ADVANCED PROGRAMING PRACTICES

Software development models

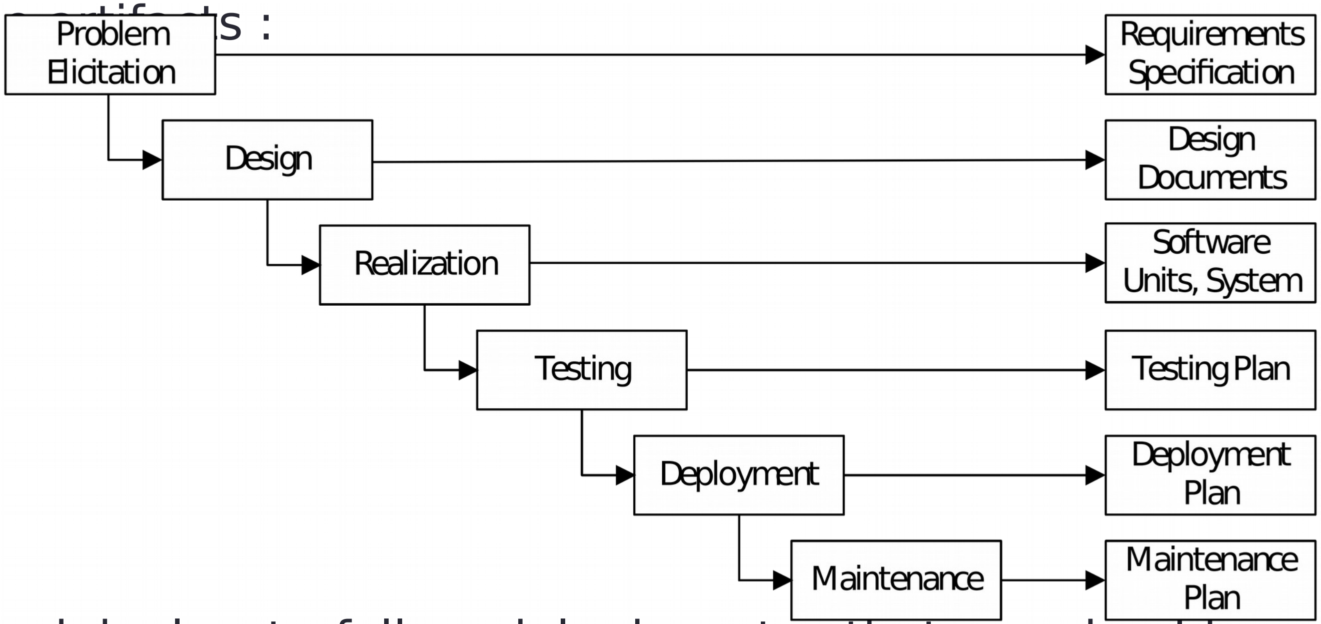
Predictive and agile models

Software development

- At its core, software development aims at producing code.
- However, if one want to produce large, complex, high quality applications, other activities are to be added, based on additional quality concerns:
  - **Are we building the right software? Do we really know what the client needs?**
    - If not, we may be building the wrong software features, and missing important features.
  - **Do we have a solid general plan of action for the design of our entire system?**
    - If not, later additions will be requiring major redesigns.
  - **Is our produced software properly tested before it is delivered?**
    - If not, the resulting software will fail, with disastrous consequences to our client and our reputation.
  - **How do we develop the system now so that its structure will sustain further development before deployment, or maintenance after deployment?**
    - If not, our system will become exponentially harder to develop/maintain, until ultimately it needs to be redone from scratch.
- Software development is a complex activity that requires many more activities and concerns than the core production of software artifacts through coding.

Software development phases: the waterfall model

- One of the earlier software development models was the waterfall model, in which the following phases are followed in order, producing some artifacts :



- The original waterfall model advocates that one should move to a phase only when its preceding phase is reviewed and verified, and that going back to a previous phase is not possible, or prohibitively costly.
- Developed from traditional Engineering processes, where physical artifacts are produced and can hardly be changed as they are designed, produced and used.
- However, software is a malleable artifact, i.e. it can be changed at any time during its lifetime.

Software development models

- A software development model is a definition of a group of related precepts, tasks, or artifacts, that are deemed necessary for the production of software.
- There are numerous examples of software development models, who emphasize different important factors and methods to take into consideration while developing software.
  - **Prototyping**: emphasizes the early development of prototype software to elicit the problem statement and develop early solutions to get feedback.
  - **Iterative and incremental development**: emphasizes the structured use of iterations during software development to bring focus on a few development issues at a time.
  - **Spiral development**: emphasizes on risks associated with a particular problem/solution and to minimize risks.
  - **Rapid application development**: emphasizes on productivity of software artifacts rather than the strict following of an elaborated process.
  - **Extreme programming**: emphasizes on precepts to be followed in order to achieve productivity while controlling potentially chaotic aspects of software development.

Predictive vs. adaptive models

- Software development models can be categorized as either **predictive** or **adaptive**:
  - **Predictive model**: Based on the notion that all activities involved in software development can be predicted and documented along the way, and that further development is based on the information accumulated in previous phases of the development.
  - Such models tend to be descriptive models, i.e. to define all the roles, activities, and artifacts involved in a clearly defined process.
  - Predictive models focus on being able to plan the future in detail.
  - A predictive team can report exactly what features and tasks are planned for the entire length of the development process.
  - Predictive teams have difficulty changing direction. The plan is typically optimized for the original destination and changing direction can cause completed work to be thrown away and done over differently.
  - Predictive teams will often institute a *change control board* to ensure that only the most valuable changes are considered.

Predictive vs. adaptive models

- **Adaptive model**: Based on the notion that software development is characterized by changing information as the development proceeds, and thus that a software development model should be made to cope with change.
- Such models tend to be prescriptive models, i.e. to define a set of precepts to be followed, without an exact definition of a process.
- Adaptive models focus on being able to adapt quickly to changing realities.
- When the needs of a project change, an adaptive team changes with it.
- An adaptive team will have difficulty describing exactly what will happen in the future.
- The further away a date is, the more vague an adaptive method will be about what will happen on that date.
- An adaptive team can report exactly what tasks are being done next week, but only which features are planned for next month.
- When asked about a release six months from now, an adaptive team may only be able to report the mission statement for the release.

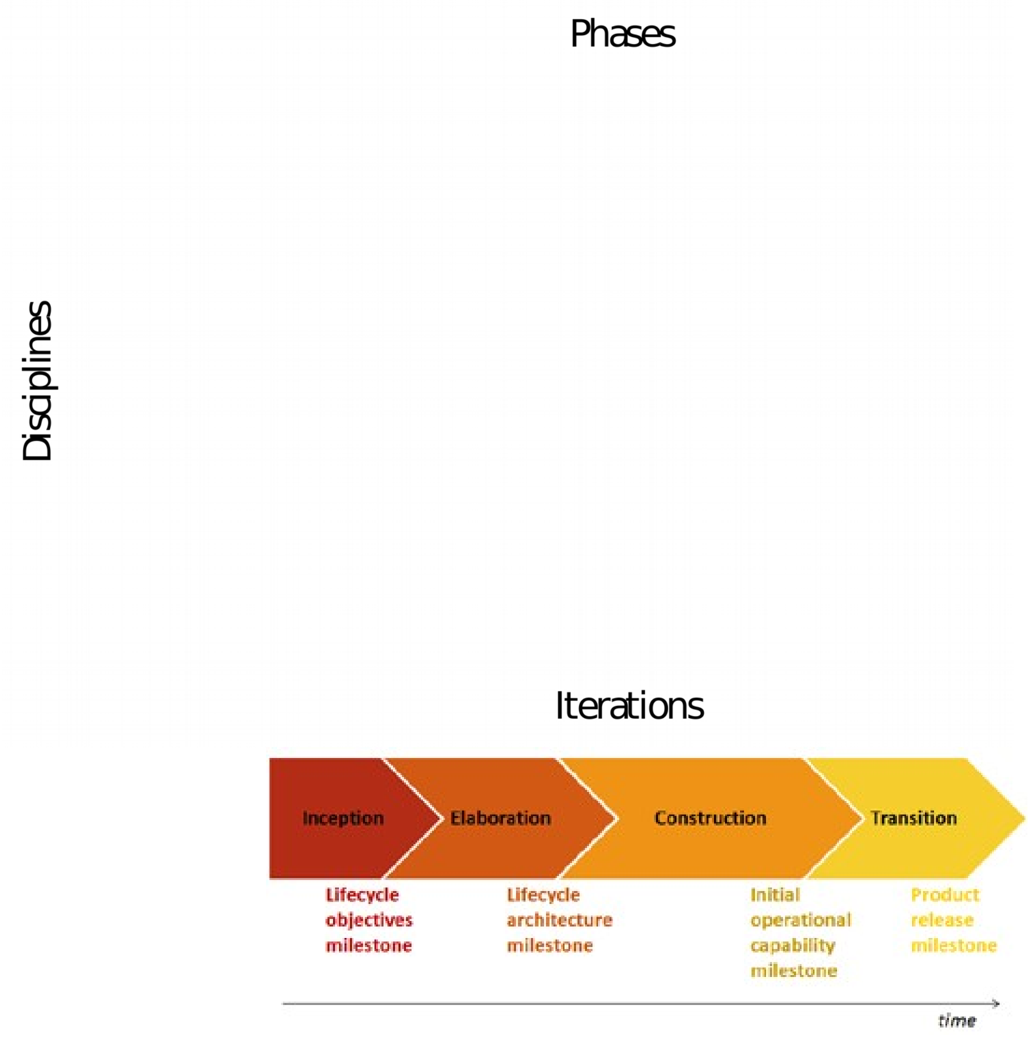
Predictive software development models

Predictive models: software development process

- Generally speaking, a software development process is a formally defined process that defines in details the *who*, *what* and *how* of everything that needs to be done in order to produce software.
- A software development process defines the following entities that all play a role in the development of software:
  - **Actor**: defines a set of skills and responsibilities that are necessary for the achievement of tasks and the production of artifacts in the process.
  - **Artifact**: defines a product resulting from the achievement of a task, which is then used as input for further tasks in the process.
  - **Task**: defines a unit of work that aims at producing one or more artifacts, using certain tools and techniques.

Software development process example: Rational Unified Process (RUP)

- A good example of a software development process is IBM’s Rational Unified Process (RUP).
- Process that defines:
  - **Disciplines**: major areas of concern in software development
  - **Phases**: plan of action for each discipline, ranging from abstract thinking to concrete development to deployment.
  - **Iterations**: any number of iteration is allowed in each phase in order to reach for the set goals of the phase


Phases
Disciplines
Iterations
Inception
Elaboration
Construction
Transition
Lifecycle objectives milestone
Lifecycle architecture milestone
Initial operational capability milestone
Product release milestone
time

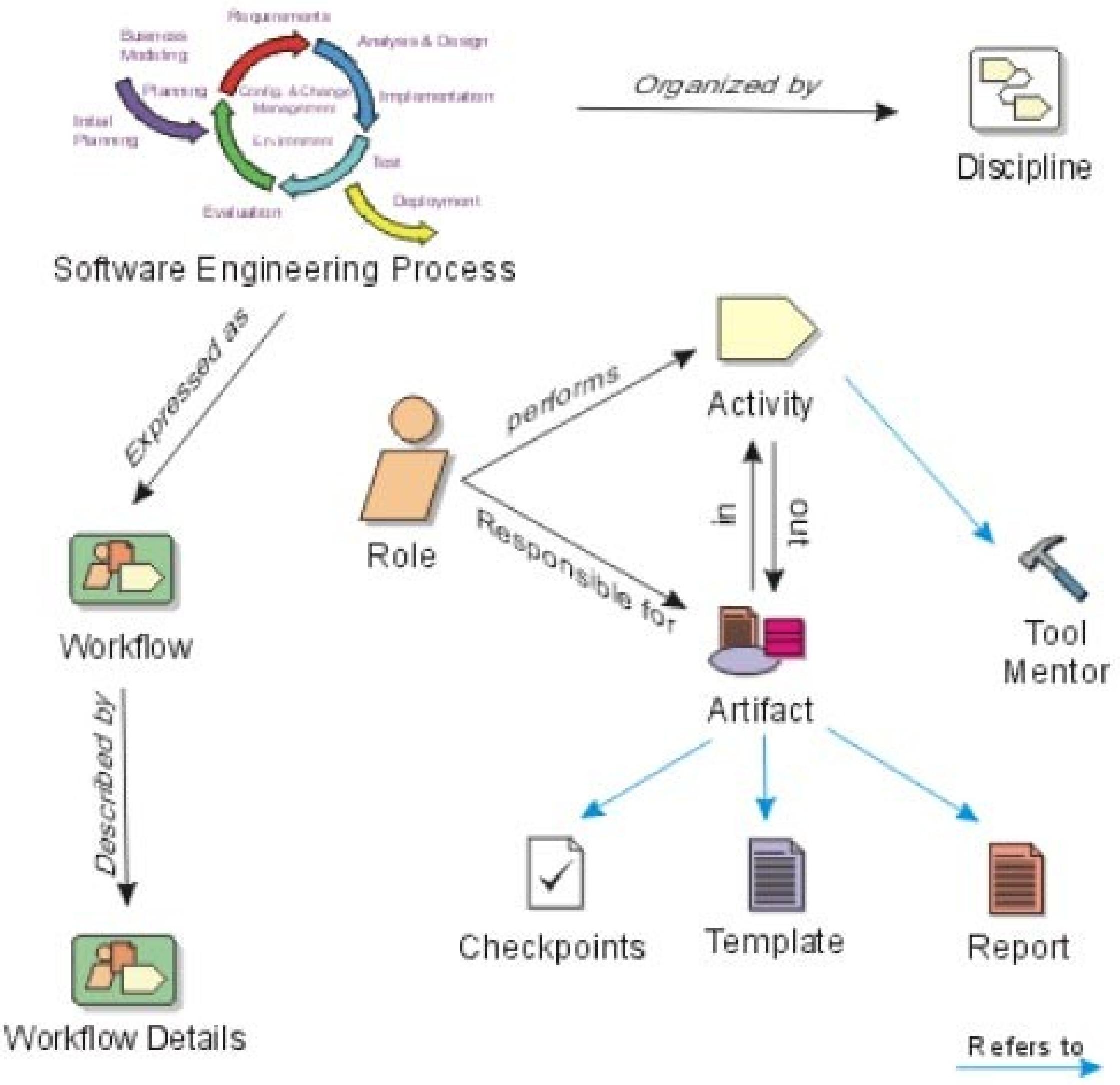
Software development process example: Rational Unified Process (RUP)

- The RUP uses the notion of iterative development.

- **Iterative development** is a design methodology based on a *cyclic* process of prototyping, testing, analyzing, and refining a product.
  - Based on the results of testing the most recent iteration of a design, changes and refinements are made.
  - This process is intended to ultimately improve the quality and functionality of a design and/or implementation.
  - In iterative design, interaction with the designed system is used as a form of research towards the evolution of a project, as successive versions, or iterations of a design are implemented.

Software development process example: Rational Unified Process (RUP)

- RUP is defined as a meta-process that expresses all meta elements of the process:
  - Role
  - Artifact
  - Activity
  - Discipline
  - Workflow



Software development process example: Rational Unified Process (RUP)

- The associations between Roles, Activities and Artifacts are well-defined in the process using *workflow diagrams*.
- Such workflow can then be used to enable control on the effective use of the process.
- For example, the *Testing Discipline*:

Roles Workflows

Roles vs. Artifacts

Roles vs. Activities

Adaptive software development models

Adaptive software development models: The Agile Manifesto

- Often also called “agile” methods.
- The “Agile Manifesto” (2001) was a statement against predictive methods.
- It proposed the following principles that are more realistic than what can be achieved by predictive methods in many software development projects:
  - Customer satisfaction by rapid delivery of useful software.
  - Welcome changing requirements, even late in the development.
  - Working software is delivered frequently (weeks rather than months).
  - Close, daily cooperation between business people and developers.
  - Projects are built around motivated individuals, who should be trusted.
  - Face-to-face conversation is the best form of communication (co-location).
  - Working software is the principal measure of progress.
  - Sustainable development, i.e. able to maintain a constant pace.
  - Continuous attention to technical excellence and good design.
  - Simplicity—the art of maximizing the amount of work not done—is essential.
  - Self-organizing teams.
  - Regular adaptation to changing circumstances.

Adaptive software development models: Concepts

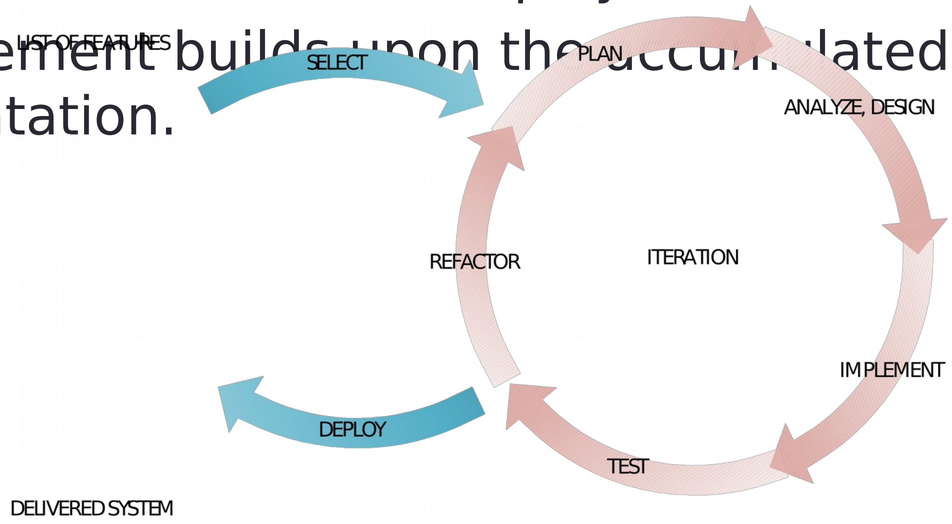
- Adaptive methods assume that software development is inherently about managing change, assuming that both the problem and the solution change during development:
  - The problem statement is refined and changes as the system is developed as the client sees the solution being developed.
  - The details of the designed solution are constantly changing during development.
- Most adaptive methods attempt to minimize risks and manage changes by developing software in short timeboxes, called **builds**, which typically last one to four weeks.
- Each build is like a miniature software project of its own, and includes all the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation.
- Capable of releasing new software at the end of every build.
- At the end of each build, the team re-evaluates project priorities.

Adaptive software development models: Concepts

- Adaptive methods emphasize real-time communication, preferably face-to-face, as opposed to communication using written documents, as documents accumulate unstable information that is costly to write, verify and change.
- Adaptive methods emphasize working software as the primary measure of progress.
- Adaptive methods produce very little written documentation relative to other methods.
- The only artifacts being produced are directly related to the efficient and sustainable production of implementation code.

Adaptive software development models: Incremental development

- Most adaptive methods develop software following an incremental model, where the software is produced in a series of “builds” that aim at the production of a solution for a small portion of the problem.
- **Incremental development** is a method of software development where the software is incrementally designed, implemented and tested until the product is finished.
  - During each increment, a set of features is selected for development, which are then analyzed, designed, implemented, tested, and deployed.
  - Each increment builds upon the accumulated system implementation.



Adaptive software development models: Coping with change

- One of the main advantages of the incremental models is their ability to **cope with change** during the development of the system.
- Predictive models rely on careful review of artifacts to avoid errors. Once a phase has been completed, there is limited provision for stepping back to fix/add something uncovered later.
- It is difficult to verify artifacts precisely and this is a weakness of the predictive models.
- As an example, consider an **error in the requirements**:
  - With the waterfall model, the error may not be noticed until acceptance testing, when it is probably too late to correct it.
  - The error may be a requirements error, but it is very tedious to verify requirements statements before they become operational, especially when buried in hundreds of other requirements statements.
  - The real problem of finding a requirements error at then end of the production phase is that a change in one requirement very often induces a “**ripple effect**” of changes in other requirements and to other following artifacts that are based on it (e.g design, code, tests, etc).

Adaptive software development models: Coping with change

- Thus, uncovering such a mistake toward then end of the production is likely to require many other changes. The uncovering of many of such mistakes at the end of the production leads to a dramatic situation that may put the whole project in jeopardy.
- On the other hand, in the incremental model, there is a good chance that a requirements error will be recognized as soon as the corresponding software is incorporated into the system and deployed.
- As software is developed then validated in short time boxes and for a reduced number of implemented features, errors uncovered are likely to have **lesser magnitude** in the ripple effect of changes that they induce.

Adaptive software development models: Distribution of feedback

- One of the main reasons why predictive models are not appropriate in many cases is the accumulation of unstable information at all stages.
  - For example, a list of 500 requirements is extremely likely to change, no matter how confident is the client on the quality of these requirements at this point.
- Inevitably, the following design and implementation phases will uncover flaws in these requirements, raising the need for the update and re-verification of the requirements and their subsequent artifacts each time a flaw is uncovered.
- A better approach is thus to limit the accumulation of unstable information by concentrating on the definition, implementation and validation of only a subset of the requirements at a time.
- Such an approach has the benefit of **distributing the feedback** on the quality of the accumulated information.
- In the Waterfall model, most of the relevant feedback is received at the end of the development cycle, where the programming and testing are concentrated. Such a model is evidently likely to lead to failure in later stages.
- By distributing the development and validation efforts throughout the development cycle, incremental models achieve distribution of feedback, thus increasing the **sustainability of further development**.

Adaptive software development models: Advantages

- Delivers an operational quality product at each stage, but one that satisfies only a subset of the clients requirements.
- A relatively small number of developers may be used.
- From the delivery of the first build, the client is able to perform useful work, providing early return on investment (ROI), an important economic factor.
- Reduces the traumatic effect of imposing a completely new product on the client organization by providing a gradual introduction.
- There is a working system at all times.
- Clients/users can see the system and provide feedback.
- Progress is concrete and visible, rather than being buried in abstract documents.
- Breaks down the problem into sub-problems, dealing with reduced complexity, and reducing the ripple effect of changes by reducing the scope to only a part of the problem at a time.
- Distributes feedback throughout the whole development cycle, leading to more stable artifacts and sustainable development and maintenance.

Adaptive software development models: Disadvantages

- Each additional build has somehow to be incorporated into the existing structure without degrading the quality of what has been built up to now.
- Addition of succeeding builds must be easy and straightforward.
- The more the succeeding builds are the source of unexpected problems, the more the existing structure has to be reorganized, leading to inefficiency and degrading internal quality and degrading maintainability.
- The incremental models can easily degenerate into the build and fix approach.
- Design errors become part of the system and are hard to remove.
- Clients see possibilities and want to change requirements.

Adaptive software development models: Dangers and Solutions

- **Planning**
  - The main danger of using incremental models is to proceed too much in an ad-hoc manner, i.e. without a global plan.
  - Initially determining a global plan of action is of prime importance to ensure the successful use of an incremental model of development.
  - The early stages of development must include a preliminary analysis phase that determines the scope of the project, tries to determine the highest risks in the project, define a more or less complete list of important features and constraints, in order to establish a **build plan**, i.e. a plan determining the nature of each build, and in what order the features are implemented.
  - Such a plan should be made in order to foresee upcoming issues in future builds, and develop the current build in light of these issues and make their eventual integration easier.

Adaptive software development models: Dangers and Solutions

- **Structural quality control**
  - The incremental model, like the build-and-fix model, is likely to result to the gradual degrading of internal structural quality of the software.
  - In order to minimize the potentially harmful effect of this on the project, certain quality control mechanisms have to be implemented, such as refactoring. Refactoring is about increasing the quality of the internal structure of the software without affecting its external behavior.
  - The net effect of a refactoring operation is to make the software more easy to understand and change, thus easing the implementation of the future builds, i.e. to achieve sustainability of development.
  - How often a refactoring operation needs to be done depends on the current structural quality degradation of the software.
  - Note that planning also has a similar effect by enabling to foresee further necessary changes and developing more flexible solutions in light of the knowledge of what needs to be done in the future.

Adaptive software development models: Dangers and Solutions

- **Architectural baseline**
  - One of the reasons for the degradation of internal structural quality of the system through increments is often associated with a lack of a well-defined overall architectural design.
  - Predictive methods advocate the early definition of the architecture of the system, or early identification and design of the system core.
  - Such a practice has the effect of easing the grafting of new parts on the system throughout increments, and minimizing the magnitude of changes to be applied upon grafting of new parts of the builds.
  - Achieving an architectural design is advisable when writing a project plan. The architecture can also help building a clear plan that developers can relate to.
  - Achieving an architectural design will help control the structural quality of the system by providing a framework for the entire application helping the developers to see the big picture of the system, as they are working on individual parts during de development of the different builds.
  - Also, refactoring operations normally have a result of conforming, or further defining or refining the architecture of the system.

Adaptive software development models: Dangers and Solutions

- **Parallel builds**
  - Various builds could be performed simultaneously by different teams.
    - For example, after the coding phase of build one is started, another team is already starting with the design the second build.
  - The risk is that the resulting builds will not fit together. Each build inevitably has some intersection with other builds.
  - Good coordination and communication is important to make sure that teams that have intersecting builds are agreeing on the nature and implementation of their common intersection.
  - The more builds are done concurrently, the more this problem is growing exponentially.
  - Also, larger number of software developers is necessary compared to linear incremental development.

Adaptive software development models: Applicability

- Adaptive development has been widely documented as working well for small (<10 developers) co-located teams.
- Adaptive development is particularly indicated for teams facing unpredictable or rapidly changing requirements.
- Adaptive development is less applicable in the following scenarios:
  - Large scale development efforts (>20 developers)
  - Distributed development efforts (non-co-located teams)
  - Mission- and life-critical efforts
  - Command-and-control company cultures
  - Low requirements change
  - Junior developers
- Agile home ground:
  - Low criticality
  - Senior developers
  - High requirements change
  - Small number of developers
  - Culture that thrives in chaos

References

- Craig Larman and Victor Basili. *Iterative and Incremental Development: A Brief History*. Computer 36 (6): 47–56. June 2003. doi:10.1109/MC.2003.1204375.
- Boehm, B. and Turner, R., *Balancing Agility and Discipline: A Guide for the Perplexed,* Addison-Wesley, Boston. 2004. ISBN-13: 978-0321186126
- Beck, et. al., *Manifesto for Agile Software Development*. http://agilemanifesto.org/
- Kenneth S. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Addison-Wesley Professional, 2012. ISBN-13: 978-0137043293
- Alistair Cockburn. *Agile Software Development.* Addison-Wesley, 2001. ISBN-13: 978-0201699692
- Alistair Cockburn. *Agile Software Development: The Cooperative Game*. Addison-Wesley, 2006. ISBN-13: 978-0321482754