Agile and Evolutionary Project Management Methods

An Introduction

Recap: Systems development lifecycle approaches

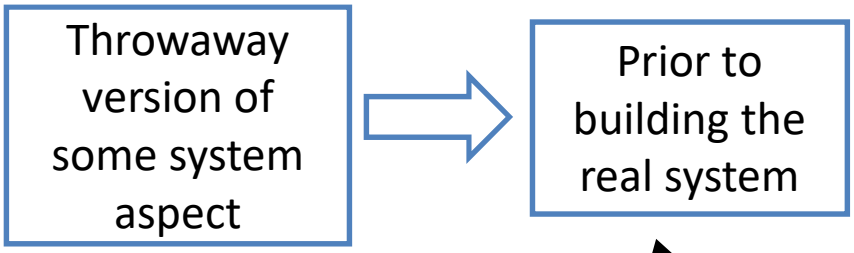
- Sequential

Linear – one pass / one delivery

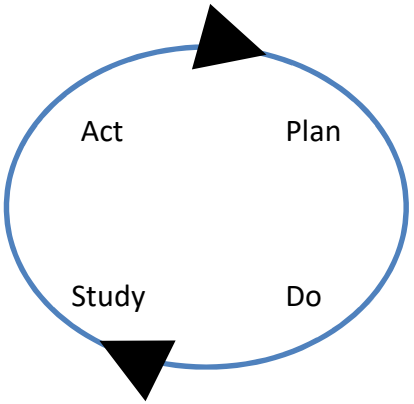
- Incremental

Build and deliver the system, increment by increment

- Prototyping



- Iterative / Evolutionary



Use feedback

Recap: Traditional ‘waterfall’ model

- Can be applied to all types of projects, but…
  - Fixed requirements
  - Limited client/business involvement at alter stages
  - Command and control style
  - Project plan – project manager
  - Structured, linear approach
  - Risk management and concerns on scope creep
  - Monitoring of progress through Gantt charts
- Plan Driven
  - fixed requirements and estimated time and resources

Recap: Traditional ‘waterfall’ model Challenges

- Lightweight application/heavyweight process
- Document intensive (perceived)
- Less flexible design
- Big bang approach to coding/integration
- Testing short-shifted
- One-shot presentation opportunity
- Lack of opportunity for process improvement

Recap: ‘Iterative’ / Evolutionary

- Involves increments/iterations
- Time-boxing (varying durations)
- Delivery to market or client
- Incorporates feedback (learning)
- Being prepared to retreat
- Not ‘freezing’ the requirements
- Decide what to do in the next increment/iteration using the feedback from the last increment/iteration and the current set of requirements

Iterative / Evolutionary Methods

- Evolutionary delivery (Evo)
- Boehm's Spiral Model
- DSDM (Dynamic Systems Development Method)
- RUP (Rational Unified Process)
- Scrum
- XP (Extreme Programming)
- Lean Software Development ('Lean')
- Others

Iterative / Evolutionary Methods

- Evolutionary delivery (Evo)
- Boehm's Spiral Model
- DSDM (Dynamic Systems Development Method)
- RUP (Rational Unified Process)
- Scrum
- XP (Extreme Programming)
- Lean Software Development ('Lean')
- Others

Agile Methods

Boehm's Spiral Model

The spiral model is similar to the incremental model, with more emphasis placed on **risk analysis.**

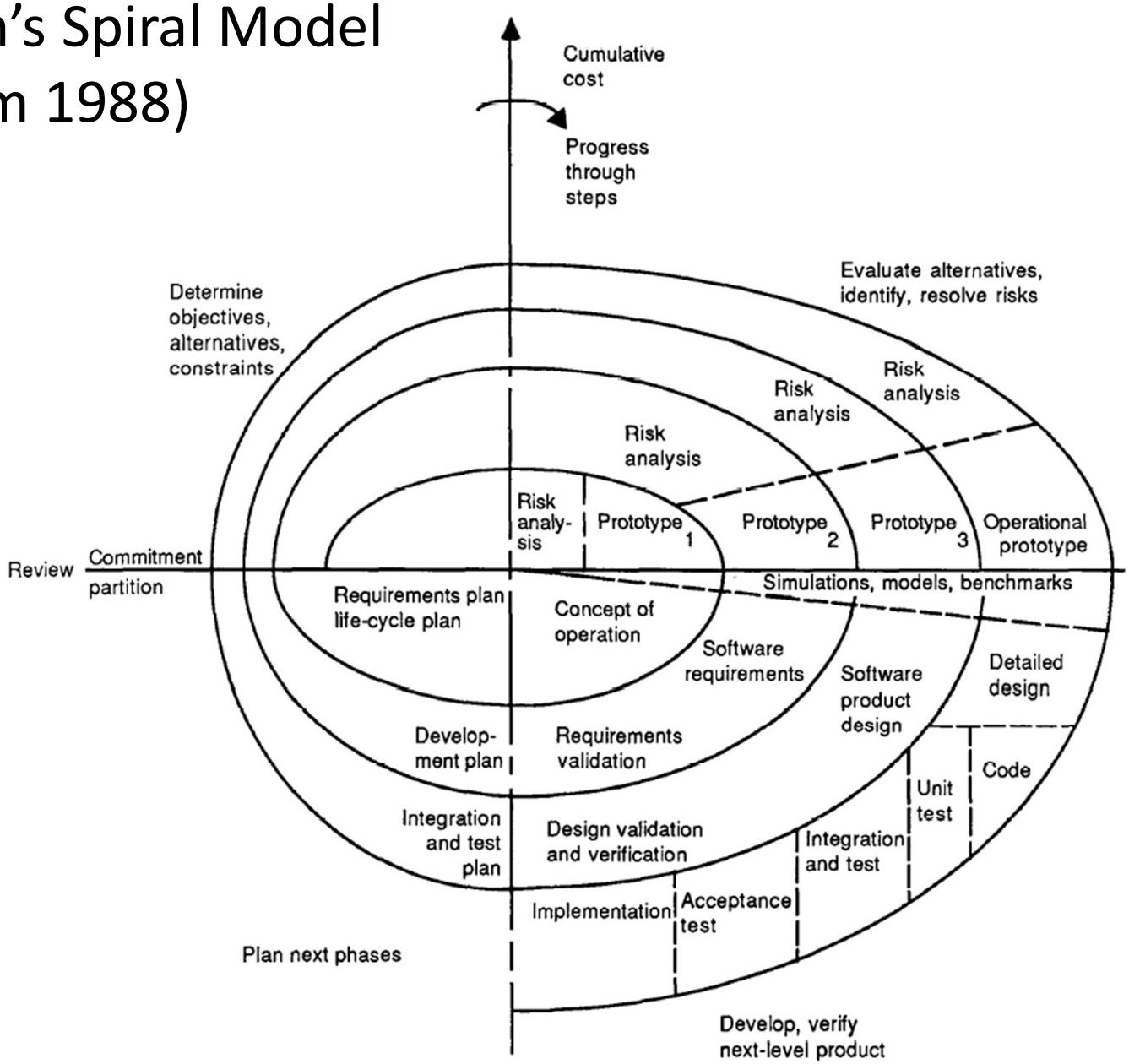
The spiral model has four phases.

A software project repeatedly passes through these phases in iterations.

Boehm's Spiral Model

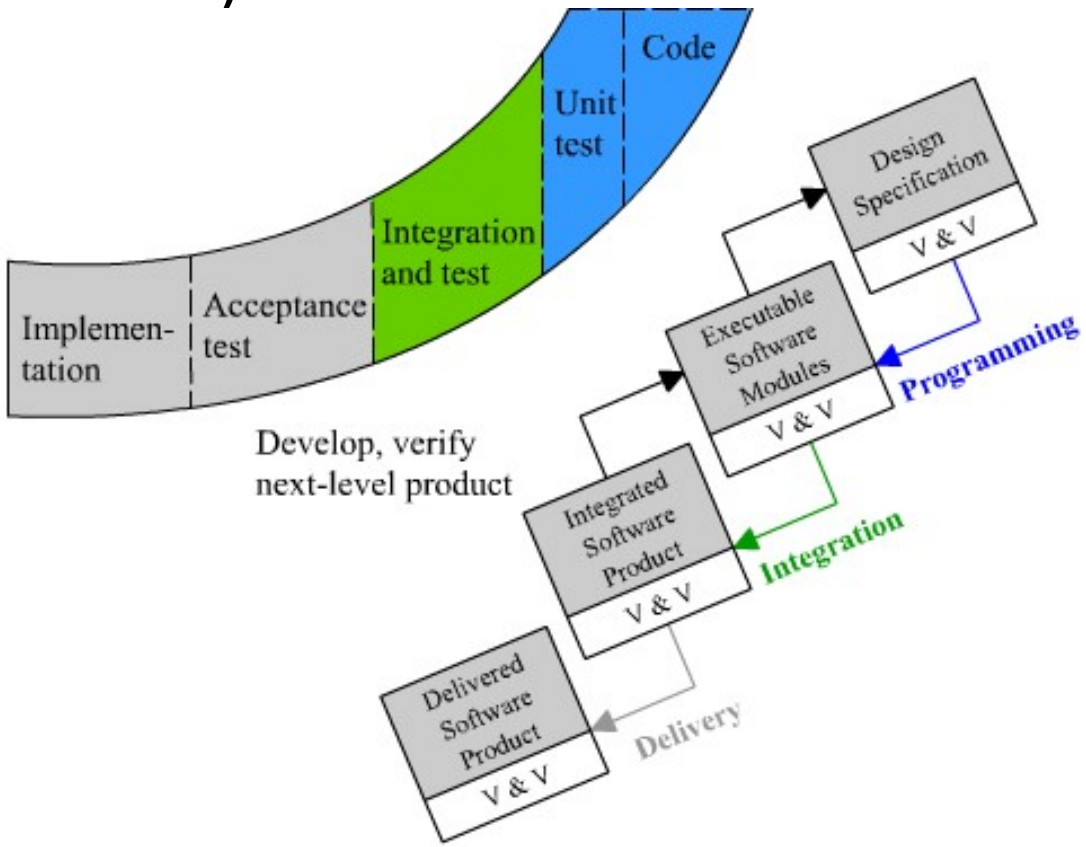
- "A new framework for guiding the software process" (Boehm 1988)
- The software project is broken down into many mini projects
- These projects address one (or more) major risks until all risks are dealt with
- A risk-driven rather than a document-driven or a code-driven approach
- A process model "provides guidance on the order (phases, increments, prototypes, validation tasks, etc.) in which a project should carry out its major tasks"
- Intended for large projects - Iterations: 6 months to 2 years
- Involves identifying alternative options
- Selects appropriate process model for an iteration based on the risks perceived: prototyping, evolutionary development, waterfall model
- Can become multiple spirals in parallel building different components for later integration
- Caveat: Not necessarily delivery to market or client for use each iteration

Boehm's Spiral Model (Boehm 1988)



width of spiral determines cumulative cost of project

Boehm's Spiral Model (Boehm 1988) - Contd



Maintenance is another spiral or phase in the life cycle of the software. Like the previous phase, the maintenance efforts undergo risk assessment to evaluate whether changes are feasible.

- The final phase of the Spiral Model is analogous to the Waterfall Model.
- At this point in the project, the software requirements should be well-understood through the development of several prototypes.
- The project should also have resolved the major risks involved with building the final version of the software.

Template	*Explanation*	*Example Phase*
Objectives	The goals of the software project	Significantly improve software quality
Constraints	Limitations which the project must meet	Within three years Without large-scale capital investment Without radical change to company standards
Alternatives	Possible ways to achieve the objectives	Reuse existing certified software Introduce formal specification and verification Invest in testing and validation tools
Risks	Potential risks for this phase	No cost effective quality improvement possible Quality improvements may increase costs excessively New methods might cause existing staff to leave
Risk Resolution	Strategies for reducing the risks	Literature survey, Pilot project, Survey of potential reusable components, Assessment of available tool support, Staff training and motivation seminars
Results	Results of applying risk resolution strategies	Experience of formal methods is limited - very hard to quantify improvements Limited tool support available for company standard development system Reusable components available but little reuse tool support
Plans	Development plans for the next phase	Explore reuse option in more detail Develop prototype reuse support tools Explore component certification scheme
Commitment	Resources needed to achieve the plans	Fund further 18-month study phase

Spiral Model Template

Boehm's Spiral Model

- In summary: get more details on some aspect with each iteration evolving towards a solution
  - Development starts on small scale
  - Risks are identified
  - Plan addressing these risks is developed
  - Alternatives evaluated
  - Deliverables identified, developed and verified before next iteration gets go-ahead

Boehm's Spiral Model

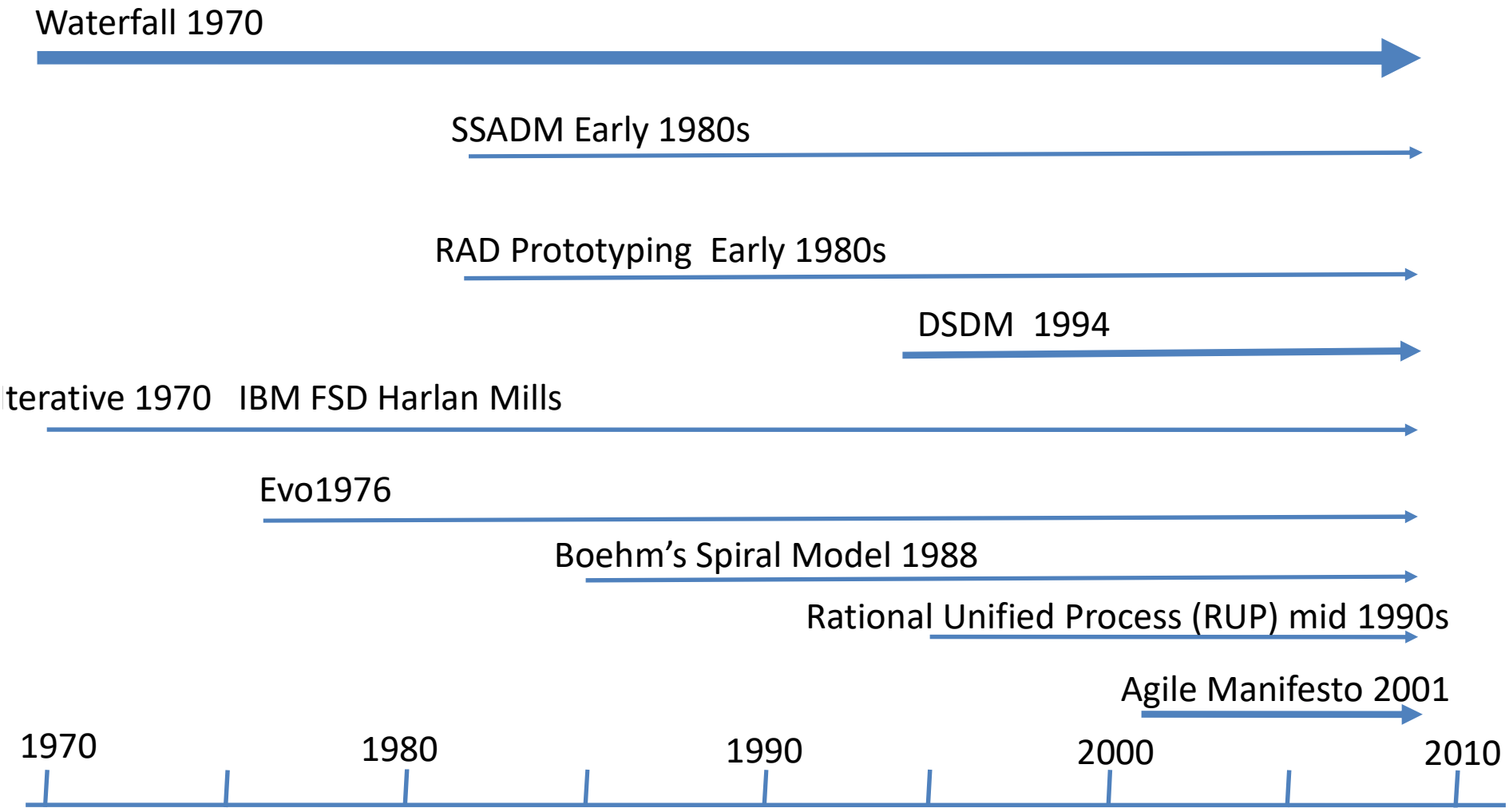
- Difficulties identified:
  - Matching up to a Contract
  - Risk-assessment Expertise
  - Further Elaboration of the Spiral Model Steps

- Concept of the usefulness of having a Risk Management Plan
  = understanding of why you are carrying out an iteration

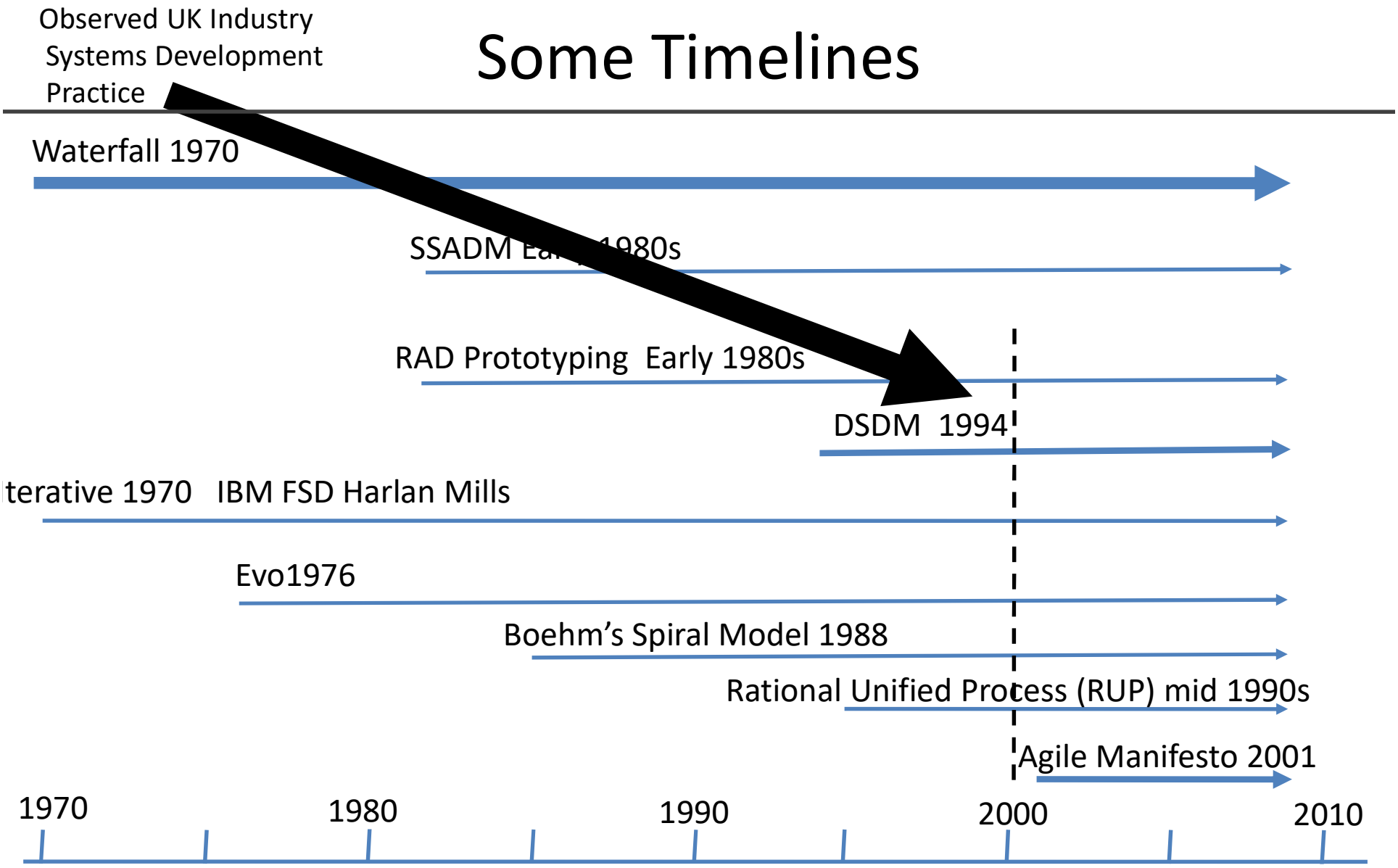
When to use?

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

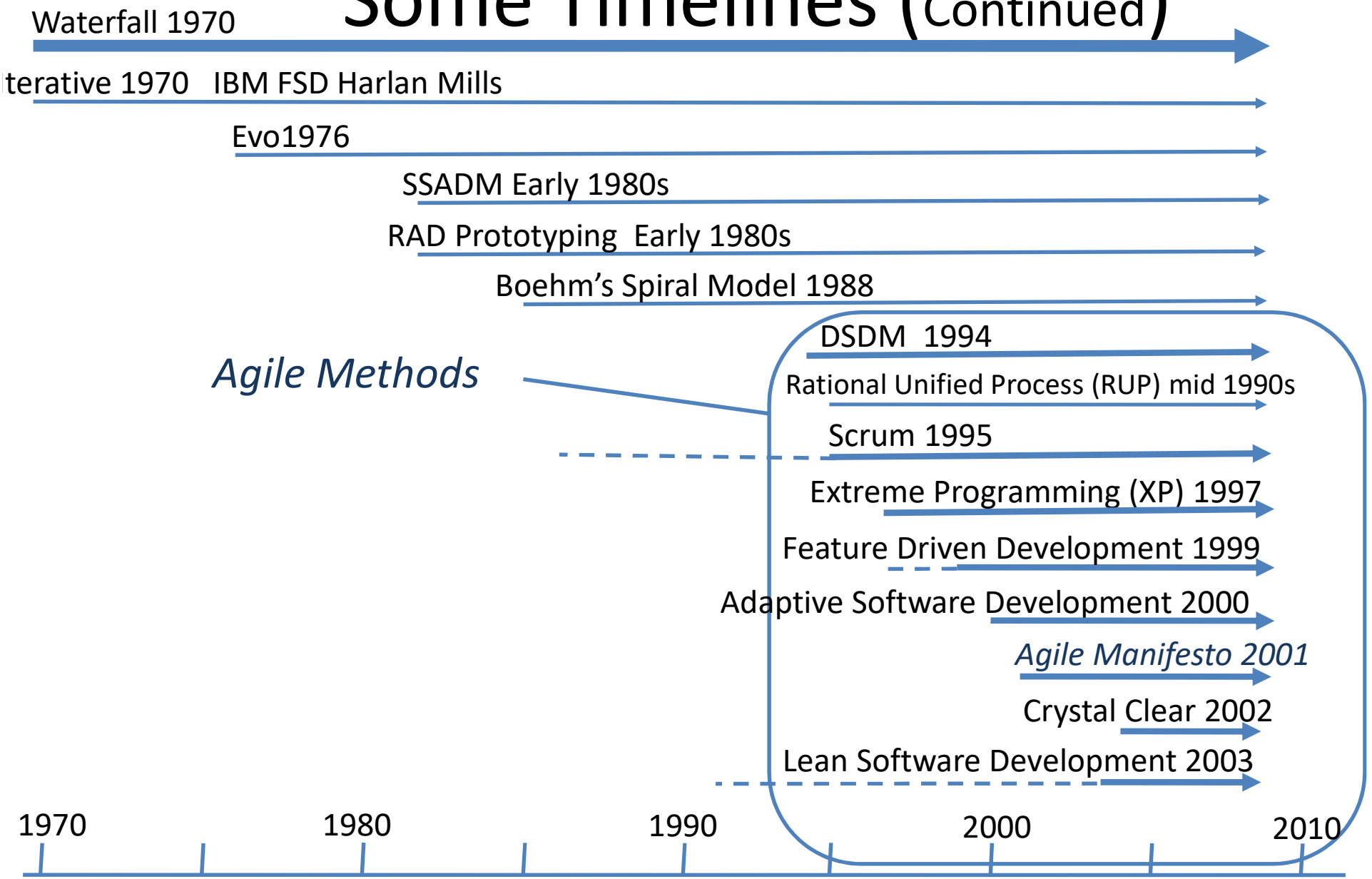
Some Timelines



Some Timelines



Some Timelines (Continued)



Agile Software Development

Rapid software development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Rapid software development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - User interfaces are often developed using an IDE and graphical toolset.

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.

Agile Alliance

- Several individuals, **The Agile Alliance**,
  - motivated to constrain activities
  - such that certain outputs and artifacts are **predictably** produced.
  - Around 2000, these notables got together to address common development problems.

- Goal: outline values and principles to allow software teams to
  - **develop quickly** and
  - **respond to change.**

Manifesto for Agile Software Development

- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to the value:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

That is, **while there is value** in the items on the **right**, we value the items on the **left** more."

Let's look at these values to discern exactly what is meant.

Value 1: Individuals and Interactions over Processes and Tools

- **Strong players**: a must, but can fail if don't work together.
- **Strong player:** not necessarily an 'ace;' work well with others!
  - Communication and interacting is **more important** than raw talent.
- '**Right' tools** are vital to smooth functioning of a team.
- **Start small**. Find a free tool and use until you can demo you've outgrown it. Don't assume bigger is better. Start with white board; flat files before going to a huge database.
- **Building a team** more important than **building environment.**
  - Some managers build the environment and expect the team to fall together.
  - Doesn't work.
  - Let the team build the environment on the **basis of need**.

Value 2: Working Software over Comprehensive Documentation

- **Code** – not ideal medium for communicating rationale and system structure.
  - Team needs to produce human readable documents describing system and design decision rationale.
- **Too much documentation is worse than too little**.
  - Take time; more to keep in sync with code;
- **Short rationale and structure document**.
  - Keep this in sync; Only highest level structure in system kept.
  - **Fatal flaw**: Pursue documentation instead of software:
- **Rule**: Produce no document unless need is immediate and significant.

Value 3: Customer Collaboration over Contract Negotiation (1 of 2)

- Not possible to describe software requirements up front and leave someone else to develop it within cost and on time.
- Customers **cannot** just cite needs and go away
- Successful projects require **customer feedback on a regular and frequent basis** – and not dependent upon a contract or SOW.

Value 3: Customer Collaboration over Contract Negotiation (2 of 2)

- **Best contracts are NOT** those specifying requirements, schedule and cost.
  - Become meaningless shortly.
- **Far** better are **contracts that govern the way the development team and customer will work together**.
- Key is intense collaboration with customer and a contract that governed collaboration rather than details of scope and schedule
  - Details ideally **not** specified in contract.
  - Rather contracts could pay when a block (deliverable) passed customer's acceptance tests.
  - With frequent deliverables and feedback, **acceptance tests** never an issue.

Value 4: Responding to Change over Following a Plan

- **Our plans and ability to respond to changes is critical!**
- **Course of a project cannot be predicted far into future.**
  - **Too many variables; not many good ways at estimating cost**
  - **As developers gain knowledge of the system and as customer gains knowledge about their needs, some tasks will become unnecessary.**
- **Better planning strategy: – make detailed plans for the next few weeks, very rough plans for the next few months, and extremely crude plans beyond that.**
- Need to know what we will be working on the next few weeks; roughly for the next few months; a vague idea what system will do after a year.
- **Only invest in a detailed plan for immediate tasks**; once plan is made, difficult to change due to momentum and commitment.
  - The lower resolution parts of the plan can be changed with relative ease.

Agile Principles

Principle 1: Our Highest Priority is to **Satisfy the Customer** through **Early** and **Continuous Delivery** of Valuable Software

---

- Number of practices have significant impact upon quality of final system:
  1. Strong **correlation** between **quality** and **early delivery of a partially functioning system.**
  - The less functional the initial delivery, the higher the quality of the final delivery.

  2. Another strong **correlation** exists between **final quality** and **frequently deliveries of increasing functionality**.
  - The more frequent the deliveries, the higher the final quality.

- **Agile processes deliver early and often**.
  - Rudimentary system first followed by systems of increasing functionality every few weeks.
  - Customers may use these systems in production, or
  - May choose to review existing functionality and report on changes to be made.

Principle 2: Welcome Changing Requirements, even late in Development. Agile Processes harness change for the Customer's Competitive Advantage.

- This is a statement of **attitude**.
- Participants in an agile process are not afraid of change.
  - Requirement changes are good;
  - Mean team has learned more about what it will take to satisfy the market.

- Agile teams work to keep the **software structure flexible**, so requirement change impact is minimal.

Principle 3: Deliver Working Software Frequently
(From a couple of weeks to a couple of months with a preference to the shorter time scale)

- We deliver working software.
  - **Deliver early and often**.
  - **Be not content** with delivering bundles of documents, or plans.
  - Don’t count those as true deliverables.

- The **goal** of delivering software that satisfies the customer’s **needs…incrementally**!

Principle 4: Business People and Developers Must **Work Together 'Daily'** throughout the Project.

- For agile projects, there must be **significant** and **frequent interaction** between the
  - customers,
  - developers, and
  - stakeholders.

An agile project must be **continuously guided**.

Principle 5: Build Projects around Motivated Individuals. (Give them the environment and support they need, and trust them to get the job done.)

- **An agile project has people the most important factor of success.**
  - –All other factors, process, environment, management, etc., are considered to be second order effects, and are subject to change if they are having an adverse effect upon the people.

- **Example:** if the office environment is an obstacle to the team, **change the office environment**.
- If certain process steps are obstacles to the team, **change the process steps**.

Principle 6: The Most Efficient and Effective Method of Conveying Information to and within a Development Team is **face-to-face Communications**.

- In agile projects, developers ***talk*** to each other.
  - The **primary mode of communication is conversation**.
  - Documents may be created, but there is no attempt to capture all project information in writing.
- An agile project team **does not demand written** specs, written plans, or written designs. **DISCUSS!**
  - They may create them if they perceive an immediate and significant need, but they are not the default.
  - The **default is conversation**.

Principle 7: Working Software is the Primary Measure of Progress

- Agile projects measure their progress by measuring the amount of **working software**.
  - Progress **not measured** by phase we are in, **or**
  - by the volume of produced documentation **or**
  - by the amount of code they have created.
- **Agile teams are 30% done when 30% of the necessary functionality is working.**

Principle 8: Agile Processes promote sustainable development The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

- An agile project is **not run like a 50 yard dash**; it is run like a marathon.
  - The team does not take off at full speed and try to maintain that speed for the duration.
  - Rather they run at a fast, but **sustainable**, pace.

- Running too fast leads to **burnout**, shortcuts, and debacle.
- Agile teams pace themselves.
  - They don't allow themselves to get too tired.
  - They don't borrow tomorrow's energy to get a bit more done today.
  - They work at a **rate** that allows them to maintain the highest quality standards for the duration of the project.

Principle 9: Continuous Attention to Technical Excellence and Good Design enhances Agility

- **High quality is the key to high speed**.
  - The way to go fast is to **keep the software as clean and robust as possible.**
  - Thus, all agile team-members are **committed** to producing only the **highest quality code** they can.
  - They do not make messes and then tell themselves they'll clean it up when they have more time.

Principle 10: Simplicity – the art of maximizing the amount of work not done – is essential

- Agile teams **take the simplest path** that is consistent with their goals.
  - They don't anticipate tomorrow's problems and try to defend against them today.
  - **Rather they do the simplest and highest quality work today, confident that it will be easy to change if and when tomorrow's problems arise.**

Principle 11: The Best Architectures, Requirements, and Designs emerge from **Self-Organizing Teams**

- **An agile team is a self organizing team**.
  - Responsibilities are **not handed to individual team members** from the outside.
  - Responsibilities are **communicated** to the team as a whole, and the **team determines** the best way to fulfill them.

- Agile team members **work together on all project aspects**.
  - Each is allowed **input** into the whole.
  - No single team member is responsible for the architecture, or the requirements, or the tests, etc.
  - The team **shares those responsibilities** and each team member has influence over them.

Principle 12: At regular Intervals, the **team reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.

- An agile team **continually adjusts its organization,** rules, conventions, relationships, etc.

- An agile team knows that its environment is continuously changing, and knows that they must change with that environment to remain agile.

Agile method applicability

- Product development where a software company is developing a small or medium-sized product for sale.
- Custom system development within an organization, where there is a **clear commitment from the customer to become involved** in the development process and where there are **not a lot of external rules and regulations that affect the software.**
- Because of their focus on small, tightly-integrated teams, there are problems in **scaling agile methods to large systems**.

Problems with agile methods

- It can be **difficult** to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the **intense involvement** that characterizes agile methods.
- **Prioritizing changes** can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

Agile methods and software maintenance

- Most organizations spend more on maintaining **existing** software than they do on **new** software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:
  - Are systems that are developed using an agile approach **maintainable**, given the emphasis in the development process of minimizing formal documentation?
  - Can agile methods be used effectively for evolving a system in response to customer change requests?
- Problems may arise if original development team cannot be maintained.

Technical, human, organizational issues (1)

- **Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:**
  - Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a **plan-driven approach.**
  - Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using **agile methods.**
  - How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

Technical, human, organizational issues (2)

- What type of system is being developed?
  - Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).
- What is the expected system lifetime?
  - Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.
- What technologies are available to support system development?
  - Agile methods rely on good tools to keep track of an evolving design
- How is the development team organized?
  - If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

Technical, human, organizational issues (3)

- Are there cultural or organizational issues that may affect the system development?
  - Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
- Is the system subject to external regulation?
  - If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

Conclusions

- The professional goal of every software engineer, and every development team, is to deliver the highest possible value to our employers and customers.
  - And yet, our projects fail, or fail to deliver value, at a dismaying rate.
- Though well intentioned, the **upward spiral of process inflation** is culpable for at least some of this failure.
- The principles and values of agile software development were formed as a way
  - to help teams break the cycle of process inflation, and
  - to focus on simple techniques for reaching their goals.

- Agile processes include
  - SCRUM,
  - Crystal,
  - Feature Driven Development,
  - Adaptive Software Development (ADP), and most significantly,
  - Extreme Programming.

References and Bibliography

- *Software Engineering,* by Ian Sommerville, 9th Edition, ISBN. 0-13-703515-7. Pearson: Addison-Wesley, 2011 –Chapter 3
- Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002), *Agile Software Development Methods. Review and Analysis*, VTT Publications, Espoo, Finland, ISBN 9513860094. www.inf.vtt.fi/pdf/
- Agile Manifesto, http://agilemanifesto.org/ [Last Accessed: November 2009]
- Beck, K. (2000), '*Extreme Programming Explained: Embrace Change*', Addison-Wesley, ISBN 0201616416.
- Boehm, B. W. (1988), 'A Spiral Model of Software Development and Enhancement', *IEEE Computer*, 21(5), pp.61-72.
- Fowler, M. (2005), 'The New Methodology', http://martinfowler.com/articles/newMethodology.html#XpextremeProgramming [Last Accessed: November 2009]
- Larman, C. (2004), *Agile and iterative Development: A Manager's Guide*, Addison-Wesley, ISBN 0131111558.
- Larman, C. and Vodde, B. (2009), Scaling Lean and Agile Development, Addison-Wesley, ISBN 0321480961.
- Rumbaugh, J., Jaconson, I. and Booch, G. (1999), *The Unified Software Development Process*, Addison-Wesley, ISBN 0201571692.
- Schwaber, K. and Sutherland, J. (2009), Scrum Guide, http://www.scrum.org [Last Accessed: November 2009]
- https://www.unf.edu/~broggio/cen4010/Ch3 - Agile Development.ppt

Further Reading

- Larman, C. & Basili, V. (2003), Iterative and Incremental Development: A Brief History, *IEEE Computer*, June 2003, pp 2-11. See http://www.craiglarman.com for a copy of this paper. Or via 'Further Reading' on Wikipedia information on 'Agile Software Development'

- Gilb, T. http://www.Gilb.com under 'Downloads' then under 'Gilb Papers'
  - Decomposition April 4 2008.pdf: Decomposition of Projects - How to design small incremental result steps
  - CAI tomgilbinterview1.pdf: What are Evolutionary (EVO) Development Methods? CAI Interview on 21 February 2006
  - Evo Principles.pdf: Fundamental Principles of Evolutionary Project Management

- Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002), *Agile Software Development Methods. Review and Analysis*, VTT Publications, Espoo, Finland, ISBN 9513860094. See http://www.inf.vtt.fi/pdf/

- Schwaber Scrum Video http://video.google.com/videoplay?docid=-7230144396191025011&q=Google+Tech+Talks&hl=en#
  (61 minutes)
- http://www.Agilealliance.org/