SEWP ZG622: Software Project Management (Lecture #1)

T V Rao, BITS-Pilani Off-campus Centre, Hyderabad

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Text Books

T1: Bob Hughes, Mike Cotterel, and Rajib Mall, Software Project Management, 5th Edition, McGraw Hill, 2011

T2: Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, McGraw Hill, 2010

R1: Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

R2: Capers Jones., Software Engineering Best Practices, TMH ©2010

R3: Robert K. Wysocki, Effective Project Management: Traditional, Agile, Extreme, John Wiley & Sons © 2014

R4: George Stepanek, Software Project Secrets : Why Software Projects Fail, Apress ©2012

R5: A Guide to the Project Management Body of Knowledge (PMBOK® Guide), Fifth Edition by Project Management Institute Project Management Institute © 2013

R6: Jake Kouns and Daniel Minoli, Information Technology Risk Management in Enterprise Environments. John Wiley & Sons © 2010

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

L1: Sotware Characterization

— Definitions, Software Characteristics, Challenges

Source Courtesy: Some of the contents of this PPT are sourced from materials provided by publishers of prescribed books

What is Software?

1) instructions (programs) that when executed provide desired function and performance
2) data structures that enable the programs to adequately manipulate information
3) documents that describe the operation and use of the programs

- A **logical** rather than physical system element

Importance of Software

- Software as a differentiator in the marketplace.
- Mechanism for automating business, industry, and government
- Medium for developing/transferring new technologies
- Means of capturing people's expertise for use by others
- Software is a hidden technology, embedded in daily activities and used without customers thinking about it

– Adapted from Roger Pressman

Dual Role of Software

- Both a product and a vehicle for delivering a product
  - Product
    - Delivers computing potential
    - Produces, manages, acquires, modifies, display, or transmits information
  - Vehicle
    - Supports or directly provides system functionality
    - Controls other programs (e.g., operating systems)
    - Effects communications (e.g., networking software)
    - Helps build other software (e.g., software tools)

Software Products

- **Generic products**
  - Stand-alone systems which are produced by a development organization and sold on the open market to any customer
  - The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.
    - Examples – PC software such as word processor, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

- **Bespoke (customized) products**
  - Systems which are commissioned by a specific customer and developed specially by some Contractor
  - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.
    - Examples –e-commerce application of an organization, embedded software for anti-lock braking systems.

- **Most software expenditure is on generic products but most development effort is on bespoke systems**

Software Applications

- System software
  - –Compiler, Operating System…
- Application software
  - –Railway reservation…
- Engineering/scientific Software
  - –Astronomy, automated manufacturing…
- Embedded software
  - –Braking system, Mobile phone…

Software Applications (contd)

- Product-line software
  - –Word processor, personal finance application…
- WebApps (Web applications)
  - –Social networks…
- AI software
  - –Pattern recognition, neural networks…

Software—New Categories

- Open world computing - pervasive, distributed Computing
- Netsourcing - the Web as a computing engine
- Open source - ”free” source code open to the computing community (a blessing, but also a potential curse!)

- Also …
  - Data mining
  - Grid computing
  - Cognitive machines

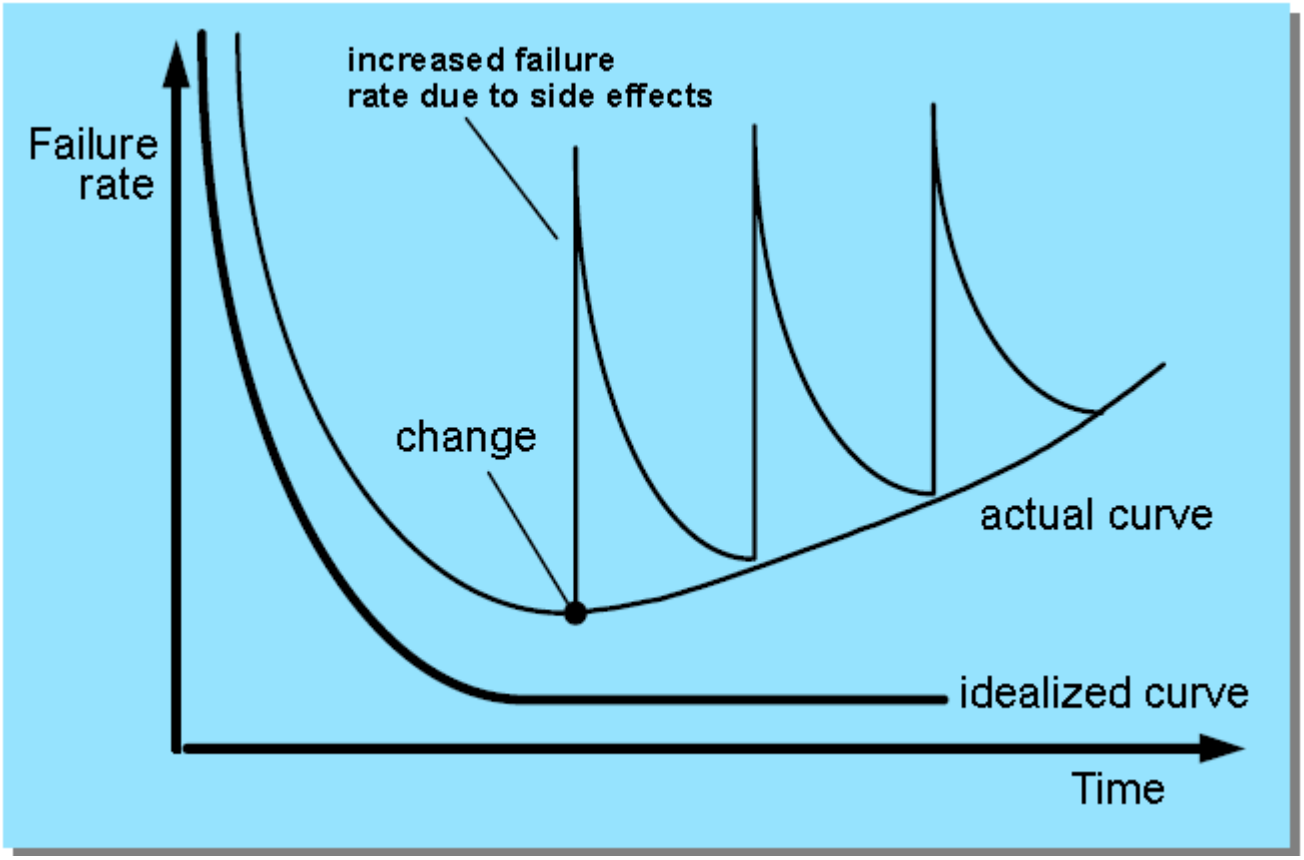
Software Characteristics

- Software is developed or engineered....Not manufactured in the classical sense
- Software does NOT “wear out”

  –Aging of software is very unlike material objects (hardware)
- Industry is moving towards component-based construction, but most of the software effort is for the custom-building

Failure ("Bathtub") Curve for Hardware

Software Deterioration



Evolving the Legacy Software

- (Adaptive) Must be adapted to meet the needs of new computing environments or more modern systems, databases, or networks
- (Perfective) Must be enhanced to implement new business requirements , improve performance etc.
- (Corrective) Must be changed because of errors found in the specification, design, or implementation

(Note: These are three major reasons for software maintenance, along with Preventive maintenance)

Characteristics of WebApps (1/2)

Network intensiveness. A WebApp resides on a network and must serve the needs of a diverse community of clients.

Concurrency. A large number of users may access the WebApp at one time.

Unpredictable load. The number of users of the WebApp may vary by orders of magnitude from day to day.

Performance. If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.

Availability. Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a “24/7/365” basis.

Data driven. The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.

Characteristics of WebApps (2/2)

Content sensitive. The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp

Continuous evolution. Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously

Immediacy. Although *immediacy—the compelling need to get* software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks

Security. Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application

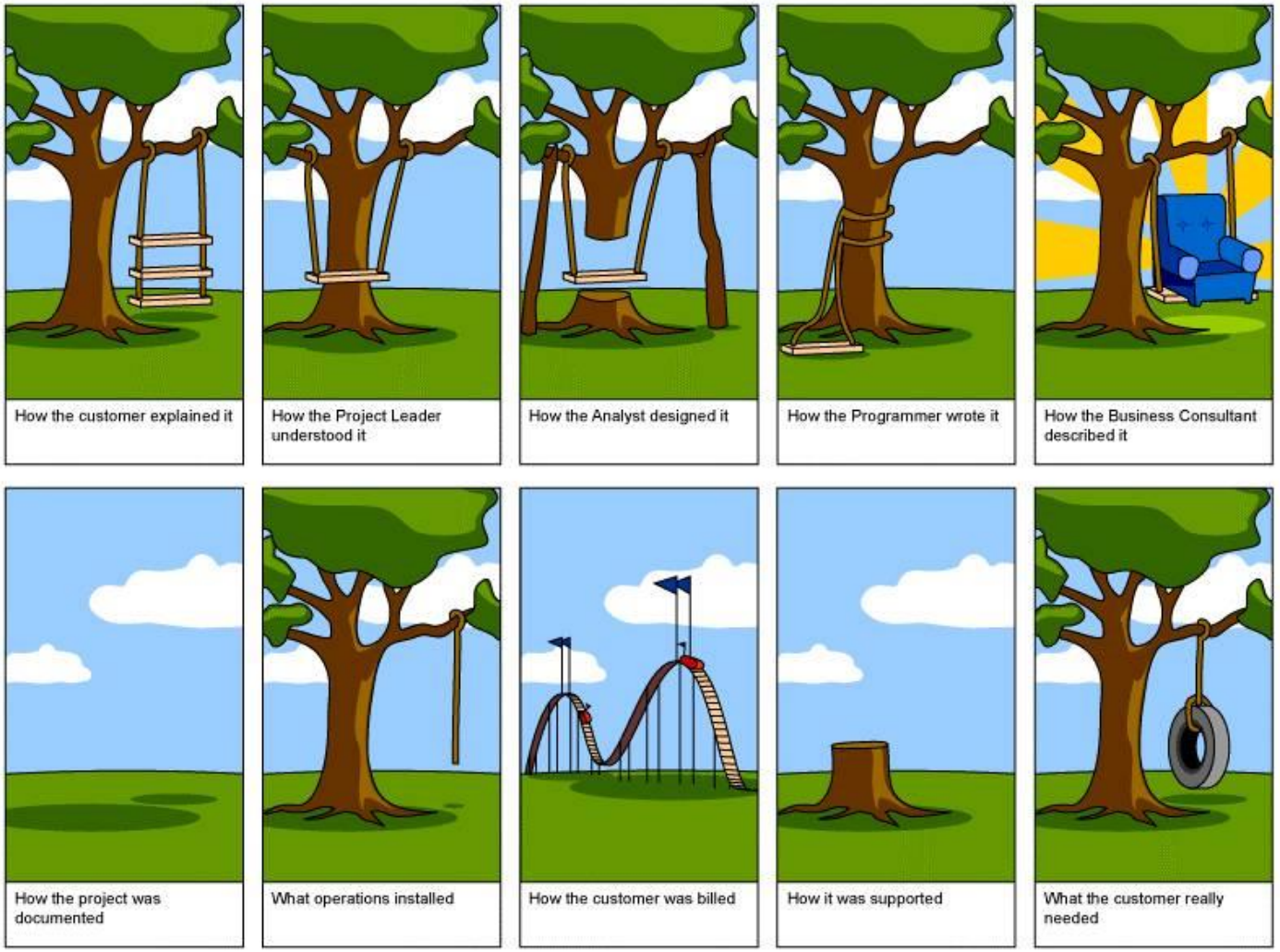
Aesthetics. An undeniable part of the appeal of a WebApp is its look and feel

Questions About Software Haven't Changed Over the Decades

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

Extreme Views on Software Engineers

(No shortage of jokes on software engineers)



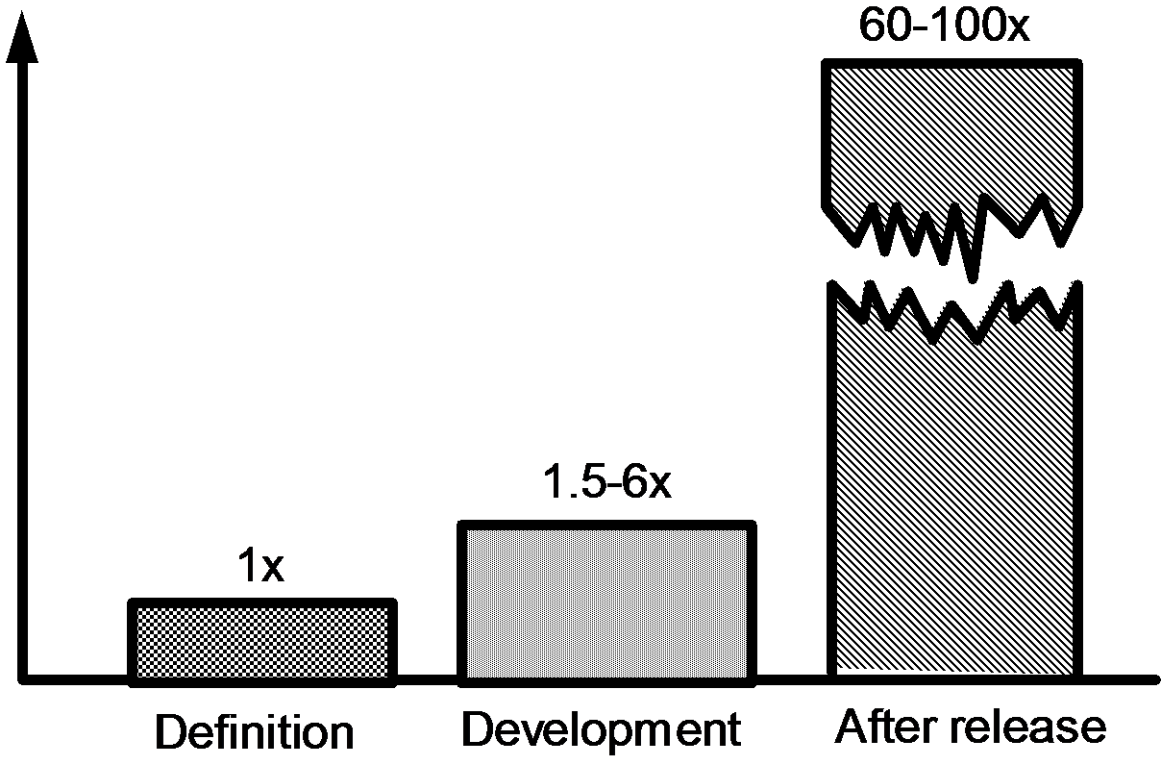
Software Myths - Management

- "We already have a book that is full of standards and procedures for building software. Won't that provide my people with everything they need to know?"
  - Not used, not up to date, not complete, not focused on quality, time, and money
- "If we get behind, we can add more programmers and catch up"
  - Adding people to a late software project makes it later
  - Training time, increased communication lines
- "If I decide to outsource the software project to a third party, I can just relax and let that firm build it"
  - Software projects need to be controlled and managed

Software Myths - Customer

- "A general statement of objectives is sufficient to begin writing programs – we can fill in the details later"
  - Ambiguous statement of objectives spells disaster
- "Project requirements continually change, but change can be easily accommodated because software is flexible"
  - Impact of change depends on where and when it occurs in the software life cycle (requirements analysis, design, code, test)

The Cost of Change



Software Myths - Practitioner

- "Once we write the program and get it to work, our job is done"
  - 60% to 80% of all effort expended on software occurs after it is delivered
- "Until I get the program running, I have no way of assessing its quality
  - Formal technical reviews of requirements analysis documents, design documents, and source code (more effective than actual testing)
- "The only deliverable work product for a successful project is the working program"
  - Software, documentation, test drivers, test results
- "Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down"
  - Creates quality, not documents; quality reduces rework and provides software on time and within the budget

Programming vs. Software Engineering

- Programmer productivity (say LOC per month) drops almost 10 times when a programmer moves from university to industry.
- Students write programs to demonstrate that it works
- In industry software engineer needs to ensure that business is never interrupted.
- Students may spend 5% time for testing, but testing effort in industry may be 50%

Software Development – Art or Science

- Experts have diversity of views
- Some believe it combines art and science
- Capers Jones thinks
- Steve McConnell has said that it is not a proper science, but it should be

> "During software design, I'm an architect. While I'm designing the user interface, I'm an artist. During construction, I'm a craftsman. And during unit testing, I'm one mean son of a bitch!"
>
> …the proper question is not "What *is* software development?" but rather "What *should* software development be?" In my opinion, the answer to that question is clear: Software development should be *engineering*. Is it? No. Should it be? Unquestionably, yes. - www.stevemcconnell.com

Engineering vs. Science

- *"A scientist builds in order to learn; an engineer learns in order to build."*

  — Fred Brooks

- Scientists learn what is true, how to test hypotheses, and how to extend knowledge in their field. Engineers learn what is true, what is useful, and how to apply well-understood knowledge to solve practical problems.

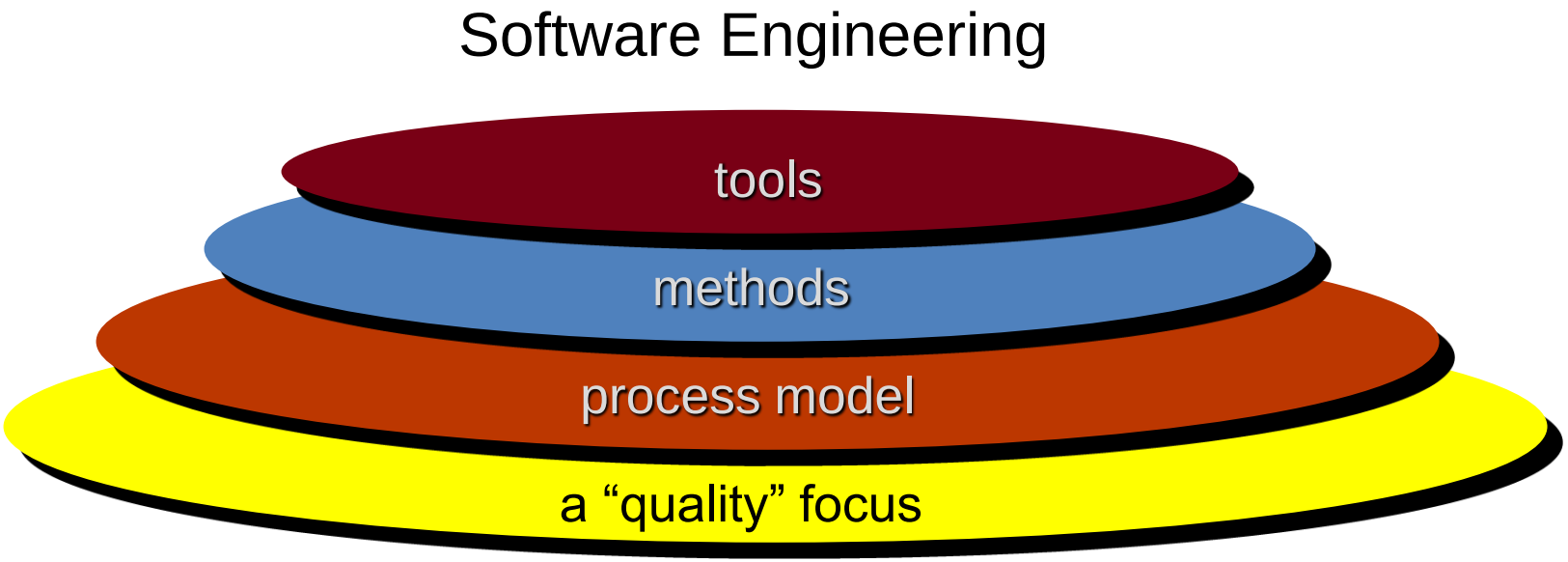
  – ***"Professional Software Development:*** *Shorter Schedules, Better Projects, Superior Products, Enhanced Careers" Steve McConnell* Addison-Wesley

Software Engineering - Definitions

- (1969 – Fritz Bauer) Software engineering is the establishment and use of *sound engineering principles* in order to obtain *economically* software that is *reliable* and works *efficiently* on *real machines*
  - Some may want to have technical aspects, customer satisfaction, timeliness, measurements, process included in the definition

- (IEEE) The application of a *systematic, disciplined, quantifiable* approach to the *development, operation, and maintenance* of software; that is, the application of engineering to software
  - Some may consider *adaptability and agility* more important than *systematic, disciplined, quantifiable approach*

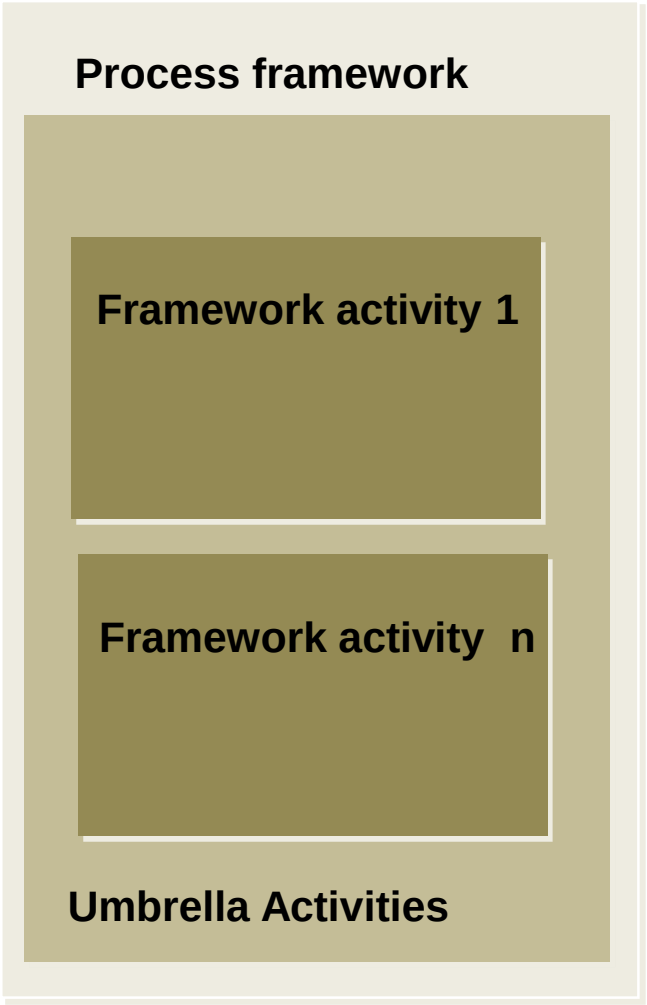
A Layered Technology

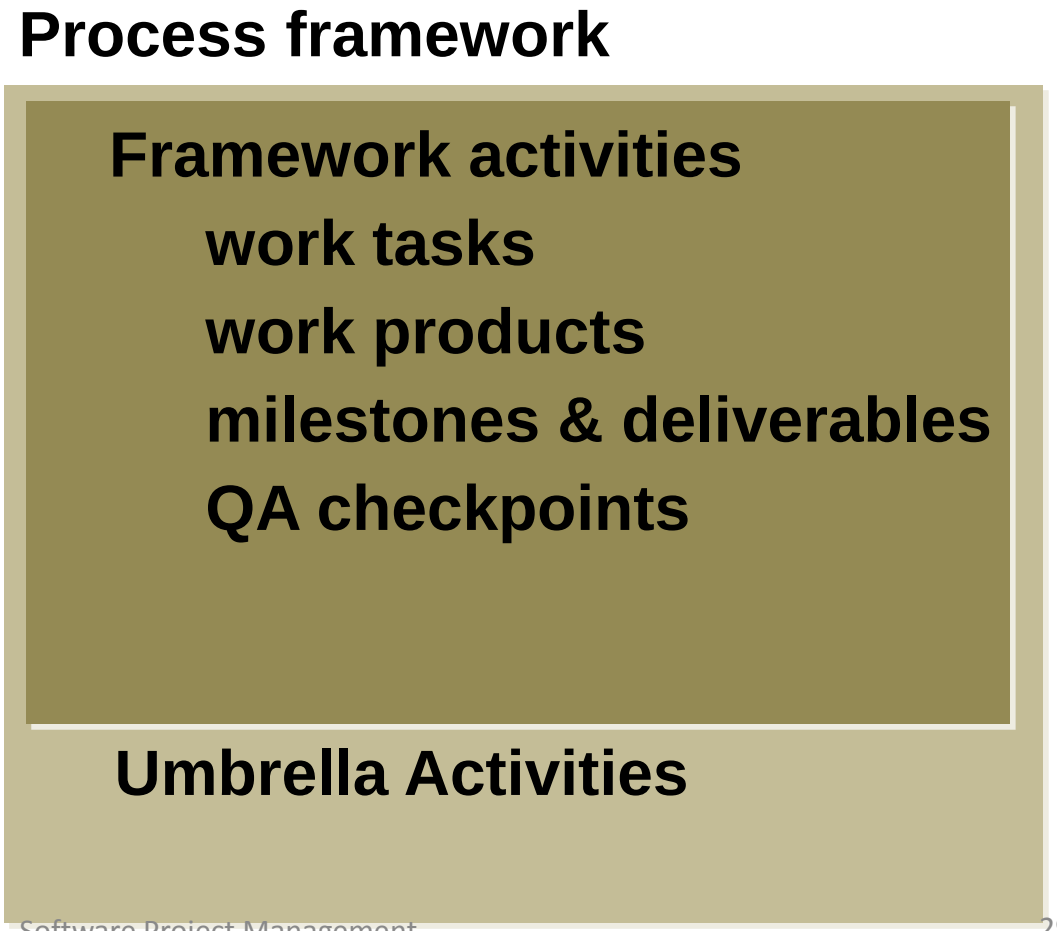


Process, Methods, and Tools

- Process
  - Provides the glue that holds the layers together; enables rational and timely development; provides a framework for effective delivery of technology; forms the basis for management; provides the context for technical methods, work products, milestones, quality measures, and change management
- Methods
  - Provide the technical "how to" for building software; rely on a set of basic principles; encompass a broad array of tasks; include modeling activities
- Tools
  - Provide automated or semi-automated support for the process and methods (i.e., CASE tools)

A Software Process Framework





A Software Process Framework

Process framework

Framework activities
- work tasks
- work products
- milestones & deliverables
- QA checkpoints

Umbrella Activities

Process framework

Modeling activity

Software Engineering action: **Analysis**

work tasks: requirements gathering, elaboration, negotiation, specification, validation

work products: analysis model and/or requirements specification

milestones & deliverables

QA checkpoints

Software Engineering action: **Design**

work tasks: data design, architectural, interface design, component design

work products: design model and/or design specification

Umbrella Activities

Framework Activities

- Communication #
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning *
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modeling (Analyze, Design)
  - Encompasses the creation of models to better understand the requirements and the design
- Construction (Code, Test)
  - Combines code generation and testing to uncover errors
- Deployment
  - Involves delivery of software to the customer for evaluation and feedback

Umbrella Activities

- Software project tracking and control *
  - Assess progress against the plan
- Software quality assurance #
  - Activities required to ensure quality
- Software configuration management #
  - Manage effects of change
- Technical Reviews
  - Uncover errors before going to next activity
- Formal technical reviews
  - Assess work products to uncover errors

Umbrella Activities (contd)

- Risk management *
  - Assess risks that may affect quality
- Measurement – process, project, product #
- Reusability management (component reuse) #
- Work product preparation and production
  - Models, documents, logs, forms, lists…

etc.

How Process Models Differ?

While all Process Models take same framework and umbrella activities, they differ with regard to

- Overall flow of activities, actions, and tasks and the interdependencies among them
- Degree to which actions and tasks are defined within each framework activity
- Degree to which work products are identified and required
- Manner in which quality assurance activities are applied
- Manner in which project tracking and control activities are applied
- Overall degree of detail and rigor with which the process is described
- Degree to which customer and other stakeholders are involved in the project
- Level of autonomy given to the software team
- Degree to which team organization and roles are prescribed

A Simple Practical Definition

A Process defines who is doing what, when, and how to reach a certain goal

-Ivar Jacobson, Grady Booch, and James Rumbaugh

Prescriptive and agile processes

- Prescriptive processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes may include elements of both plan-driven and agile approaches.

- ***There are NO right or wrong software processes.***

Prescriptive Process Model

- Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software

- The activities may be linear, incremental, or evolutionary

Another View on Software Process

- Because software, like all capital, is embodied knowledge, and because that knowledge is initially dispersed, tacit, latent, and incomplete in large measure, software development is a social learning process. The process is a dialogue in which the knowledge that must become software is brought together and embodied in the software.

-Howard Baetjer (economist)

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

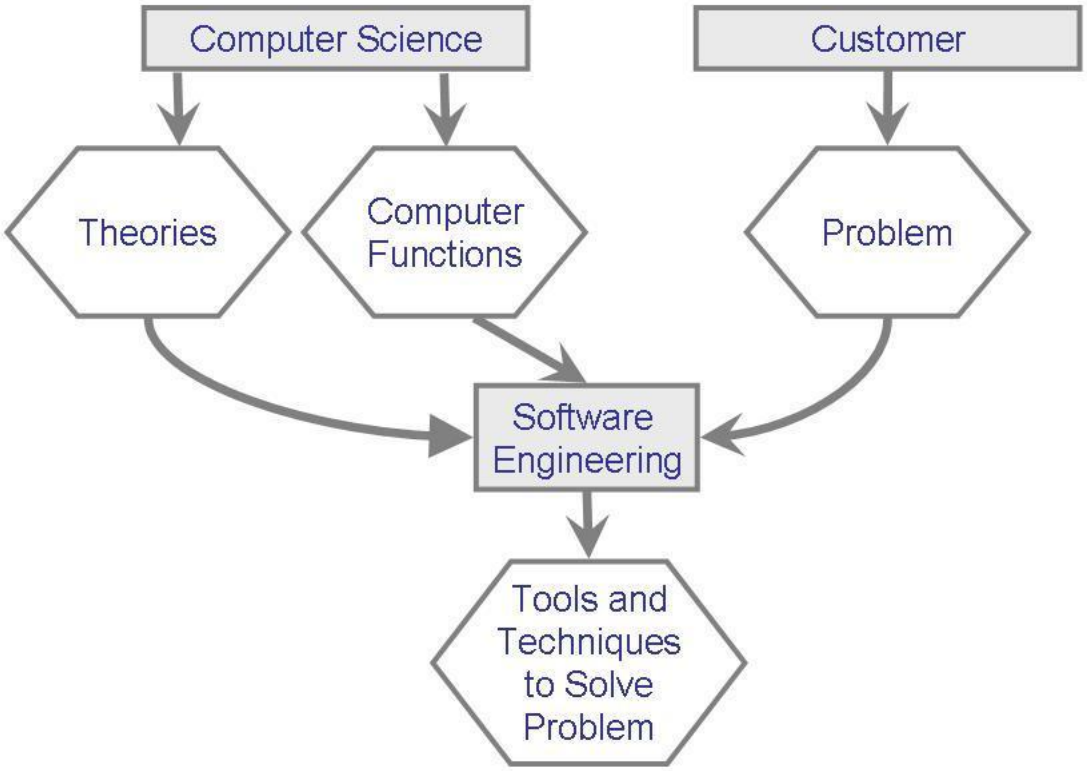
Process Patterns

- A proven solution to a problem
  - Describes process-related problem
  - Environment in which it was encountered
- A template to describe solution
  - Ambler proposed a template for describing a process pattern
- Can be defined at any level of abstraction
  - Complete process model (e.g. Prototyping)
  - Frame work activity (e.g. Planning)
  - An action within framework activity (e.g. Estimation)

Process Patterns – Ambler's Template

- Pattern Name
- Forces
  - Describes environment of the problem
- Type
  - Phase (e.g. Prototyping), Stage (e.g. Planning), or Task (e.g. Estimation)
- Initial Context
- Problem
- Solution
- Resulting Context
- Related Patterns
- Known uses & examples

Software Engineering Context



Software Engineering

(as per author Sommerville)

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

  - Engineering discipline
    - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
  - All aspects of software production
    - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

FAQs on Software

by author Sommerville

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

FAQs on Software Engineering

by author Sommerville

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Attributes of good software

by author Sommerville

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

The software process (as per Somerville)

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

The Essence of Problem Solving

(as Per George Polya – A Hungarian Mathematician)

1) Understand the problem
   - What are the unknowns?
   - Can the problem be compartmentalized?
   - Can the problem be represented graphically?
2) Plan a solution
   - Have you seen similar problems like this before?
   - Has a similar problem been solved and is the solution reusable?
   - Can subproblems be defined and solved?
3) Carry out the plan
   - Is the solution as per plan?
   - Is each component of the solution correct?
4) Examine the results for accuracy
   - Is it possible to test each component of the solution?

The Essence of Problem Solving

(in Software Engineering)

1) **Understand the problem (communication and analysis)**
   - Who has a stake in the solution to the problem?
   - What are the unknowns (data, function, behavior)?
   - Can the problem be compartmentalized?
   - Can the problem be represented graphically?
2) **Plan a solution (planning, modeling and software design)**
   - Have you seen similar problems like this before?
   - Has a similar problem been solved and is the solution reusable?
   - Can subproblems be defined and solved?
3) **Carry out the plan (construction; code generation)**
   - Is the solution as per plan? Is the source code traceable to the design?
   - Is each component of the solution correct? Has the design and code been reviewed?
4) **Examine the results for accuracy (testing and quality assurance)**
   - Is it possible to test each component of the solution?
   - Does the solution produce results that conform to the data, function, and behavior that are required?

Seven Core Principles for Software Engineering

(David Hooker – 1996)

1) Remember the reason that the software exists
   - The software should provide value to its users and satisfy the requirements
2) Keep it simple, stupid (KISS)
   - All design and implementation should be as simple as possible
3) Maintain the vision of the project
   - A clear vision is essential to the project's success
4) Others will consume what you produce
   - Always specify, design, and implement knowing that someone else will later have to understand and modify what you did
5) Be open to the future
   - Never design yourself into a corner; build software that can be easily changed and adapted
6) Plan ahead for software reuse
   - Reuse of software reduces the long-term cost and increases the value of the program and the reusable components
7) Think, then act
   - Placing clear, complete thought before action will almost always produce better results

What Lies Ahead?

Some trends that will exacerbate Hardware-software-human factors integration problems are

- ***Complex, multi-owner systems of systems.*** Current collections of incompatible, separately-developed systems will cause numerous challenges.
- ***Emergent requirements*****.** Demands for most appropriate user interfaces and collaboration modes for a complex human-intensive system.
- ***Rapid change*****.** Specifying current-point-in-time snapshot requirements on a cost-competitive contract generally leads to a big design up front, and a point-solution architecture that is hard to adapt to new developments.
- ***Reused components.*** Reuse-based development has major bottom-up development implications, and is incompatible with pure top-down requirements-first approaches.
- ***High assurance of qualities.*** Future systems will need higher assurance levels of such qualities as safety, security, reliability/availability/maintainability, performance, adaptability, interoperability, usability, and scalability.

Barry Boehm and Jo Ann Lane, University of Southern California, 2007

Software Engineering Trends

- Managing complexity of increasingly larger software projects
- Open-world software encompasses ambient intelligence, context aware applications, and pervasive computing
- The realities of emergent requirements on large projects force organizations to adopt incremental process models
- Getting the right talent mix on software dream teams
  - Brain – chief architect able map stakeholder demands into a technology framework that is extensible and implementable
  - Data Guru – data base/structures guru
  - Blocker – manager who allows team to work free from interference
  - Hacker – consummate programmer
  - Gatherer – gathers system requirements and expresses them with clarity
- Software building blocks that allow unique solutions to be assembled by reusing existing artifacts
- Changing perception of value from business (cost and profitability) toward customer (delivery speed, functionality, product quality)
- Open source software that harnesses the power of distributed review and transparency of process (an end to predatory vendor lock-in)

  – Roger Pressman

Another perspective of SE:

Do we stand on quicksand or the shoulders of giants?

- Have you ever found that a new method or practice is just the re-branding and regurgitation of old?
- Do you find every new idea about software development seems to be at the expense and in aggressive competition with everything that has gone before?
- Does it seem to you that following that latest software development trend has become more important than producing great software?
- Have you noticed how in their hurry to forge ahead people seem to throw away the good with the bad? It is as though they have no solid knowledge to stand upon.
- Many teams carelessly discard expensive process and tool investments, almost before they have even tried them.
- Every time someone changes their job they have to learn a new approach before they can get on with the real task at hand. People cannot learn from experience as they are forever starting over.

Ivar Jacobson, Ian Spencer "Why we need a theory for Software Engineering", 2009

Standish Group's Chaos Report

About Standish Group

We focus on failure to help you succeed. The Standish Group is based in Boston, Massachusetts and is the Information Technology leader in project and value performance. We are a group of highly dedicated professionals with years of practical experience in assessing risk, cost, return and value for Information Technology (IT) Investments.

Every two years, Standish Group publishes Chaos Report that gives statistics of success among IT projects based on their proprietary research on IT projects during previous two years.

	1994	1996	1998	2000	2002	2004	2006	2008
Successful	16%	27%	26%	28%	34%	29%	35%	32%
Challenged	53%	33%	46%	49%	51%	53%	46%	44%
Failed	31%	40%	28%	23%	15%	18%	19%	24%

The Chaos Report is widely quoted. However, some experts/practitioners advise to take this report with a pinch of salt.

Summary

- Software is a logical entity
- Software is engineered; not manufactured
- Software deteriorates in different ways than hardware
- Software Engineering could be looked upon as layers of elements
- Software Engineering requires framework activities and umbrella activities
- Patterns help gain from established solutions
- There are many myths associated with software engineering
- Software Engineering activities can be mapped to Polya's scientific approach to solve a problem
- David Hooker offers seven valuable software engineering principles

Thank You

Any Questions?