

A Guide to Selecting Software Development Methodologies

Hamid Faridani
(h.faridani@rogers.com)
March 2011

Introduction

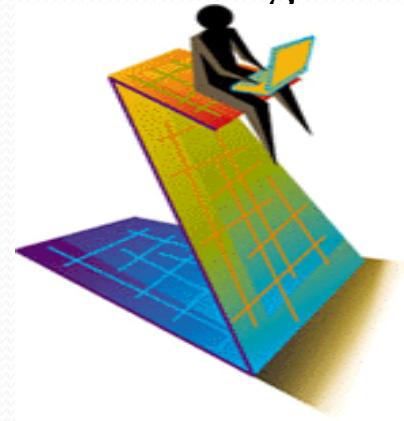
- Methodologies like Waterfall, RUP and Agile have all become key tools for software developers and project manager's to aid them in delivering projects on time, on budget while meeting customer's requirements.
- With so many methodologies, how does a project manager know which methodology is the right one to produce satisfactory results.
- In this talk I will address this question by considering seven critical success factors such as scope, resources, size etc for project's success. Each factor is dealt differently depending on the methodology that is being used. This study analyzes these factors and indicates which type of methodology (light versus heavy) is suitable based on each factor.

Agenda

- Software Development Methodologies
 - The Need for Software Development Methodology
 - Origin and Evolution of Software Development Methodologies
- Overview of key methodologies
 - Description
 - Benefits and drawbacks
 - Recommendations and Hints and Tips
- Project Characteristics
 - Decision Tree Analysis
- Conclusions
- References
- Q & A

Software Development Methodology

- A software development methodology is a structure imposed on the development of a software product.
 - It includes procedures, techniques, tools and documentation aids which will help system developers in their task of implementing a new system.
 - The intent of a methodology is to formalize what is being done, making it more repeatable.



Need for Software Development Methodology

- During the last 4 decades a number of development methodologies have appeared in the marketplace
- Their messages are sometimes confusing and contradictory ...
- All of them seem to be similar... but at the same time very different..
- How do I know what's the best fit for my project ?



Need for Software Development Methodology

- According to the Standish group(<http://www.sandishgroup.com>) 2009 study, 32% of all projects are delivered on time, on budget, 44% were challenged which are late, over budget, and 24% failed which are cancelled prior to completion or delivered and never used."
- A study conducted by the Forrester research group [1] states that nearly one-third of all IT projects commenced would be an average of three months late . In many cases the failure is the result of either not using a methodology or using the wrong methodology
- According to CIO.com [2], nearly three quarters of all IT projects in the Internet era that were conceived in the last seven years have suffered from one or more of the following: total failure, cost overruns, time overruns, or a rollout with fewer features or functions than promised

Software Development Methodology



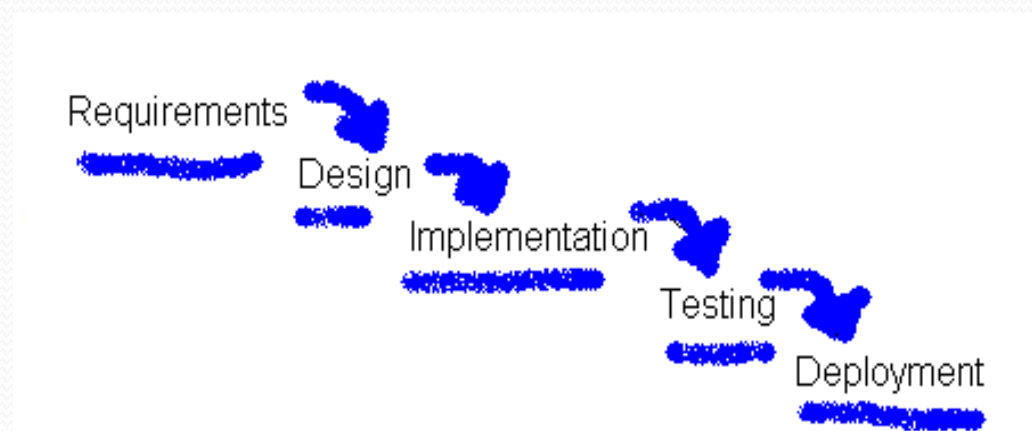
- In the earliest days of software development, code was written and then debugged (Code-and-Fix model). There was no formal design or analysis....
- Since the approach to developing complex hardware systems was well understood, it provided a model for developing software.
- These methods approached system development in a requirements/design/build paradigm with standard, well-defined processes. These methodologies are referred to as **Heavy Methodologies** or plan driven.
- However, newer methodologies have started making an appearance in software projects. These methodologies unlike the more classical ones are considered to be more agile and more able to adapt to change. They do not focus on a long development cycle but rather on short iterations, lightweight processes and rely heavily on close customer involvement . These types of methodologies have come to be known as **Light or Agile Methodologies**

Overview of key Heavy and Light Methodologies

- The next few slides provide a overview of both heavy and agile methodologies and a comparison between them to highlight their similarities and differences.
- For heavyweight methodologies I will review the followings: Waterfall, Unified Process and Spiral
- For agile methodologies I will review: Extreme Programming (XP) and Scrum

Waterfall Methodology

- During the 1960s, “code and fix” was the method employed by software developers.
- Due to the difficult nature of “code and fix” approach, Winston Royce in 1970 proposed the waterfall methodology [3] to deal with the increasing complexity of aerospace software .
- The waterfall approach emphasizes a structured progression between defined phases. Each phase consists of a definite set of activities and deliverables that must be accomplished before the following phase can begin.



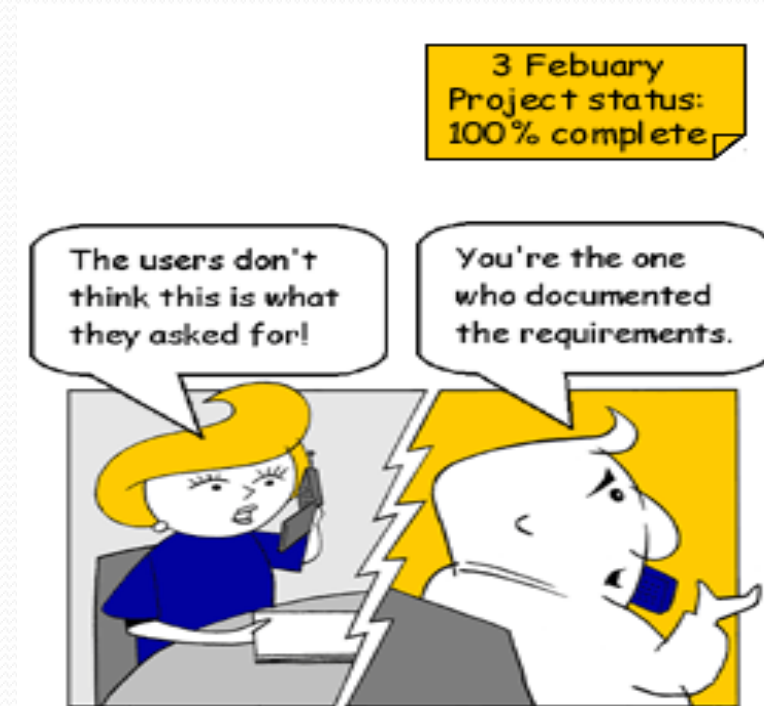
Benefits and Drawbacks of the Waterfall Methodology

Benefits:

- Disciplined process
- Forces to have complete requirements prior to start
- Forces analysis and design first

Drawbacks:

- No early feedback (prototyping)
- Slow to respond to change
- High cost for missed or unclear requirements
- It is optimized for hardware, thereby neglecting the essential characteristics of software .



Software Development is Inherently Complex

- Users generally find it very hard to give precise expression to their needs in a form that developers can understand.
- Systems sizes are growing, which demands a large team of developers and more complex communication and coordination
- In large systems, there is combinatorial explosion that makes the number of paths through the code very large. This makes exhaustive testing impossible.
- Lack of mathematical tools and difficulty to model the complete behavior of large discrete systems, forces us to be content with sub-optimal levels of confidence regarding their correctness.

Evolution of Software Development Methodology

Software development methodologies have evolved in order to:

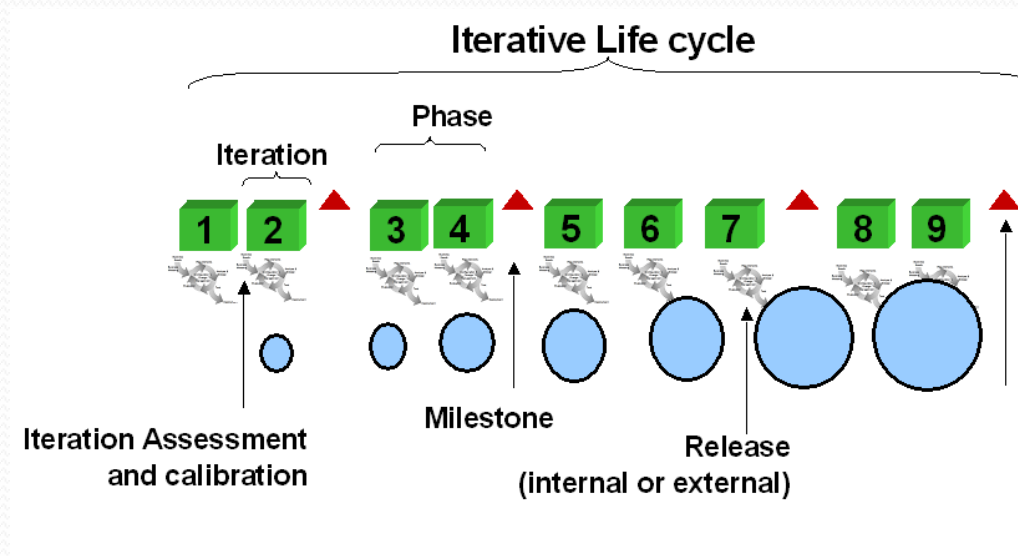
- Optimize the waterfall model
- Allow for software prototyping to help provide a better understanding of the requirements
- Reducing time to market and pressures for improved quality
- Increase the ability for reuse



Incremental and Iterative (IID) Approach

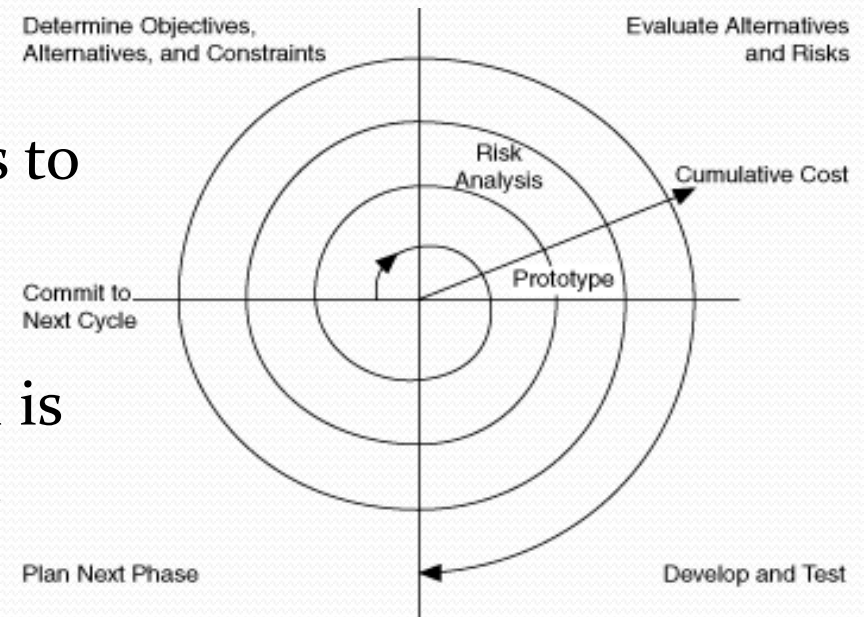
Iterative and Incremental development approach was developed in response to the weaknesses of the waterfall methodology[4] . It starts with an initial planning and ends with deployment with the cyclic interactions in between.

- **Incremental:** Additional functionality is implemented in each increment/release
- **Iterative:** Repeat the cycle of design, build and test until the desired functionality is complete



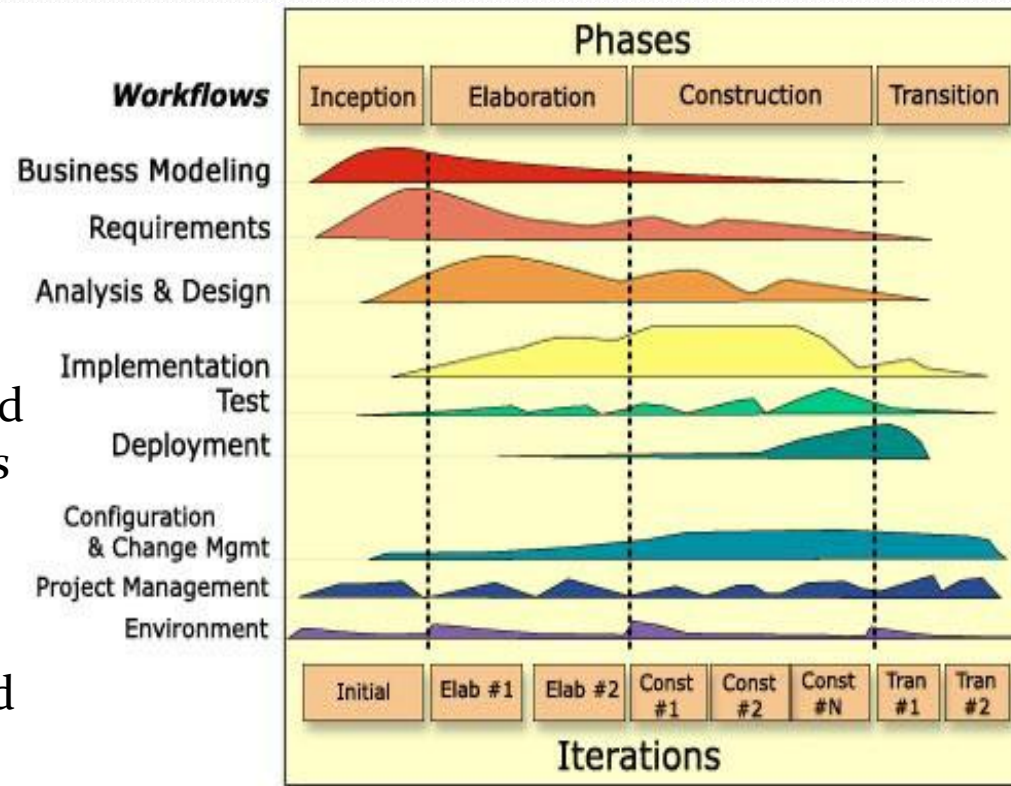
The Spiral Methodology

- The spiral model is an IID developed in 1988 by Larry Boehm[5].
- As originally envisioned, the iterations were typically 6 months to 2 years long
- Combines prototyping and the waterfall model. The spiral model is intended for large, expensive, and complicated projects.
- The aim of this methodology was to shift the emphasis to risk evaluation and resolution.



RUP (Rational Unified Process)

- The Rational Unified Process provides guidelines, templates and tools necessary for the entire team to take full advantage of among others the following best practices [6]:
 - Develop software iteratively and incrementally
 - Manage requirements using use cases
 - Use component-based architectures
 - Visually model software using UML
 - Verify software quality
 - Control changes to software
- The horizontal axis represents time and shows the dynamic aspect of the process and it is expressed in terms of cycles, phases, iterations, and milestones.
- The vertical axis represents the static aspect of the process: how it is described in terms of activities, artefacts, workers and workflows.



Common Characteristics of Heavy Methodologies

The heavyweight methodologies have these similar characteristics.

- **Predictive approach** – Heavy methodologies have a tendency to first plan out a large part of the software process in great detail for a long span of time. This approach follows an engineering discipline where the development is predictive and repeatable.
- **Comprehensive Documentation** – Traditional software development view the requirements document as the key piece of documentation. A main process in heavyweight methodologies is the big design upfront (BDUF) process.
- **Process Oriented** - The goal of heavy methodologies is to define a process that will work well for whoever happens to be using it . The process would consist of certain tasks that must be performed by the managers, designers, coders, testers etc. For each of these tasks there is a well defined procedure.
- **Tool Oriented** – Project management and software development tools must be in use for completion and delivery of each task.

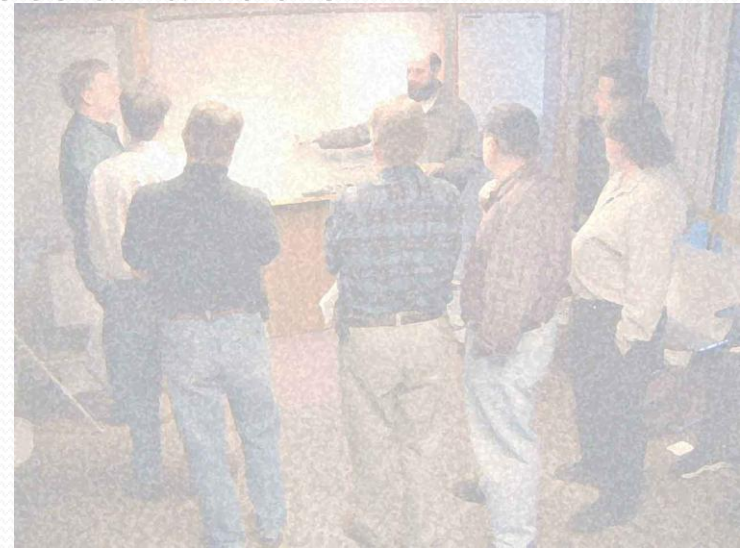
Agile Software Development

- In February 2001, 17 software developers met in a ski resort in Utah to discuss lightweight development methods. They published the "Manifesto for Agile Software Development" to define the approach now known as agile software development.

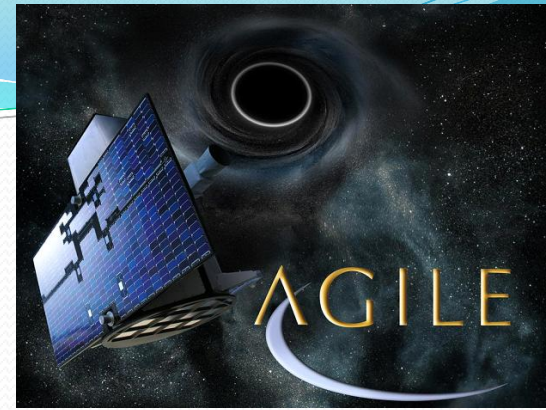
Agile Manifesto reads, in its entirety, as follows:

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

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan



Agile Approaches



Key principles :

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress.
- Even late changes in requirements are welcomed.
- Close, daily, cooperation between business people and developers
- Face-to-face conversation is the best form of communication.
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design.
- Self-organizing teams
- Regular adaptation to changing circumstances

SCRUM

- Scrum is a software-management process for iterative-incremental software development projects.
- At the end of every iteration it produces a potential set of functionality.
- The term ‘scrum’ originated from a strategy in the game of rugby where it denotes “getting an out-of-play ball back into the game” with teamwork [7] .
- Scrum relies on self-organization, with the team deciding what to do while management runs interference and removes roadblocks.

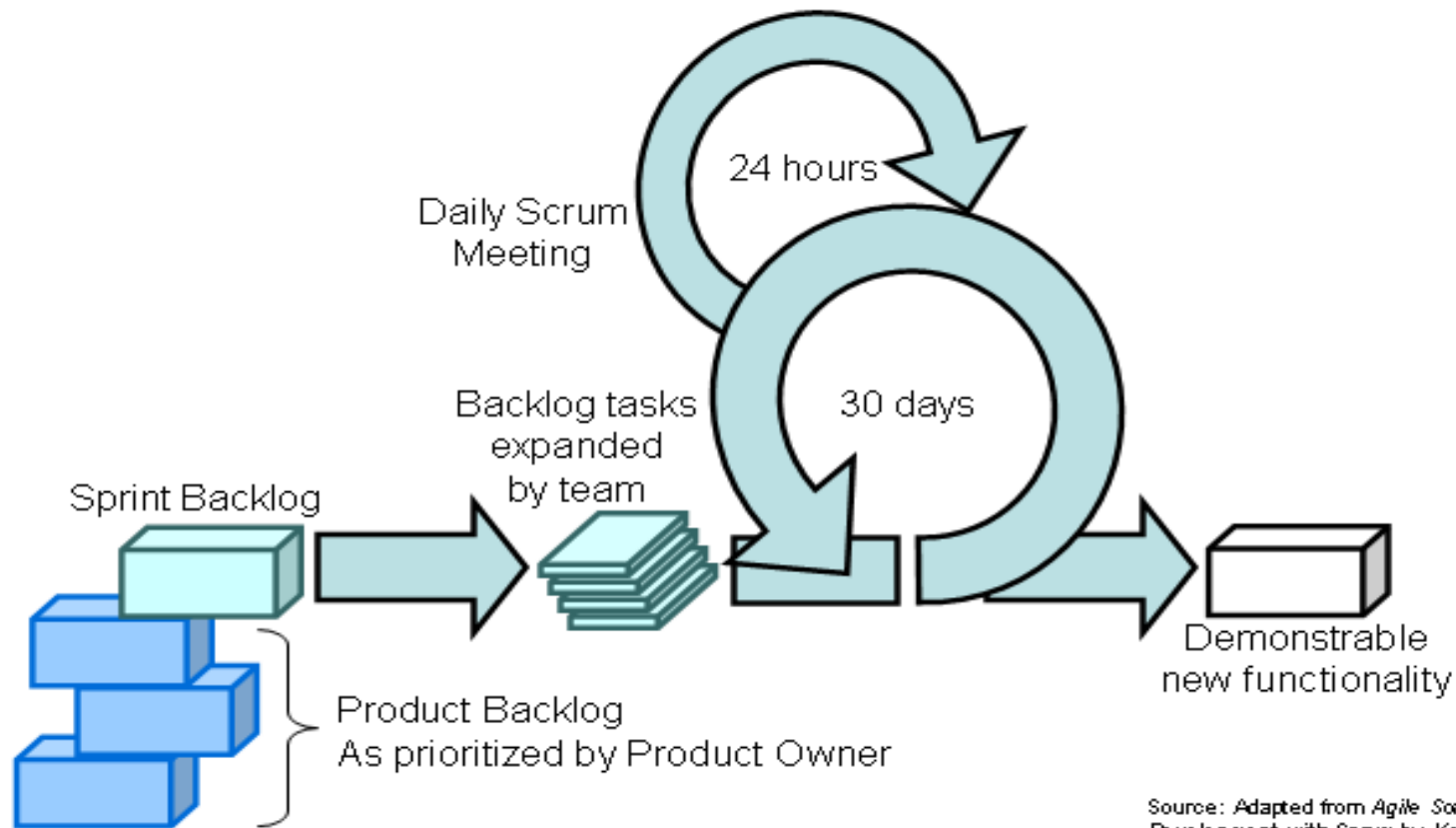


SCRUM

Key characteristics :

- Customers become a part of the development team.
- Frequent intermediate deliveries with working functionality.
- Frequent risk and mitigation plans developed by the development team itself.
- A daily status discussion asking each team member:
 - What have you done since yesterday?
 - What are you planning to do by tomorrow?
 - Do you have any problems preventing you from accomplishing your goal?
- Frequent stakeholder meetings to monitor progress

SCRUM's Lifecycle



Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

Extreme Programming (XP)

- XP is an iterative and incremental development methodology[8]
- XP focuses on programming , where Scrum focuses on Project management. Both are “light” techniques and are often used together.
- Scrum iterations are longer than XP (2 weeks vs. 4 weeks for Scrum sprints)
- Scrum teams do not allow changes into their sprints. XP teams are much more amenable to change within their iterations.



Key XP Practices

- **Planning:** make a rough plan quickly and refine it as things become clearer
- **Pair Programming:** pairs of developers write all production code
- **Test driven development:** developers write tests for every function that could possibly break, before they write the code
- **Refactoring:** a technique of improving code without changing functionality. An XP team refactors mercilessly
- **Continuous Integration:** XP teams integrate their code several times a day, after they get all the unit tests to run.
- **On-site customer:** XP team needs to have a customer available on site to clarify requirements and to make critical business decisions

How do I know if Agile is appropriate for my project?

Consider using agile development in the following situations:

- Environments experiencing rapid change
 - Unclear/emerging requirements
- High Priority / Revenue-Producing Projects → When time to market is critical
 - Agile was designed for on-time delivery, and if required releasing early increments of functionality
- Project Remediation/Rescue
 - By focusing on immediate delivery of functionality
 - Constant delivery of working, bug-free software could quickly build the trust between the business and the delivery team.

Project Characteristic Analysis

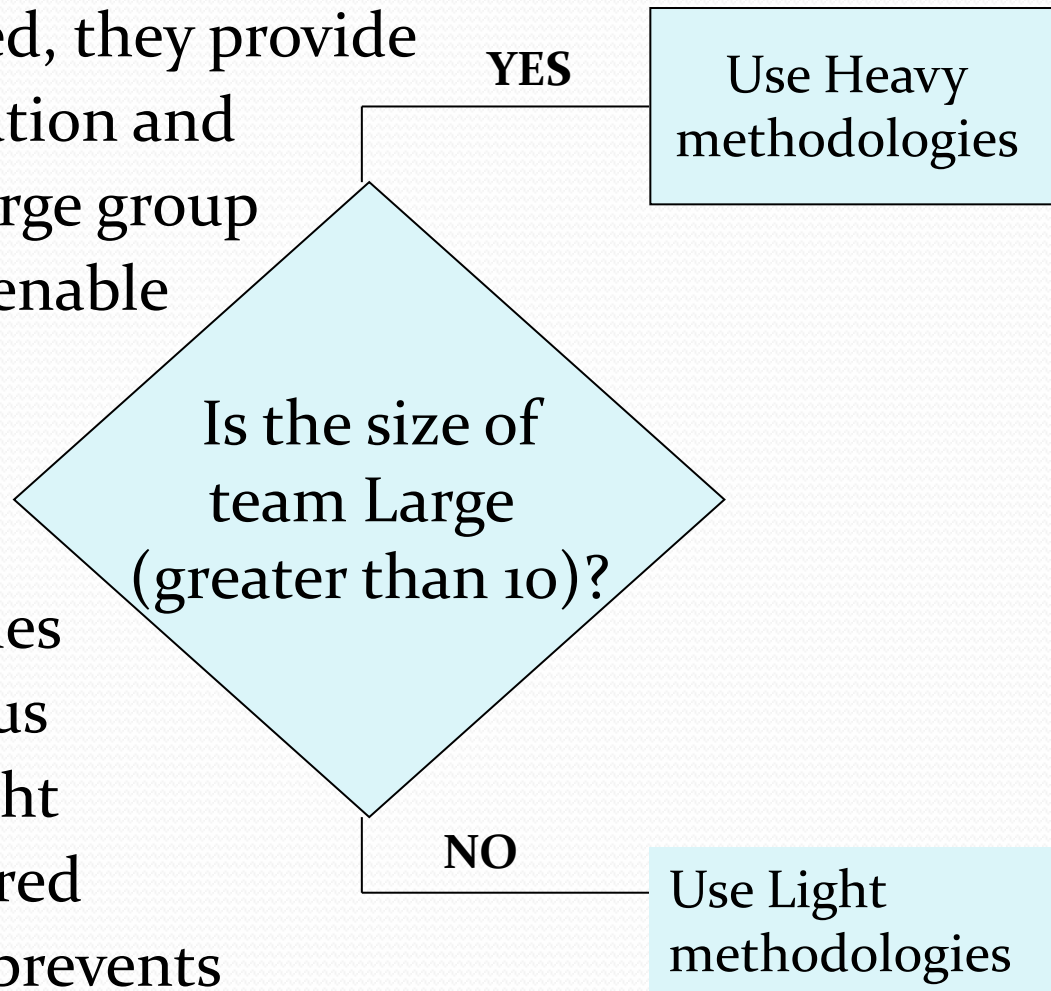
- Key project characteristics considered:
 - Size of the project team
 - Rate of expected change
 - Primary project goal
 - Requirement Management
 - Project Communication
 - Customer Relationship
 - Customer Organizational Culture

Project Characteristic Analysis

- General comments:
 - A decision tree analysis is used to compare various methodologies
 - They do not constitute parts of one large decision tree- it is a one-dimensional analysis
 - The ranking of the seven characteristics would have to be done by the project manager and architect with the assistance of the project leaders.
 - The results are not meant to be a substitute for sound project management and technical direction but as guidelines for practitioners.
 - The methodology used can also depends on the customer request

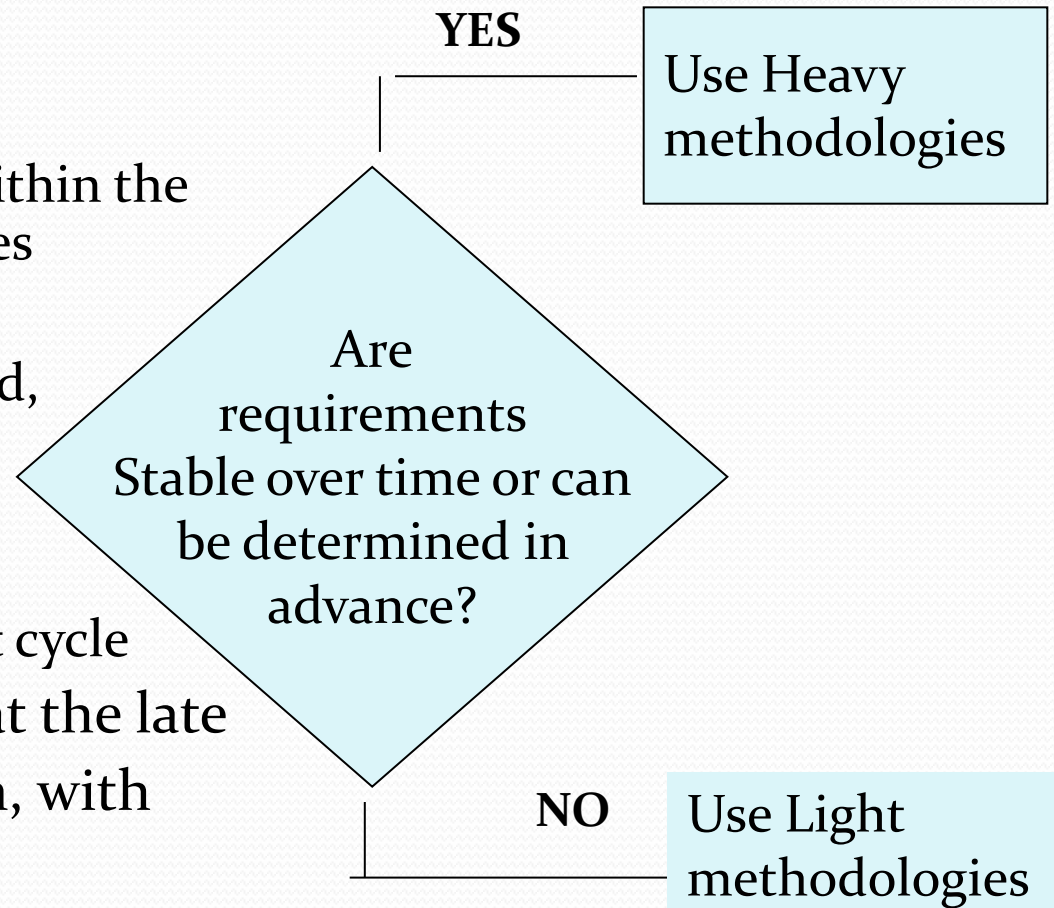
Size of the Team

- Heavy methodologies are highly process and document oriented, they provide for better communication and coordination across large group
- Small, dynamic teams enable interactions and adaptation through close relationships and clear responsibilities
- Industry wide consensus is that the need for tight coordination and shared knowledge generally prevents agile methods with teams over forty [9]



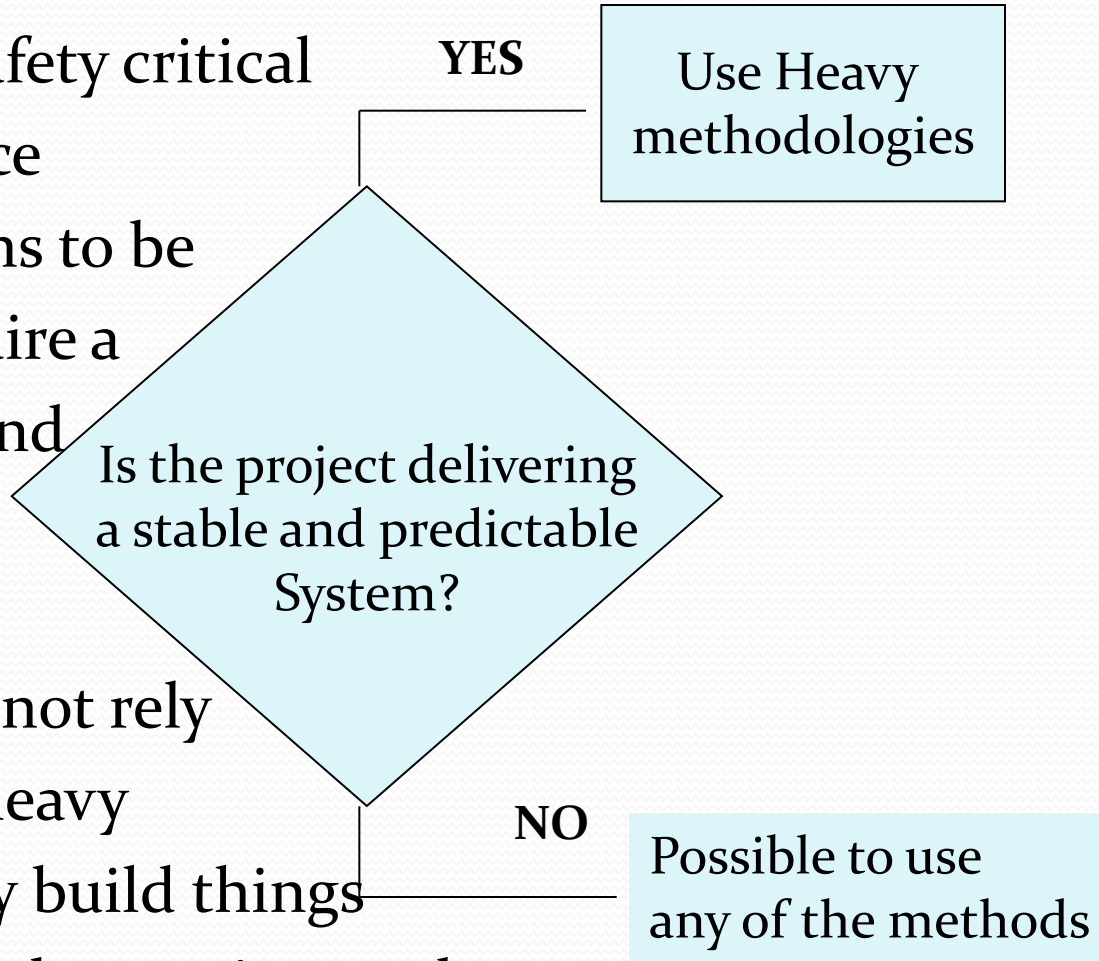
Rate of Expected Change

- Light Methodologies:
 - Are not concerned with how the project integrates within the overall infrastructure or scales in production
 - Because of their low overhead, can respond to change quicker and can support changes to the product even quite late in the development cycle
- The cost of change, specially at the late stages of development is high, with Heavy methodologies
- Heavy methodologies work best when the requirements are largely determinable in advance and remain stable



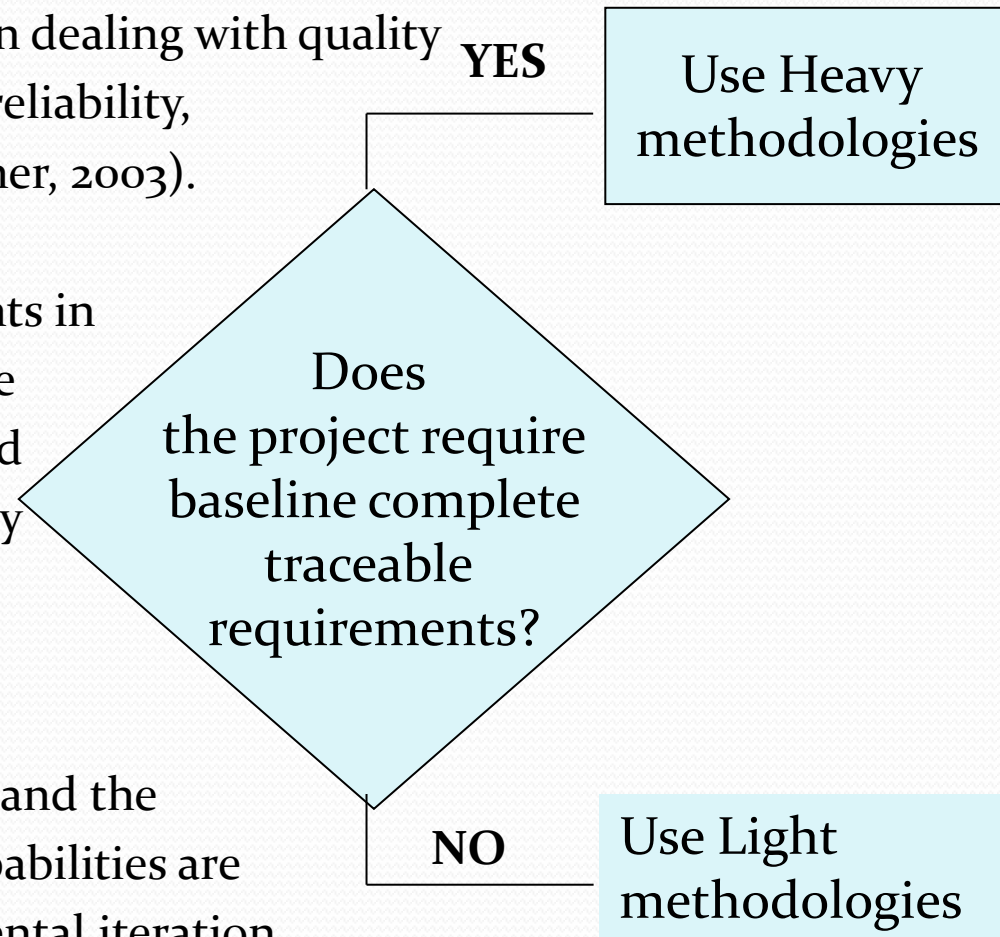
Primary Project Goal

- For projects which are safety critical and demand high assurance a heavy methodology seems to be a better fit, since they require a documented set of plans and specifications and adherence to standards
- Light methodologies do not rely on plan-driven goals and heavy documentation rather they build things quickly and find out through experience what activity or feature will add the most value next [10].



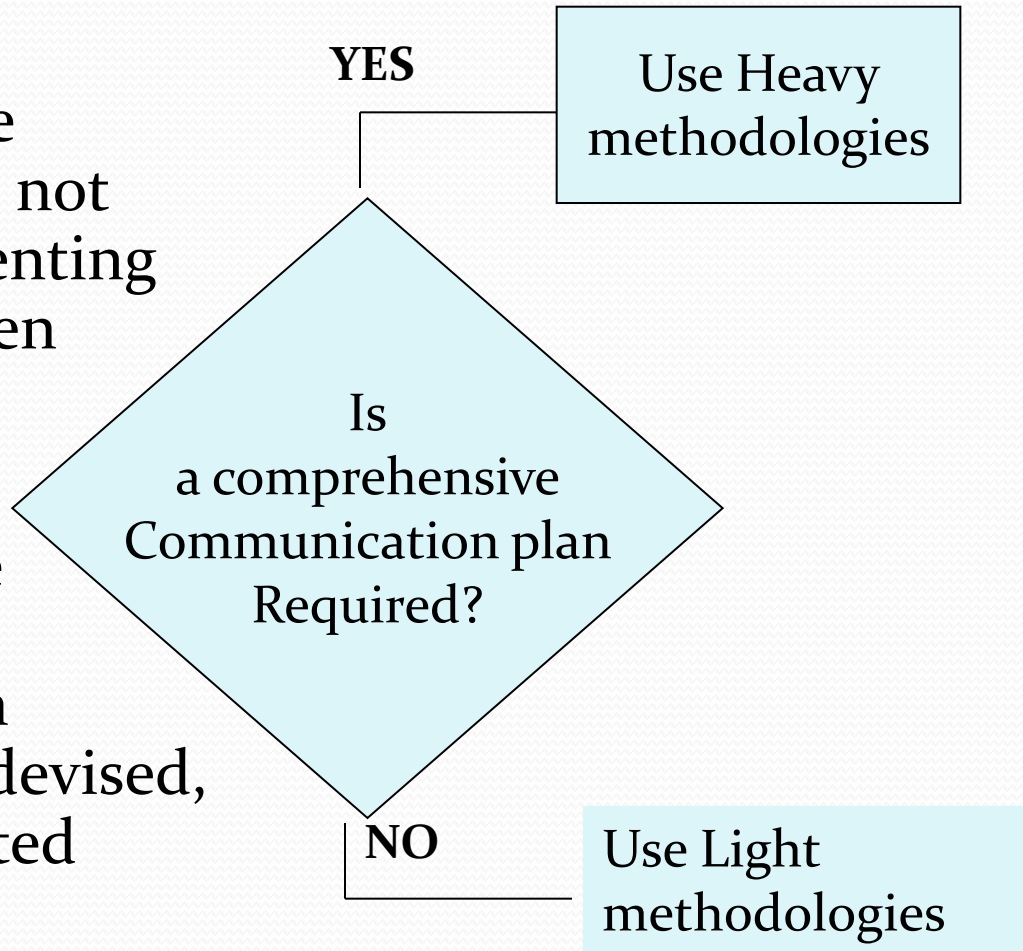
Requirement Management

- Heavy methodologies first identify requirements, define requirements and then hand them off to the appropriate team.
- Heavy methodologies also focus more on dealing with quality of non functional requirements such as reliability, performance or scalability (Boehm & Turner, 2003).
- Light methodologies express requirements in terms of changeable, informal stories. There is a close interaction between customers and developers to determine the highest-priority set of requirements to be included in each iteration.
- Customers express their strongest needs and the developers assess what combinations of capabilities are feasible for inclusion in the next developmental iteration. Negotiations establish the contents of the next iteration.



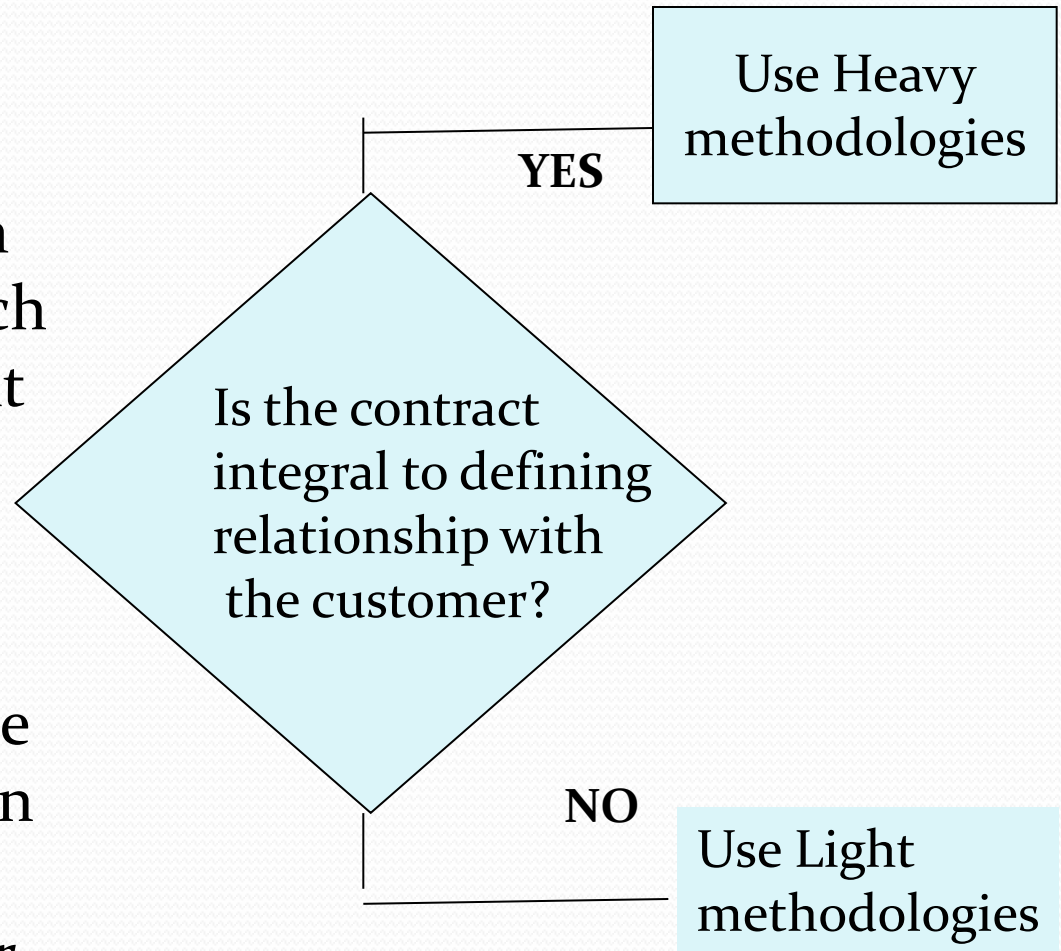
Methodology Decision Analysis: Project Communication

- Light approaches recommend face-to-face communication and are not concerned with documenting communications between entities
- Although distributed development is possible with Light methods, additional collaboration mechanism need to be devised, planned and implemented which may require the tailoring of the process



Methodology Decision Analysis: Customer Relationship

- Light methodologies rely on working software and presence of dedicated customer liaison, that understand user needs in order to ensure total synch between the development team and the customer
- Heavy methodologies generally depend on some form of contracts between the developers and customers as the basis for customer relations.



Methodology Decision Analysis: Organizational Culture

In Light methodologies:

- There are no specific roles or work that has to be completed by a team member
- Each person is expected and trusted to do whatever work is necessary to the success of the project
- If the organizational culture dictates that clear policies and procedures will define a person's role then heavy methodologies tend to best fit that organization. Every person's task is well defined and documented

YES

Use Heavy methodologies

-
- Project roles need to be clearly defined/documented
 - Customer has rigid and bureaucratic culture

NO

Use Light methodologies

Conclusions

- There is no silver bullet keep in mind that :
 - Most modern software development methodologies are based on an iterative approach.
 - Waterfall development still exist and is a valid approach to use. Make sure to use it where makes sense....
- The results are not meant to be a substitute for sound project management but as guidelines.
- In cases where the decision trees contradict each other, one pointing to a heavy methodology and one pointing to a light methodology, the seven characteristics must be prioritized based on organizational culture and project situation.

References

1. Hoffman, T. (2003, July 21). Value of Project Management Offices questioned. *Computerworld*. Vol. 37, Iss. 29, pg. 7
2. Berinato, S. (2001). The Secret to Software Success <http://www.cio.com/archive/070101/secret.html>
3. Royce, Winston (1970), "[Managing the Development of Large Software Systems](#)", *Proceedings of IEEE WESCON 26* (August): 1-9,
4. Craig Larman and Victor R. Basili, "Iterative and Incremental Development: A Brief History," *IEEE Computer*, vol. 36, no. 6, pp. 47-56, June 2003
5. Boehm B, "[A Spiral Model of Software Development and Enhancement](#)", *ACM SIGSOFT Software Engineering Notes*, "ACM", 11(4):14-24, August 1986
6. Kruchten, P. (1999). *Rational Unified Process –An Introduction*. Addison-Wesley.
7. K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Upper Saddle River, NJ, Prentice – Hall, 1st Edition, Oct 2001
8. K. Beck, Embracing change with Extreme Programming. *IEEE Computer*, Vol. 32, Issue 10 October 1999.
9. Constantine, L. (2001). *Methodological Agility: Software Development*. Pg 67-69
10. Boehm, B. (2003). *Value-Based Software Engineering*. *ACM SIGSOFT Software Engineering Notes*, Vol. 26-2



Questions ?

Thanks!