelligence type of services.

- **Entailing thorough problem analysis, thus need for domain knowledge**

- **Light weight solutions and scalability**
  - From Palmtops to Teraflop devices
Orientation Changes

- Increased maturity for software engineering itself acting as ‘validity’ barrier for any further radical changes.
- Industry seeking substantial evidence of value from such changes to processes & methods.
- Challenges of scalability arising out of Internet as well as large volume data.
Discipline
Unlike ‘Engineering industries’, software industry is about 40 years old.

Process maturity and discipline is about 10-15 years old.
  - Influenced by ISO in early 1990s

Historically defined itself in terms of testing and debugging

Failures such as London Ambulance service Computer aided dispatch system, Ariane 5 can lead to negative orientation.

Time to market as well as pressing need to rapidly roll out a service to meet new business demands, forces change in outlook towards more positive ‘holistic’ view of the software engineering in delivering satisfaction to users.
Brief overview of software processes
Software Processes

- Developing software is a complex process
- It is just not programming alone, tools & technologies cannot deliver software
- Software development is a collective, complex and creative effort
- Quality of the product depends upon:
  - People
  - Processes
  - Organization
Software Development life cycle

- SDLC defines the different stages in the lifetime of a software solution
  - Requirement analysis
  - Design
  - Development
  - Verification & validation
  - Deployment
  - Operation
  - Maintenance and support
  - Retirement

- SDLC also defines the principles according to which various stages are executed
  - Waterfall model, spiral model
Software Development life cycle

- **Waterfall model**
  - Well defined entry & exit criteria for each stage
  - Specific stage can start only when the deliverables of the previous stage has been completed
  - Minimum customer involvement after requirement study, thus poses higher risks to be managed
  - Applicable for bespoke development projects

- **Spiral model**
  - Iterative processes
  - Higher customer involvement at each stage, thereby lower risks
  - Proximity to customer is very essential
Software Development life cycle

- SDLC defines the base processes with which software need to be developed
- It does not provide precise course of actions, tools and operating procedures, development policies and constraints
  - Example
    - Software configuration management
    - Technical reviews
    - Requirement collection & design techniques such as SSAD
- Thus, it is only a starting point but not enough to practically guide & control software project
Need for process frameworks

- Two decades of base level processes provided SDLC resulted in
  - Unfulfilled promises of productivity & quality
  - Gains from new technologies were neutralized owing to poor process application
  - Benefits of technology advancements and better processes cannot be realised in an undisciplined environments
  - Projects were often delayed, exceed the planned budgets.
  - Lack of infrastructure and support to help projects avoid these issues.

- Thus arose the need for organization-wide software process.
Software processes

- The coherent set of policies, organizational structures, technologies, procedures and artifacts that are needed to conceive, develop, deploy and maintain a software product.

- Requires
  - Technology support, tools, infrastructure that makes it economically feasible
  - Techniques, guidelines & templates to accomplish development activities.
  - Organizational behavior as software development is carried out by people that have to be coordinated & managed within an effective organizational structure
Software processes

- Process oriented approach to deliver software projects has helped establish effective software processes.

- Resulting, in a set of new initiatives such as:
  - Process Modelling
  - Tools to be used for process modelling
    - Unified Software development processes
    - Unified Modelling
    - Architectural models
  - Technologies
    - Rational Tools to enable Unified process framework
Software processes

- **Process Modeling**
  - Can be used to represent a precise way how a process is structured and organized.
  - Helps in removing inconsistencies and resolve ambiguities
  - Helps in designing new processes
  - Helps in training & education
  - Helps in simulation and optimisation
    - Identify bottlenecks and opportunities for improvements
Despite moving towards process led approach, there were key issues such as:

- Overheads in software development activities
- Change management & variations

Processes are not static & frozen once for all.
Requires continuous improvements to deal with changing market requirements & expectations

Hence needs continuous assessment and improvements
These in turn has led to creation of quality models & improvement methods

- ISO 9001 & CMM are reference models

In order to improve, you need to assess (determination of maturity of a process wrt to a quality model)
Metrics & Measurements

- Process Modeling & process improvements need to be based on reliable & effective practices. To improve, we need to know where we are
  - What are the indicators that tell us about quality of a process?
  - What are techniques are more effective to improve a specific process?

- Thus systematic evaluation of the quality of a process & its constituents (tools & procedures)

- Resulting in, development of techniques & methods related software metrics
Some of the key metric's are:

- Lines of Code
- Productivity expressed as KLOC/hr
- Sizing measures as Function Points
- Defects per function point
- Defect Density
- Programmer Productivity for various technologies expressed as Hours per function point

Resulting in process capability baseline for an organization.
Recap

- Changing context & orientation as well as inherent weaknesses of traditional SDLC models resulted in a process oriented software engineering methods
- Metrics & measurement techniques were used to determine process improvements
- ISO / CMM process frameworks helped software development to focus on better quality as well enabled organizations to benchmark & improve.
- Swing from one end to another end.
  - Lack of process orientation to regimented process orientation & better engineering practices
Emerging trends
The dichotomy

- **Process Pressures**
  - The ‘so called’ process driven approach puts huge burden on software projects
  - Rigorous process documentation is a pre-requisite
  - Leading to huge efforts to document every step
  - No lunch is free and all these efforts are priced increasing the total costs of ownership from a customer perspective.
  - ISO / CMM process compliance is estimated to add about 20 – 25% of software development efforts

- **Can there be a possible balance? How much process do we need? What do we keep and what do we discard?**
A barely sufficient methodology

- Streamlining the methodology to focus on the practice rather than the process.

- Two reasons behind this:
  - Value & Innovation
  - Ruthless elimination of non-value adding activities
  - Discarding such processes that slows down the team effort.

- Such thinking gave rise to a new methodology known as ‘Agile Methodology’ in 2002.
Agile Manifesto

- Individuals and Interactions over Processes & tools
- Working Software over comprehensive documentation
- Customer Collaboration over Contract negotiation
- Responding to change over following a plan
An agile model is a model that is just barely good enough, which implies that it exhibits the following traits:

- It fulfills its purpose.
- It is understandable.
- It is sufficiently accurate.
- It is sufficiently consistent.
- It is sufficiently detailed.
- It provides positive value.
- It is as simple as possible.
What is not Agile Model?

What Is(n’t) AM?

- AM is a supplement to existing methods, it is not a complete methodology.
- AM is a way to work together effectively to meet the needs of project stakeholders.
- AM is effective and is about being effective.
- AM is something that works in practice, it isn’t an academic theory.
- AM is not a silver bullet.
- AM is not a prescriptive process.
- AM is for the average developer, but is not a replacement for competent people.
- AM is not an attack on documentation, instead AM advises to create documents that have value.
- AM is not an attack on CASE tools.
- AM is not for everyone.
Principles

- Core Principles of AM
  - Assume Simplicity
  - Embrace Change
  - Incremental Change
  - Maximize Stakeholder Investment
  - Model With a Purpose
  - Multiple Models
  - Quality Work
  - Rapid Feedback
  - Software is Your Primary Goal
  - Travel Light

- Supplementary Principles of AM
  - Content Is More Important Than Representation
  - Open and Honest Communication
 Agile Models in play

- **Feature Driven Development (FDD)**
  - Emphasis on delivering feature by feature rather than all at one go.
  - Simple approach to processes
  - A feature is a small, client-valued function expressed in the form: for example, "Calculate the total of a sale,", "Validate the password of a user" and "Authorize the sales transaction of a customer".
  - FDD contains just enough process to ensure scalability and repeatability, all the while encouraging creativity and innovation.
Scrum Development Process

Roles:
- **PO**: Product Owner
  - Set priorities
- **SM**: ScrumMaster
  - Manage process, remove blocks
- **T**: Team
  - Develop product
- **SH**: Stakeholders
  - Observe & advise

Key Artifacts:
- **Product Backlog**
  - List of requirements & issues
  - Owned by Product Owner
  - Anybody can add to it
  - Only Product Owner prioritizes
- **Sprint Goal**
  - One-sentence summary
  - Declared by Product Owner
  - Accepted by team
- **Sprint Backlog**
  - List of tasks
  - Owned by team
  - Only team modifies it
- **Blocks List**
  - List of blocks & unmade decisions
  - Owned by ScrumMaster
  - Updated daily
- **Increment**
  - Version of the product
  - Shippable functionality (tested, documented, etc.)

Key Meetings:
- **Sprint Planning Meeting**
  - Hosted by ScrumMaster; ½-1 day
  - In: Product Backlog, existing product, business & technology conditions
  1. Select highest priority items in Product Backlog; declare Sprint Goal
  2. Team turns selected items into Sprint Backlog
  - Out: Sprint Goal, Sprint Backlog
- **Daily Scrum**
  - Hosted by ScrumMaster
  - Attended by all, but Stakeholders don’t speak
  - Same time every day
  - Team updates Sprint Backlog; ScrumMaster updates Blocks List
- **Sprint Review Meeting**
  - Hosted by ScrumMaster
  - Attended by all
  - Informal, 4-hour, informational
  - Team demos Increment
  - All discuss
  - Hold retrospective
  - Announce next Sprint Planning Meeting

Development Process:
- **Sprint**
  - 30 days each
  - Starts with Sprint Planning Meeting
  - Ends with Sprint Review Meeting

---

www.itcinfotech.com

An SEI CMM Level 5 Company
Conclusion
Shifting focus

- From processes to Practice
- From heavy documentation to iterative releases of working software
- From paper based code inspections to ‘Pair Programming’
- From price fixing contract negotiations to collaboration with customer
- From following a development plan to adapting quickly to change
Conclusions

- Software engineering innovations do take a longer time to percolate through to every day use.
- Despite this, changes such as OODM, CASE tools with code generation, development patterns, incremental delivery based life cycle, component models have reshaped the SE practice
- Future could be a setting of assembly line process with cookie cutter technologies where in each component is selected and integrated.