COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

CHAPTER 4

EXTREME PROGRAMMING (XP)

M. MAHMUDUL HASAN
ASSISTANT PROFESSOR, CS, AIUB
http://www.dit.hua.gr/~m.hasan

EXTREME PROGRAMMING

- Evolved from the problems caused by the long development cycles of traditional development models (Beck 1999a).
- First started as 'simply an opportunity to get the job done' (Haungs 2001) with practices that had been found effective in software development processes during the preceding decades (Beck 1999b)

- Method is formed around common sense principles and simple to understand practices
  - No process fits every project, rather, simple practices should be tailored to suit an individual project

XP VALUES

- **Communication:** XP has a culture of oral communication and its practices are designed to encourage interaction.

  "Problems with projects can invariably be traced back to somebody not talking to somebody else about something important."

- **Simplicity:** Design the simplest product that meets the customer's needs. An important aspect of the value is to only design and code what is in the current requirements rather than to anticipate and plan for unstated requirements.

XP VALUES

- **Feedback:** The development team obtains feedback from the customers at the end of each iteration and external release. This feedback drives the next iteration.

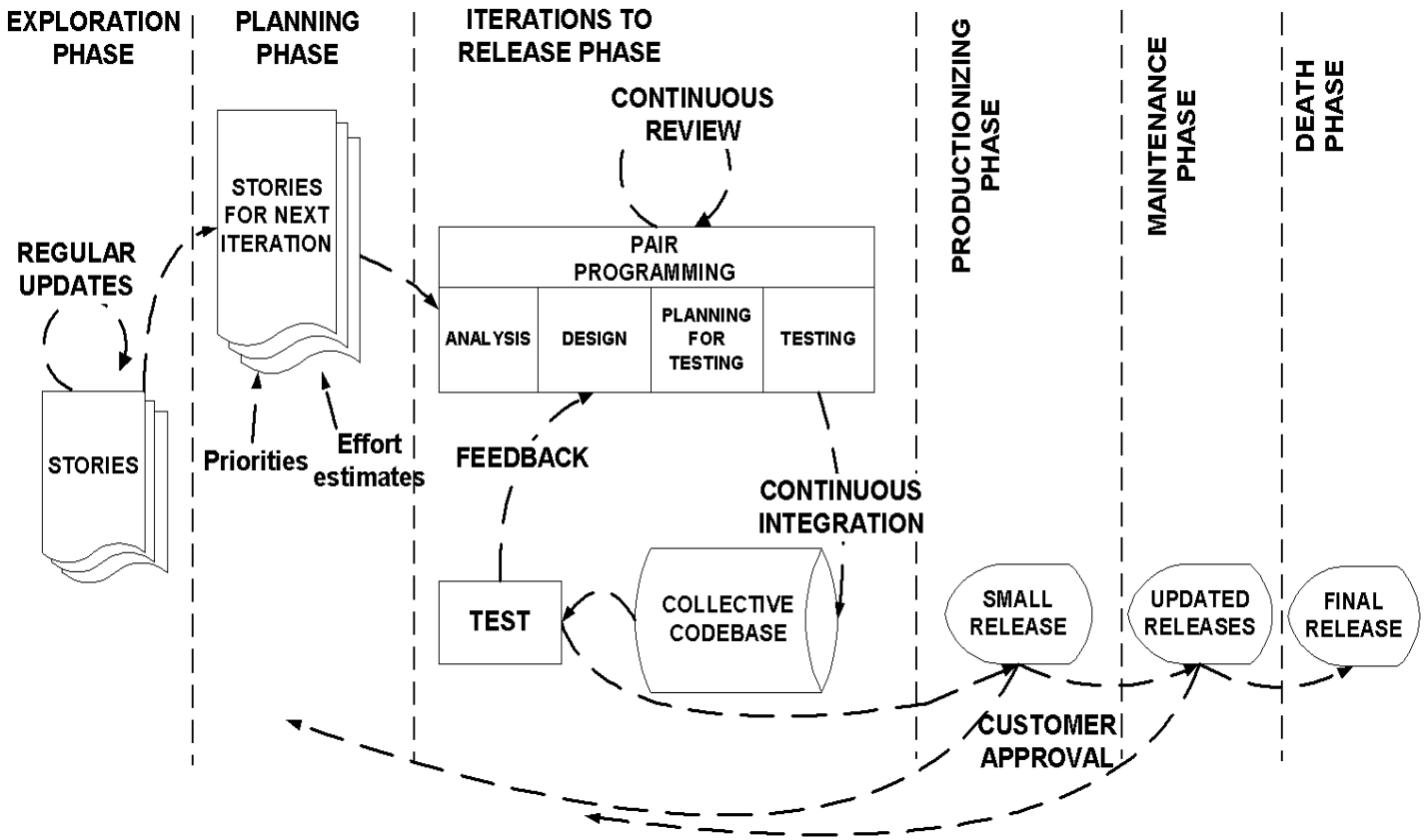
- **Courage**: Allow the team to have courage in its actions and decision making. For example, the development team might have the courage to resist pressure to make unrealistic commitments.

- **Respect:** Team members need to care about each other and about the project.

XP PROCESS

- The life cycle of XP consists of five phases:

1. Exploration
2. Planning
3. Iterations to Release
4. Productionizing
5. Maintenance and Death



XP PROCESS – EXPLORATION PHASE

- The customers write out the story cards that they wish to be included in the first release
- At the same time the project team familiarize themselves with the tools, technology and practices they will be using in the project
- The exploration phase takes between a few weeks to a few months, depending largely on how familiar the technology is to the programmers

XP PROCESS – PLANNING PHASE

- Users stories are written
- Estimate the effort of working with the user stories
- Priorities are given to the user stories to be implemented
- Release planning creates the release schedule

XP PROCESS – ITERATIONS TO RELEASE PHASE

- Includes several iterations of the systems before the first release
- Each take one to four weeks to implement
- The first iteration creates a system with the architecture of the whole system. This is achieved by selecting the stories that will enforce building the structure for the whole system
- The customer decides the stories to be selected for each iteration
- At the end of the last iteration the system is ready for production

XP PROCESS – PRODUCTIONIZING PHASE

- Requires extra testing and checking of the performance of the system before the system can be released to the customer
- New changes may still be found and the decision has to be made if they are included in the current release
- The iterations may need to be quickened from three weeks to one week
- The postponed ideas and suggestions are documented for later implementation

XP PROCESS – MAINTENANCE PHASE

- After the first release is productionized for customer use, the XP project must both keep the system in the production running while also producing new iterations
- Requires an effort also for customer support tasks
- Development velocity may decelerate after the system is in production
- May require incorporating new people into the team and changing the team structure

XP PROCESS – DEATH PHASE

- When the customer does no longer have any stories to be implemented
- System satisfies customer needs also in other respects (e.g., concerning performance and reliability)
- Necessary documentation of the system is finally written as no more changes to the architecture, design or code are made
- Death may also occur if the system is not delivering the desired outcomes, or if it becomes too expensive for further development

XP - ROLES AND RESPONSIBILITY

- **Customer:** writes the stories and functional tests, and decides when each requirement is satisfied. The customer sets the implementation priority for the requirements
- **Programmer:** keeps the program code as simple and definite as possible
- **Tester:** helps the customer write functional tests, also run functional tests regularly, broadcast test results and maintain testing tools

XP - ROLES AND RESPONSIBILITY

- **Tracker:** gives feedback in XP. He traces the estimates made by the team (e.g. effort estimates) and gives feedback on how accurate they are in order to improve future estimations. He also traces the progress of each iteration and evaluates whether the goal is reachable within the given resource and time constraints or if any changes are needed in the process
- **Coach:** Coach is the person responsible for the process as a whole. A sound understanding of XP is important in this role enabling the coach to guide the other team members in following the process
- **Consultant:** Consultant is an external member possessing the specific technical knowledge needed
- **Manager (Big Boss)**: Manager makes the decisions

XP - PRACTICES

- **Interaction:** Close interaction between the customer and the programmers. The programmers estimate the effort needed for the implementation of customer stories and the customer then decides about the scope and timing of releases.
- **Small/short releases:** A simple system is "productionized“ rapidly – at least once in every 2 to 3 months. New versions are then released even daily, but at least monthly.
- **Metaphor:** The system is defined by a metaphor/set of metaphors between the customer and the programmers. This "shared story" guides all development by describing how the system works.

XP - PRACTICES

- **Simple design:** The emphasis is on designing the simplest possible solution that is implementable at the moment
- **Testing:** Software development is test driven. Unit tests are implemented continuously
- **Refactoring:** Restructuring the system by removing duplication, improving communication, simplifying and adding flexibility
- **Collective ownership:** Anyone can change any part of the code at any time

XP - PRACTICES

- **Pair programming**
  - Two people write the code at one computer
  - One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical defects (syntactic, spelling, etc.) and also thinks strategically about the direction of the work.
  - Two programmers can brainstorm any challenging problem. Because they periodically switch roles.

- **Continuous integration:** A new piece of code is integrated into the code-base as soon as it is ready.

XP - PRACTICES

- **40-hour week:** A maximum of 40-hour working week
- **On-site customer:** Customer has to be present and available full-time for the team
- **Coding standards:** Coding rules exist and are followed by the programmers (e.g. error message by exception handling). Communication through the code should be emphasized
- **Open workspace:** A large room with small cubicle (compartment) is preferred
- **Just rules:** Team has its own rules that are followed, but can also be changed at any time. The changes have to be agreed upon and their impact has to be assessed

XP - ARTEFACTS

- **User story cards**
  - Paper index cards which contain brief requirement (features, non-functional) descriptions
  - Not a full requirement statement
  - Commitment for further conversation between the developer and the customer
  - During this conversation, the two parties will come to an oral understanding of what is needed for the requirement to be fulfilled
  - Customer priority and developer resource estimate are added to the card
  - The resource estimate for a user story must not exceed the iteration duration

XP - ARTEFACTS

- **Task list**
  - A listing of the tasks (one-half to three days in duration) for the user stories that are to be completed for an iteration
  - Tasks represent concrete aspects of a user story
  - Programmers volunteer for tasks rather than are assigned to tasks
- **CRC (Class-Responsibility-Collaboration) cards** (optional)
  - Paper index card on which one records the responsibilities and collaborators of classes which can serve as a basis for software design
  - The classes, responsibilities, and collaborators are identified during a design brainstorming/role-playing session involving multiple developers

XP - ARTEFACTS

Customer Acceptance Tests

- Textual descriptions and automated test cases which are developed by the customer
- The development team demonstrates the completion of a user story and the validation of customer requirements by passing these test cases.

Visible Wall Graphs

- To foster communication and accountability, progress graphs are usually posted in team work area. These progress graphs often involve how many stories are completed and/or how many acceptance test cases are passing.

XP - ARTEFACTS

Scope and use & Limitation

- XP is aimed at small to medium-sized teams
- Requires the development team to be located in one place. Scattering of programmers on two floors or even on one floor is intolerable for XP
- Communication and coordination between project members should be enabled at all times
- Require experienced team member through the project development time (XP is people based rather than plan based)

HOW XP SOLVES SOME SE PROBLEMS

Problem	**Solution**
Slipped schedule	Short development cycles
Cost of changes	Extensive, ongoing testing, system always running
Defect rates	Unit tests, customer tests
Misunderstand the business	Customer is a part of the team
Business changes	Changes are welcome
Staff turnover (replacement)	Intensive teamwork

REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, *33*(1), 182-214.

COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

CHAPTER 5

SCRUM

M. MAHMUDUL HASAN
ASSISTANT PROFESSOR, CS, AIUB
http://www.dit.hua.gr/~m.hasan

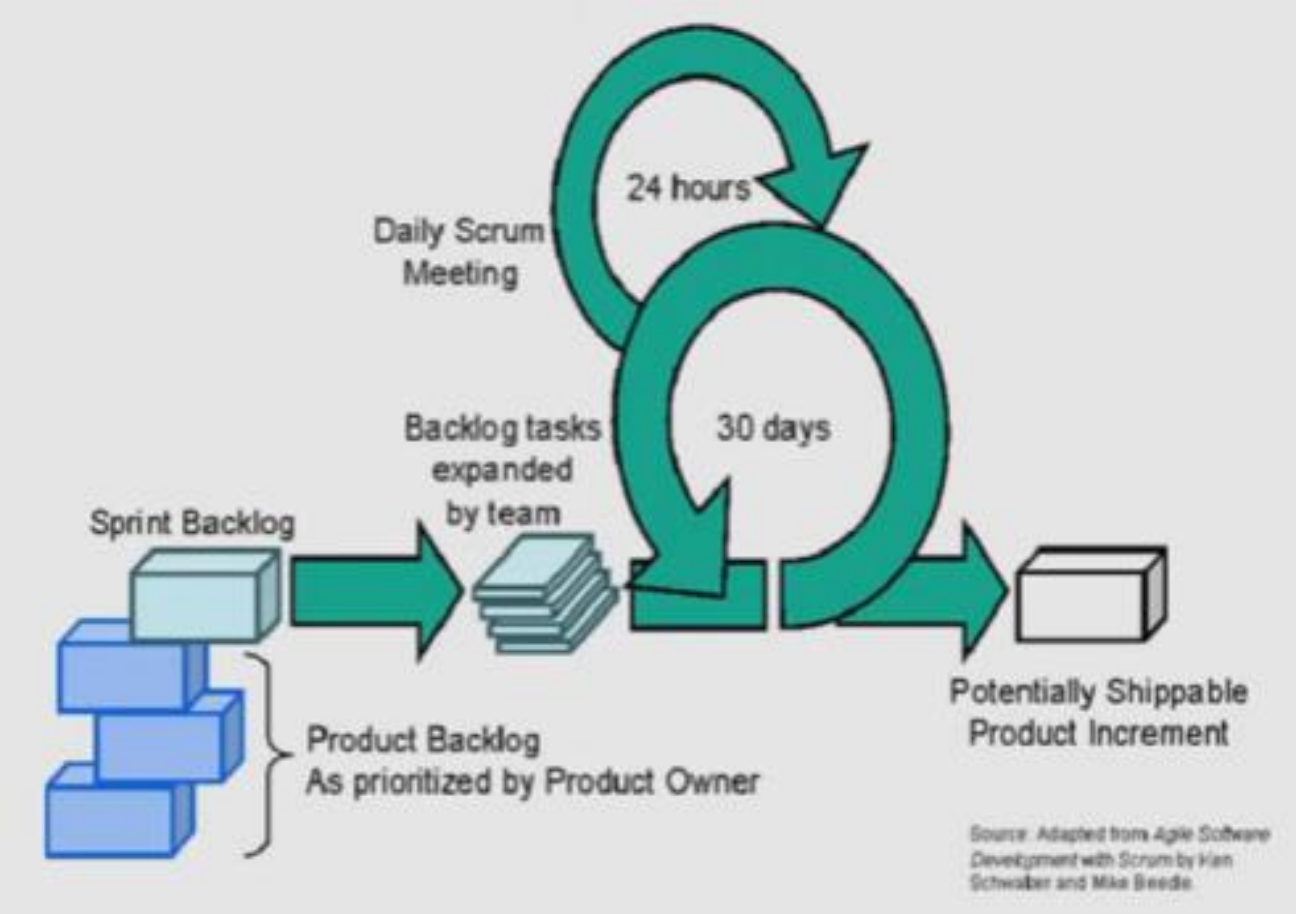
SCRUM

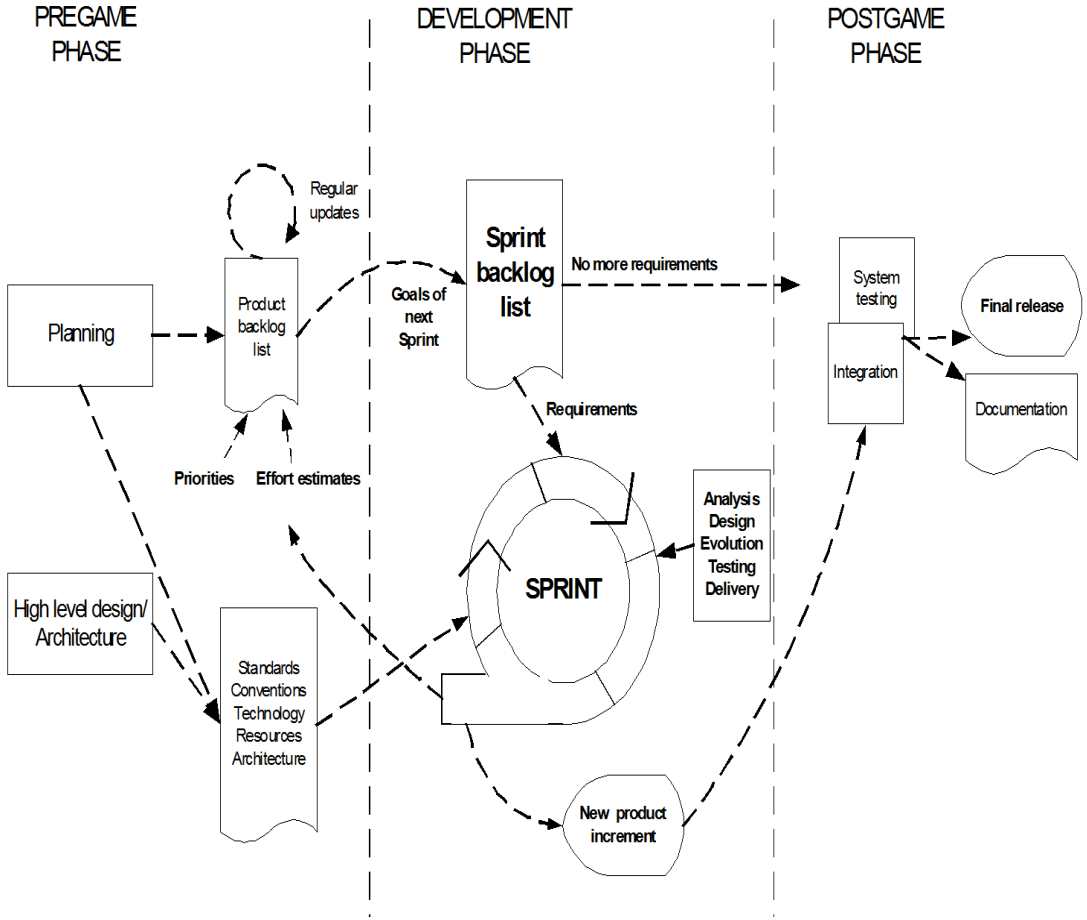
- The first references in the literature to the term 'Scrum‘ point to the article of Takeuchi and Nonaka (1986) in which an adaptive, quick, self-organizing product development process originating from Japan is presented (Schwaber and Beedle 2002).

- The term 'scrum' originally derives from a strategy in the game of rugby where it denotes "getting an out-of play ball back into the game" with teamwork (Schwaber and Beedle 2002).

- SCRUM process includes three phases
  - Pre-game
  - Development (game phase)
  - Post-game

SCRUM PROCESS





PRE-GAME PHASE

- The pre-game phase includes two sub-phases:

1. Planning:

- Definition of the system being developed
- A Product Backlog list is created containing all the requirements that are currently known
- The requirements are prioritized and the effort needed for their implementation is estimated
- The product Backlog list is constantly updated with new and more detailed items, as well as with more accurate estimations and new priority orders
- Planning also includes the definition of the project team, tools and other resources, risk assessment and controlling issues, training needs and verification management approval

PRE-GAME PHASE

2. Architecture

- The high level design of the system including the architecture is planned based on the current items in the Product Backlog
- In case of an enhancement to an existing system, the changes needed for implementing the Backlog items are identified along with the problems they may cause
- A design review meeting is held to go over the proposals for the implementation and decisions are made on the basis of this review

DEVELOPMENT (GAME) PHASE

- This phase is treated as a "black box" where the unpredictable is expected
- The system is developed in Sprints
  - Sprints are iterative cycles where the functionality is developed or enhanced to produce new increments.
  - Each Sprint includes the traditional phases of software development: requirements, analysis, design, evolution and delivery phases.
  - One Sprint is planned to last from one week to one month.

POST-GAME PHASE

- This phase is entered when an **agreement** has been made such as the requirements are completed.
- In this case, **no more items and issues** can be found nor can any new ones be invented.
- The system is now ready for the **release** and the preparation for this is done during the post-game phase, including the tasks such as the integration, system testing and documentation.

ROLES AND RESPONSIBILITIES

Scrum Master

- Scrum Master is responsible for ensuring that the project is carried through according to the practices, values, and rules of Scrum and that it progresses as planned.
- Scrum Master interacts with the project team as well as with the customer and the management during the project.

Product Owner

- Product Owner is officially responsible for the project, managing, controlling, and making visible the Product Backlog list.
- He is selected by the Scrum Master, the customer, and the management.
- He makes the final decisions of the tasks related to product Backlog.

ROLES AND RESPONSIBILITIES

- **Scrum Team**
  - Scrum Team is the project team that has the authority to decide on the necessary actions and to organize itself in order to achieve the goals of each Sprint.
  - The scrum team is involved, for example, in effort estimation, creating the Sprint Backlog, reviewing the product Backlog list and suggesting impediments that need to be removed from the project.
- **Customer**
  - Customer participates in the tasks related to product Backlog items for the system being developed or enhanced.
- **Management**
  - Management is in charge of final decision making, along with the agreements, standards, and conventions to be followed in the project.
  - Management also participates in the setting of goals and requirements.

SCRUM PRACTICES

Product Backlog

- Defines the work to be done in the project
- A prioritized and constantly updated list of business and technical requirements for the system being built or enhanced
- Include features, functions, defects, bug fixes, requested enhancements and technology upgrades

Effort Estimation

- Effort estimation is an iterative process, in which the Backlog item estimates are focused on a more accurate level when more information is available on a certain Product Backlog item.
- The Product Owner together with the Scrum Team(s) are responsible for performing the effort estimation.

SCRUM PRACTICES

Sprint

- Sprint is the procedure of adapting to the changing environmental variables (requirements, time, resources, knowledge, technology etc.).
- The working tools of the team are Sprint Planning Meetings, Sprint Backlog, and Daily Scrum meetings.

Sprint Backlog

- Sprint Backlog is the starting point for each Sprint. It is a list of Product Backlog items selected to be implemented in the next Sprint.
- The items are selected by the Scrum Team together with the Scrum Master and the Product Owner in the Sprint Planning meeting, on the basis of the prioritized items and goals set for the Sprint.
- Unlike the Product Backlog, the Sprint Backlog is stable until the Sprint (i.e. 30 days) is completed. When all the items in the Sprint Backlog are completed, a new iteration of the system is delivered.

SCRUM PRACTICES

- **Sprint Planning meeting**
  - A Sprint Planning Meeting is a two-phase meeting organized by the Scrum Master.
  - The Scrum Master, Management, Product Owner, and Scrum Team participate in the first phase of the meeting to decide upon the goals and the functionality of the next Sprint.
  - The second phase of the meeting is held by the Scrum Master and the Scrum Team focusing on how the product increment is implemented during the Sprint.

SCRUM PRACTICES

Daily Scrum meeting

- Daily Scrum meetings are organized to keep track of the **progress** of the Scrum Team continuously and they also serve as planning meetings: what has been done since the last meeting and what is to be done before the next one.
- Also problems and other variable matters are discussed and controlled in this short (approximately 15 minutes) meeting held daily. Any deficiencies or impediments in the systems development process or engineering practices are looked for, identified and removed to improve the process. The Scrum Master conducts the Scrum meetings. Besides the Scrum team also the management, for example, can participate in the meeting.

SCRUM PRACTICES

- **Sprint Review meeting**

- On the **last day** of the Sprint, the Scrum Team and the Scrum Master present the results (i.e. working product increment) of the Sprint to the management, customers, users, and the Product Owner in an informal meeting.
- The participants assess the product increment and make the decision about the following activities.
- The review meeting may bring out new Backlog items and even change the direction of the system being built.

REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, *33*(1), 182-214.

COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

CHAPTER 6

THE DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM)

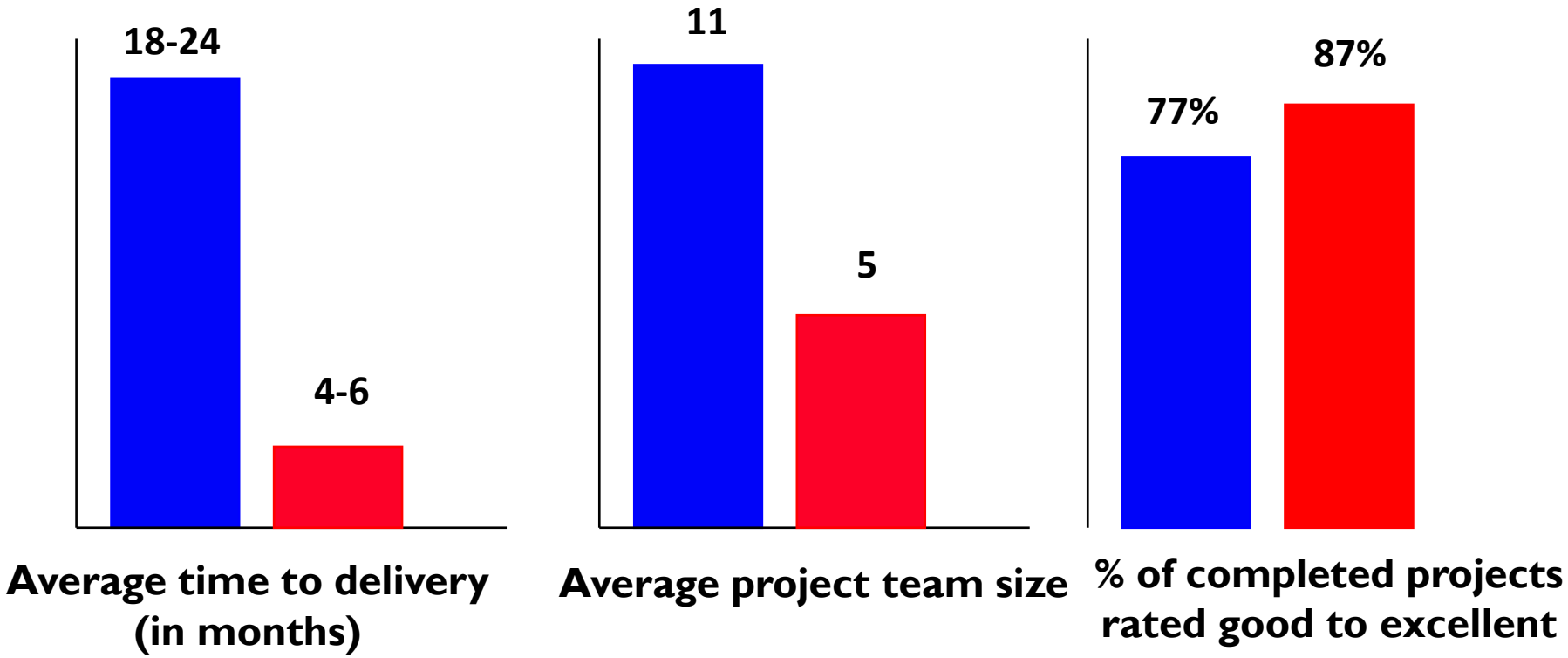
M. MAHMUDUL HASAN
ASSISTANT PROFESSOR, CS, AIUB
http://www.dit.hua.gr/~m.hasan

DSDM

- **The Dynamic Systems Development Method (DSDM)** is a public domain Rapid Application Development method which has been developed through capturing the experience of a large group of vendor and user organisations. It is now considered to be the UK's de-facto standard for RAD.

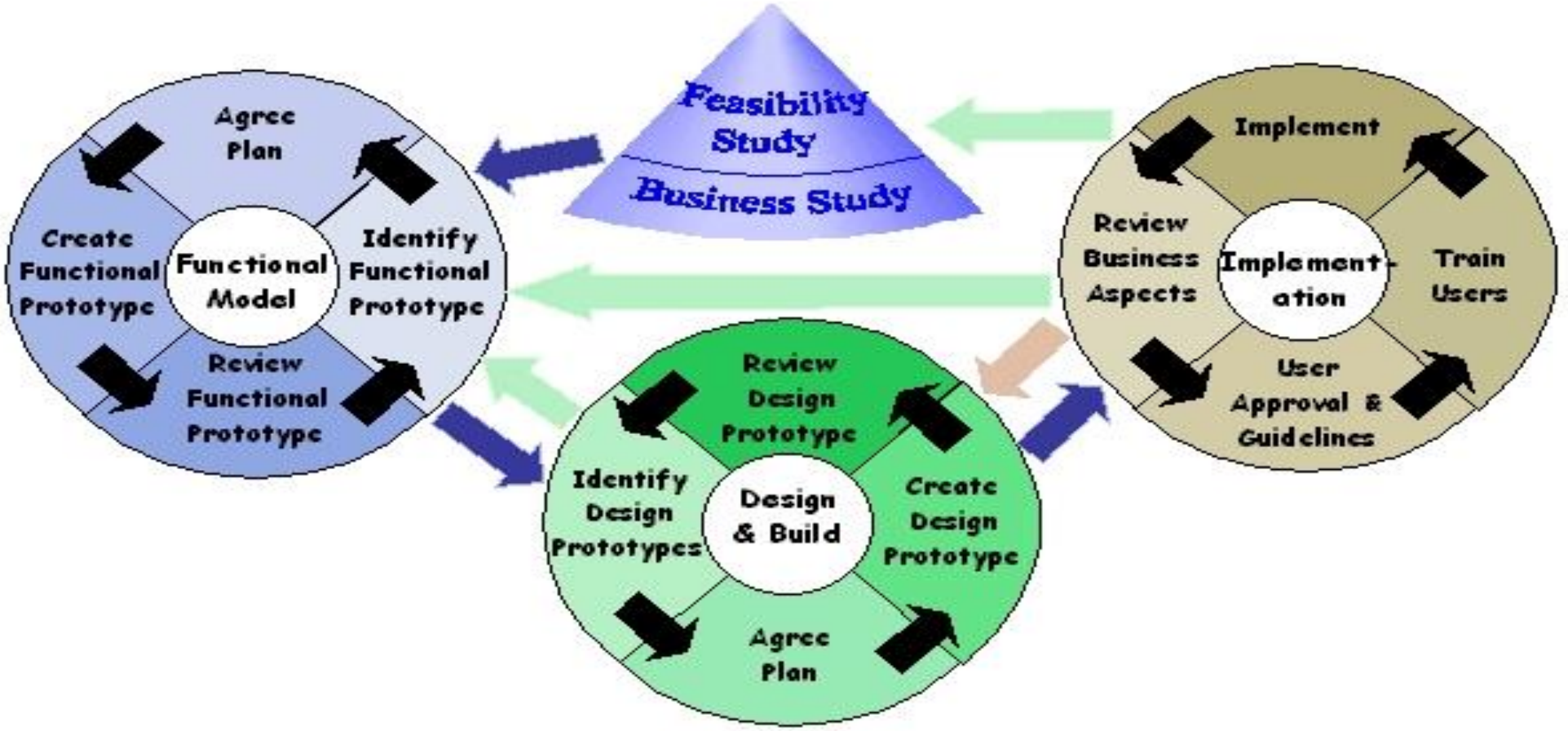
- The key to DSDM is to deliver **what** business needs **when** it needs
  - Achieved by using the various techniques in the framework and flexing requirements
  - The aim is always to address the current and imminent needs of the business rather than to attack all the perceived possibilities

TRADITIONAL METHOD VS. DSDM



Source: British Airways IM Department, Newcastle

DSDM PROCESS VIEW



DSDM PROCESS

Activity	Sub activity	Description
Study	**Feasibility Study**	Stage where the suitability of DSDM is assessed. Judging by the type of project, organizational and people issues, the decision is made, whether to use DSDM or not. Therefore it will generate a FEASIBILITY REPORT, a FEASIBILITY PROTOTYPE, and a GLOBAL OUTLINE PLAN which includes a DEVELOPMENT PLAN and a RISK LOG.
	Business Study	Stage where the essential characteristics of business and technology are analyzed. Approach to organize workshops, where a sufficient number of the customer's experts are gathered to be able to consider all relevant facts of the system, and to be able to agree on development priorities. In this stage, a PRIORITIZED REQUIREMENTS LIST, a BUSINESS AREA DEFINITION, a SYSTEM ARCHITECTURE DEFINITION, and an OUTLINE PROTOTYPING PLAN are developed.

DSDM PROCESS

Activity	Sub activity	Description
Functional Model Iteration	Identify functional prototype	Determine the functionalities to be implemented in the prototype that results from this iteration. In this sub-stage, a FUNCTIONAL MODEL is developed according to the deliverables result of business study stage.
	Agree schedule	Agree on how and when to develop these functionalities.
	Create functional prototype	Develop the FUNCTIONAL PROTOTYPE, according to the agreed schedule and FUNCTIONAL MODEL.
	Review functional prototype	Check correctness of the developed prototype. This can be done via testing by end-user and/or reviewing documentation. The deliverable is a FUNCTIONAL PROTOTYPING REVIEW DOCUMENT.

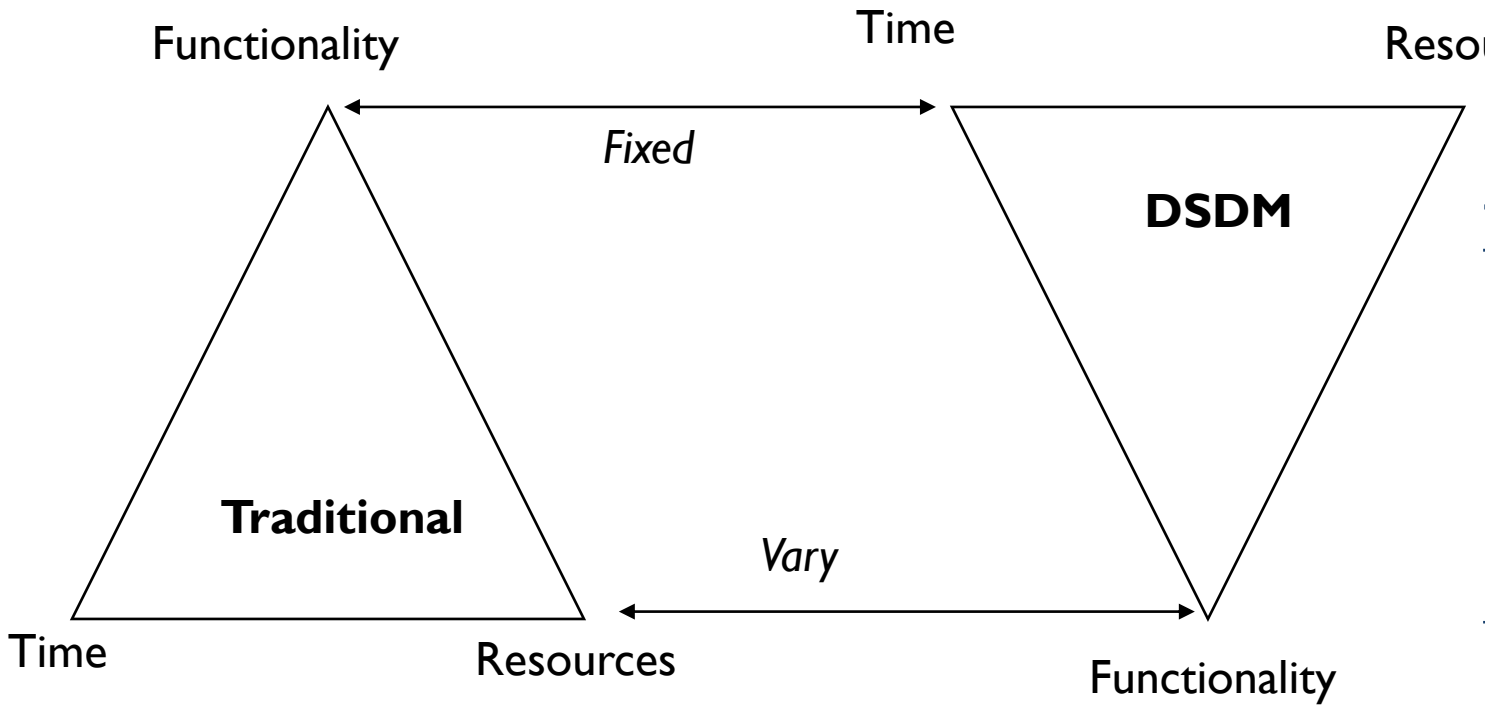
DSDM PROCESS

Activity	Sub activity	Description
Design and Build Iteration	Identify design prototype	Identify functional and **non-functiona**l requirements that need to be in the tested system. And based on these identifications, an IMPLEMENTATION STRATEGY is involved. If there is a TEST RECORD from the previous iteration, then it will be also used to determine the IMPLEMENTATION STRATEGY.
	Agree schedule	Agree on how and when to realize these requirements.
	Create design prototype	Create a system (DESIGN PROTOTYPE) that can safely be handed to end-users for daily use, also for testing purposes.
	Review design prototype	Check the correctness of the designed system. Again testing and reviewing are the main techniques used. An USER DOCUMENTATION and a TEST RECORD will be developed.

DSDM PROCESS

Activity	Sub activity	Description
Implementation	User approval and guidelines	End users approve the tested system (APPROVAL) for implementation and guidelines with respect to the implementation and use of the system are created.
	Train users	Train future end user in the use of the system. TRAINED USER POPULATION is the deliverable of this sub-stage.
	Implement	Implement the tested system at the location of the end users, called as DELIVERED SYSTEM.
	Review business	Review the impact of the implemented system on the business, a central issue will be whether the system meets the goals set at the beginning of the project. Depending on this the project goes to the next stage, the post-project or loops back to one of the preceding stages for further development. This review is will be documented in a PROJECT REVIEW DOCUMENT.

DIFFERENCE BETWEEN TRADITIONAL DEVELOPMENT VS. DSDM



Traditional Method

- Functional/requirements are fixed
- Time & resources can varies

DSDM/Agile Methods

- Functional/requirement varies
- Time & resources are fixed

TECHNIQUES TO CONSIDER IN DSDM

- Flexibility
- Timeboxing
- MoSCoW Rules
- Prototyping
- Facilitated Workshops

DSDM TECHNIQUES: FLEXIBILITY

- A fundamental assumption of DSDM is that nothing is built perfectly first time
- 80:20 Rule: assumes that a usable and useful 80% of the proposed system can be produced in 20% of the time it would take to produce the total system.
- In "traditional" development practice, a lot of time is spent in getting from the 80% solution to the total solution, with the assumption that no step ever needs to be revisited. The result is either projects that are delivered late and over budget or projects that fail to meet the business needs since time is not spent reworking the requirements.
- DSDM assumes that all previous steps can be revisited as part of its iterative approach. Therefore, the current step need be completed only enough to move to the next step, since it can be finished in a later iteration.

DSDM TECHNIQUES: TIMEBOXING

- Without effective timeboxing, prototyping teams can lose their focus and run out of control.
- Timeboxing works by concentrating on when a business objective will be met as opposed to the tasks which contribute to its delivery.
- **Timeboxing Basics**
  - Time between start and end of an activity
  - DSDM uses nested timeboxes, giving a series of fixed deadlines
  - Ideally 2 - 4 weeks in length
  - Objective is to have easiest 80% produced in each timebox
  - Remaining 20% potentially carried forward subsequent timeboxes
  - Focus on the essentials
  - Helps in estimating and providing resources

DSDM TECHNIQUES: MOSCOW RULES

- **MoSCoW** rules formalised in DSDM version 3

Must have – fundamental to project success

Should have – important but project does not rely on

Could have – left out without impacting on project

Want to have but Won't have this time for those valuable requirements that can wait till later development takes place; in other words, the Waiting List.

DSDM TECHNIQUES: PROTOTYPING

Prototypes are necessary in DSDM because

- Facilitated workshops define the high-level requirements and strategy
- Prototypes provide the mechanism through which users can ensure that the detail of the requirements is correct
- Demonstration of a prototype broadens the users‘ awareness of the possibilities and assists them in giving feedback to the developers
- Speeds up the development process and increases confidence that the right solution will be delivered

DSDM TECHNIQUES: FACILITATED WORKSHOPS

- Purpose is to produce clear outcomes that have been reached by consensus

- **Participants**
  - Workshop sponsor
  - Participants (development team)
  - Scribes (record)
  - Observers
  - Prototypers
  - Facilitator (help a group of people understand their common objectives and assists them to plan how to achieve these objectives)

- **Advantages of Workshops**
  - Speed
  - Involvement /ownership
  - Productivity
  - Consensus
  - Quality of decisions
  - Overall perspective / synergy (cooperation)

REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, *33*(1), 182-214.

COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

CHAPTER 7

FEATURE DRIVEN DEVELOPMENT (FDD)

M. MAHMUDUL HASAN
ASSISTANT PROFESSOR, CS, AIUB
http://www.dit.hua.gr/~m.hasan

HISTORY OF FDD

- Original Creator: Jeff De Luca
  - Singapore in late 1997
- FDD evolved from an actual project
  - Bank Loan Automation
  - Luca was Project manager
  - 50 member developer team

WHAT IS FDD?

- Feature Driven Development (FDD)
- FDD is an agile software development process
- FDD uses a short-iteration model
- FDD combines key advantages of other popular agile approaches along with other industry-recognized best practices
- FDD was created to easily scale to much larger projects and teams

WHAT IS A FEATURE?

- FDD delivers the system feature by feature
- Feature is a small function expressed in client-valued terms which presents the customer requirements to be developed in software using small iteration
- Features are to be "small" in the sense they will take no more than two weeks to complete Features that appear to take longer are to be broken up into a set of smaller features. Two weeks is the maximum, most features take less time (1 - 5 days)
- Feature naming template:

  <action> the <result> <by|for|of|to> a(n) <object>

- Examples: Calculate the total of a sale
  Validate the password of a user
  Authorize the sales transaction of a customer

CLASS OWNERSHIP

- Class (feature) assigned to specific developer
- Class owner responsible for all changes in implementing new features
- Collective Ownership
  - Any developer can modify any artifact at any time
- Advantages of Class Ownership are:
  - Someone responsible for integrity of each class
  - Each class will have an expert available
  - Class owners can make changes much quicker
  - Easily lends to notion of code ownership (XP)

FDD ROLES

- FDD Primary Roles

Project Manager	Chief Architect
Class Owners	Domain Experts
	Chief Programmers

- FDD Supporting Roles

Language Guru (shared vocabulary)	
Toolsmith (making tools for application)	
Tester	
Technical Writer (documentation)	

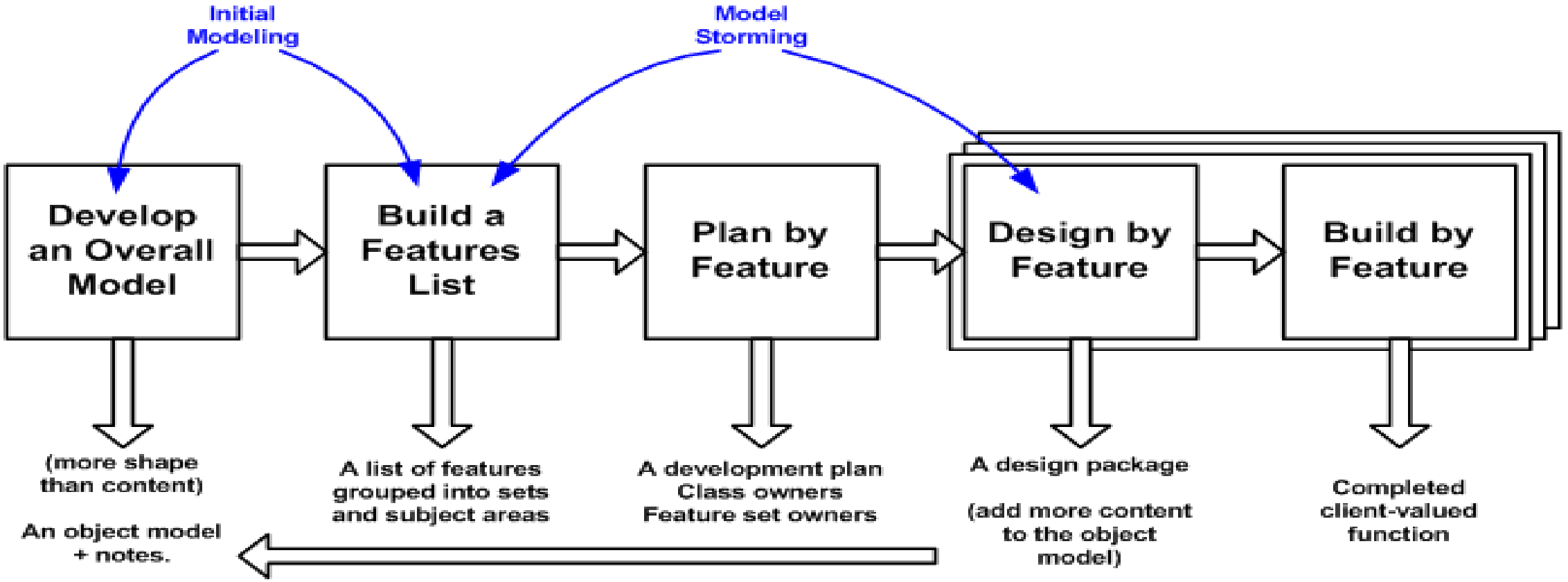
FDD PROCESS

- Process #1: Develop an Overall Model
- Process #2: Build a Features List
- Process #3: Plan By Feature
- Process #4: Design By Feature
- Process #5: Build By Feature

FDD PROCESS

- Project wide upfront design activities:
  - Process #1: Develop an Overall Model
  - Process #2: Build a Features List
  - Process #3: Plan By Feature
  - Goal: not to design the system in its entirety but instead is to do just enough initial design that you are able to build on
- Deliver the system feature by feature:
  - Process #4: Design By Feature
  - Process #5: Build By Feature
  - Goal: Deliver real, completed, client-valued function as often as possible

FDD PROCESS



FDD PROCESS

- **Process #1: Develop an Overall Model**
  - Form a modeling team
  - Domain walk-through
  - Build High-level object model
  - Record Notes
  - **Goal -** for team members to gain a good, shared understanding of the problem domain and build a foundation

- **Process #2: Build a Features List**
  - All Features are organized in a three level hierarchy :

Domain Subject Area
Business Activity
Features

FDD PROCESS

- **Process #3: Plan by Feature**
  - Construct initial schedule
    - Formed on level of individual features
      - Prioritize by business value
      - Also consider dependencies, difficulty, and risks
  - Assign responsibilities to team members
    - Determine Class Owners
    - Assign feature sets to chief programmers

FDD PROCESS

- **Process #4: Design by Feature**
  - Form Feature Teams
  - Team members collaborate on the full low level analysis and design
  - Certain features may require teams to bring in domain experts
  - Teams need to update the model artifact to support their changes

Feature Team

- Chief Programmers pick teams based on the current feature in development
- Chief Programmers lead picked team (usually 3 to 5 people)
- Upon completion of the current feature the team disbands
- Each team will concurrently work on their own independent iteration
- Possible to be on multiple teams at once

FDD PROCESS

- **Process #5: Build by Feature**
  - Implement designed feature
  - Test feature
    - Unit-level
    - Feature-level
  - Mandated Code Inspections (formal review with checklist)
  - Integrate with regular build

FDD PROCESS

- **Mandated Code Inspections** for Two Main Reasons
  - Research has shown that when it is done properly, inspections find more bugs as well as different types of bugs than any other form of testing.
  - It is also a great learning experience

- **Reporting**
  - FDD emphasizes the ability to provide accurate, meaningful, and timely progress information to all stakeholders within and outside the project
  - Feature Milestones

REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, *33*(1), 182-214.