Software Best Practices

Sridhar Pandurangiah
Director - Engineering
sridhar@sastratechnologies.in

Agenda

- Highlight some of the best practices of software development
- Establish a context for sound management and engineering practices

Introduction...

- Software intensive systems world-wide are expanding in size, complexity, distribution and importance
- Extreme pressure on what we as software engineers know how to develop

Introduction

- Clients continue to demand increased productivity and improved quality with faster development and deployment
- Net result is that the whole process of constructing software is plagued by problems and is getting difficult by the day

Symptoms ...

- Inaccurate understanding of end user needs
- Inability to deal with changing requirements
- Modules that don't fit together
- Software that is hard to maintain or extend
- Late discovery of serious project flaws

Symptoms

- Poor software quality
- Unacceptable software problems
- Team members in each other's way, making it impossible to reconstruct who changed what, when, where and why
- An untrustworthy build-and-release process
- Reactive project management

However

Treating these symptoms does not treat the disease

Root Causes ...

- Ad hoc requirements management
- Ambiguous and imprecise communication
- Brittle architectures
- Overwhelming complexity
- Undetected inconsistencies in requirements, designs and implementations
- Insufficient testing

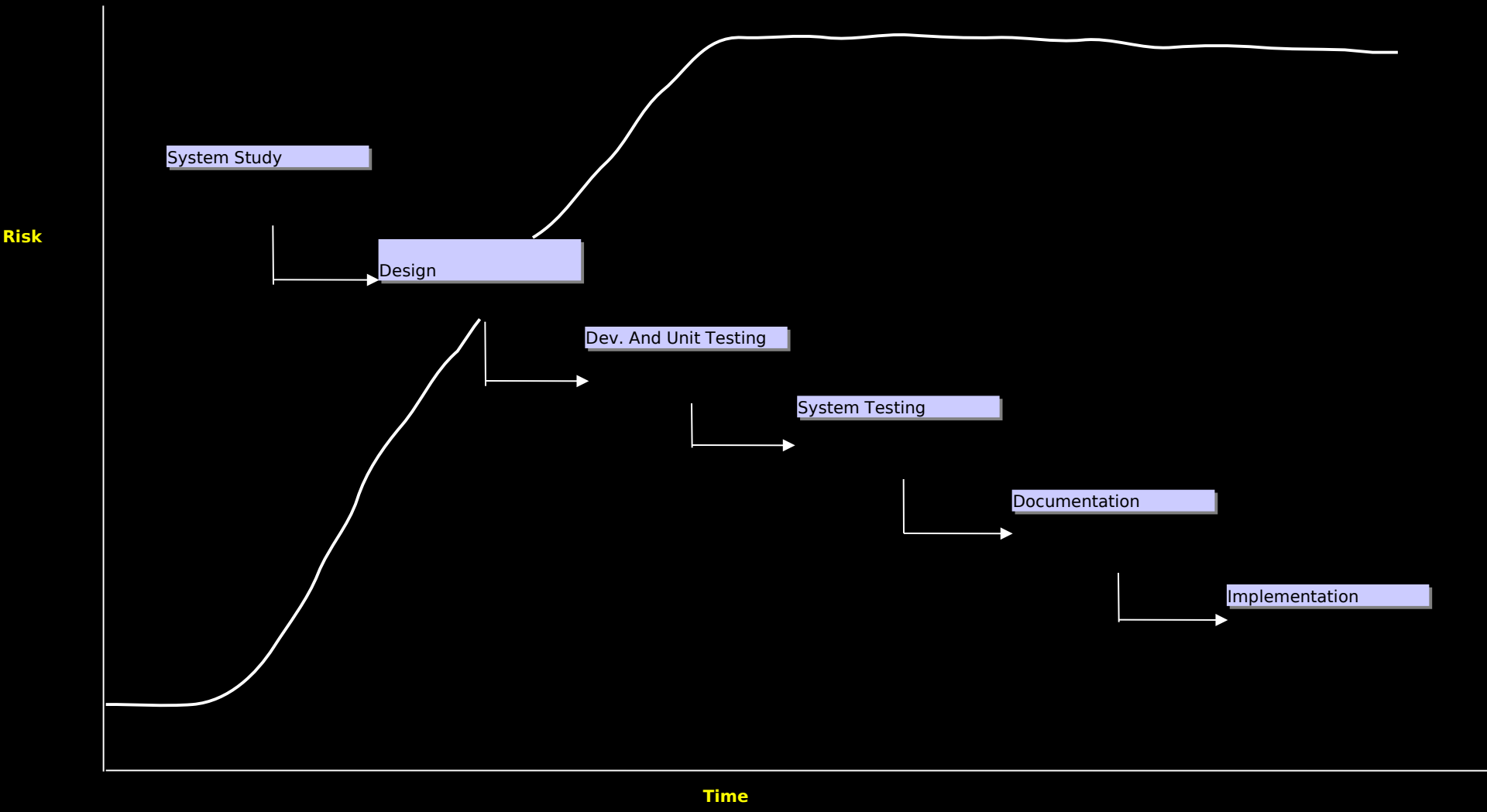
Root Causes

- Subjective Project status assessment
- Failure to attack risk
- Uncontrolled change propagation
- Insufficient automation

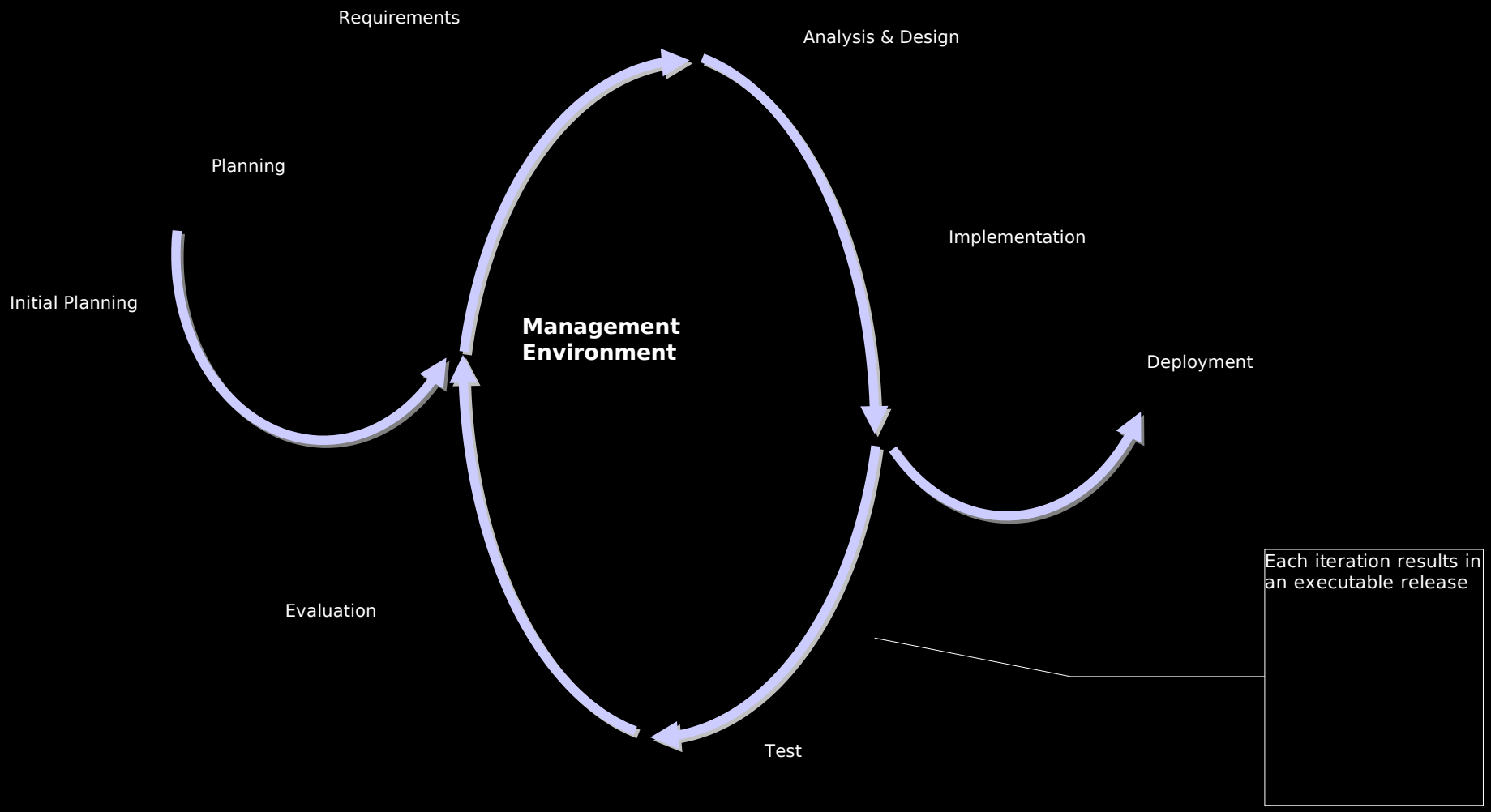
Best Practices

- Develop Software Iteratively
- Manage requirements
- Use component based architectures
- Visually model software
- Verify software quality
- Control changes to software

Develop Iteratively...



Develop Iteratively



Solutions of Iterative Development...

- Serious misunderstandings are made evident early in the life cycle, when it is possible to react to them.
- This approach enables and encourages user feedback so as to elicit the system's real requirements.

Solutions of Iterative development...

- The development team is forced to focus on those issues that are most critical to the project and are shielded from those issues that distract them from the project's real risks.
- Continuous, iterative testing enables an objective assessment of the project's status.

Solutions of Iterative Development

- Inconsistencies among requirements, designs and implementations are detected early.
- The workload of the team, especially the testing team, is spread out more evenly throughout the lifecycle.
- The team can leverage lessons learned and therefore can continuously improve the process.

Solutions of Iterative Development

- **Stakeholders in the project can be given concrete evidence of the project's status throughout the lifecycle.**

Manage Requirements

- Its impossible to completely state the system's requirements before the start of development especially the systems economic and technical goals
- ELICIT / ORGANISE / DOCUMENT

Solutions of Requirements Management...

- A disciplined approach is built into requirements management
- Communications are based on defined requirements
- Requirements can be prioritised, filtered and traced
- An objective assessment of functionality and performance is possible.

Solutions of Requirements Management

- Inconsistencies are more easily detected.
- With suitable tool support, it is possible to provide a repository for a system's requirements, attributes and traces with automatic links to external documents.

Component Based Architectures...

- Different stakeholders in a project have a different agenda and look at the system in a different way at different times over the project's life
- system's architecture is the most important deliverable that can be used to manage these different viewpoints

Component Based Architectures...

- The Systems architecture encompasses
  - The organisation of a software system.
  - The selection of the structural elements and their interfaces by which the system is composed.
  - Their behaviour, as specified by the collaborations among those elements

Component Based Architecture...

- The composition of these structural and behavioural elements into progressively larger subsystems.
- The architectural style that guides this organisation: these elements and their interfaces, their collaborations, and their composition.

Component Based Architecture

- Component based architectures are important because
  - enable economically significant degrees of reuse
  - offer a clear division of work among teams of developers
  - isolate hardware and software dependencies that may be subject to change and
  - improve maintainability

Solutions of Component Based Architecture ...

- Components facilitate resilient architectures
- Modularity enables a clear separation of concerns among elements of a system that are subject to change.
- Reuse can be facilitated using standardised frameworks such as COM+, CORBA and EJB.

Solutions of Component Based Architecture

- Components provide a natural basis for configuration management.
- Visual modelling tools provide automation for component based development.

Visually Model Software ...

- A picture is worth a hundred words if not thousand!
- IDEF10, IDEF1, IDEF1X, IDEF3, IDEF4, IDEF5 or the UML
- Unambiguously communicate decisions to different team members
- maintain consistency between a system's artefacts
- helps improve the team's ability to manage software complexity

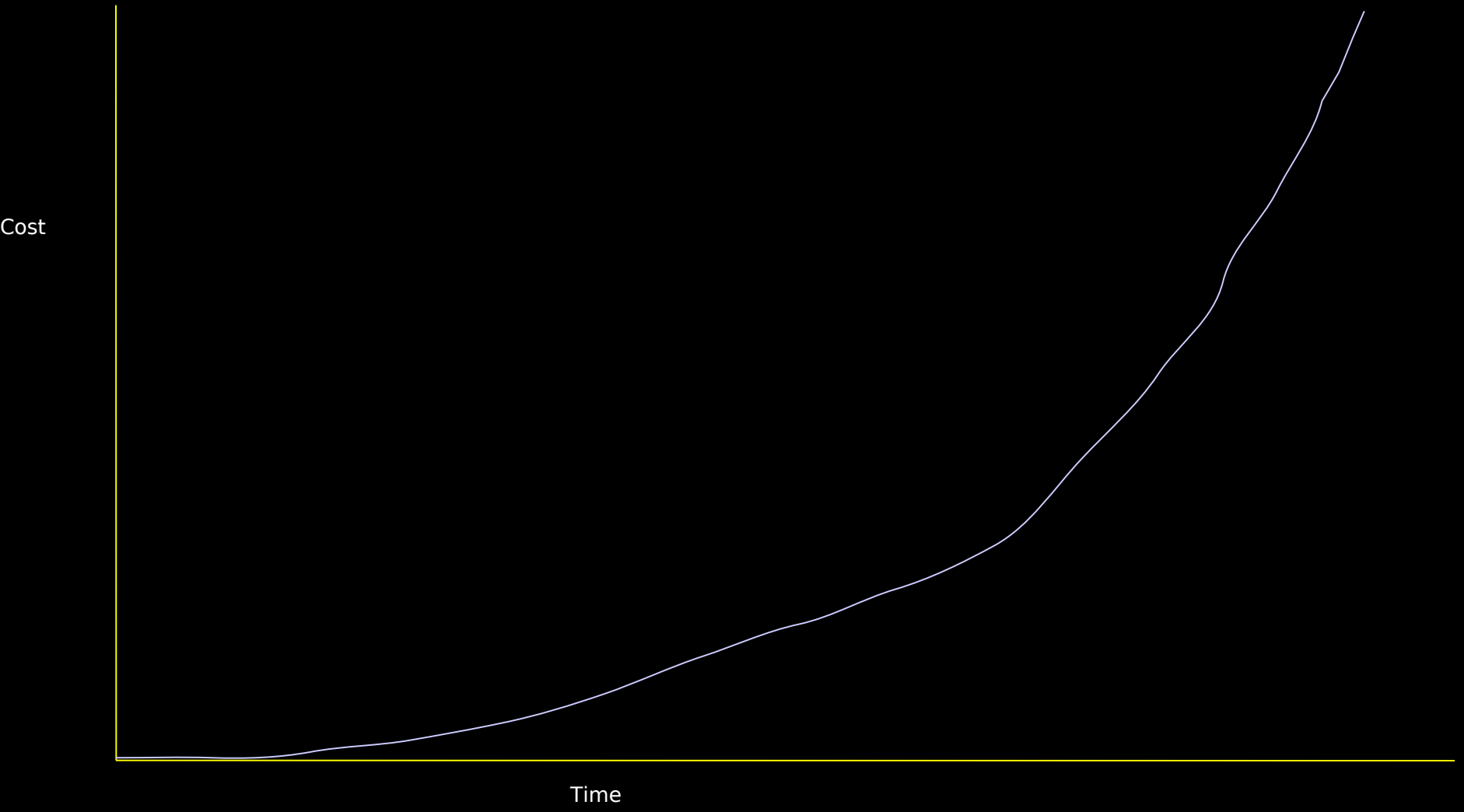
Solutions of Visual Modelling...

- Use cases, Scenarios, Entities, Relationships and attributes unambiguously specify behaviour.
- Models unambiguously capture software design
- Nonmodular and inflexible architectures are exposed.

Solutions of Visual Modelling

- Detail can be hidden when necessary
- Unambiguous designs reveal their inconsistencies more readily.
- Application quality starts with good design.
- Visual modelling tools provide support for IDEF1X and UML modelling.

Verify Software Quality



Solutions of Verifying Quality...

- **Project status assessment is made objective, and not subjective, because test results, and not paper documents are evaluated.**
- **This objective assessment exposes inconsistencies in requirements, designs and implementations.**

Solutions of Verifying Quality

- Testing and verification are focussed on areas of highest risk, thereby increasing their quality and effectiveness.
- Defects are identified earlier, radically reducing the cost of fixing them.
- Automated testing tools provide testing for functionality, reliability and performance.

Control Changes to Software

- Multiple developers
- Different feature teams
- Different sites
- Multiple iterations, releases, products and platforms

Solutions of Controlling changes to software...

- The workflow of requirements change is defined and repeatable
- Change requests facilitate clear communications
- Isolated workspaces reduce interference among team members working in parallel.

Solutions of Control Changes to Software ...

- Change rate statistics provide good metrics for objectively assessing project status.
- Workspaces contain all artefacts, facilitating consistency.
- Change propagation is assessable and controlled.
- Changes can be maintained in a robust, customisable system.

Conclusion...

- **A Software Process**
  - **Provide guidance as to the order of a team's activities.**
  - **Specify which artefacts should be developed and when they should be developed.**
  - **Direct the tasks of individual developers and the team as a whole.**
  - **Offer criteria for monitoring and measuring the project's products and activities.**

Conclusion ...

- **Without a sound process**
  - **Development is adhoc & chaotic**
  - **Depends on heroics of few individuals**
  - **Depends on pressure from Project Manager**

Conclusion …

- With a well defined process
  - Complex systems can be developed in a repeatable and predictable way
  - This is sustainable and improved with each new project
  - Such well defined processes should incorporate the best practices

References

1. ***The Rational Unified Process, an Introduction***, Philippe Kruchten, Addison-Wesley Object Technology Series
2. ***Leaving Kansas***, Grady Booch, IEEE Software 15(1) Jan-Feb 1998
3. ***Patterns of Software Systems Failures and Success***, Caper Jones, London:International Thompson Computer Press, 1996

References

4. ***Death March: Managing "Mission Impossible" Projects***, **Edward Yourdon, Upper Saddle River, NJ: Prentice Hall, 1997**
5. ***Principles of of Software Engineering Management***, **Tom Gilb, Harlow, UK: Addison-Wesley, 1998**

References

6. ***"A Spiral Model of Software Development and Enhancement"*, Barry W. Boehm, IEEE Computer, 1998**
7. ***Object Solutions – Managing the Object Oriented Project*, Grady Booch, Reading, MA: Addison-Wesley, 1995**

Thank You