FACTFILE: GCE SOFTWARE SYSTEMS DEVELOPMENT

UNIT A2 1: SYSTEMS METHODOLOGIES

Rewarding Learning

A2.1 Systems Methodologies

Their Nature and Purpose

Building a software system can be a long and complicated process with many interdependent tasks and large teams to manage. This process is helped by using a **software methodology** which…

- describes the formal approach that will be used to manage (plan, monitor and control) the **development** of a new software system;
- provides a framework/recipe/plan incorporating knowledge and wisdom gleamed from years of software development;
- splits the software development process into a set of **stages/phases**;
- these stages are followed in a specified order through a development **lifecycle**;
- details the **activities** to complete within each stage supported with guidelines, principles, tools, techniques and documentation;
- helps to improve the **quality** of the software system and the software system development process producing a software system that is more likely to meet or exceed customer expectations and be delivered within time and budget;

A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. They vary widely in philosophy with some being quite specific about the documentation produced, in their coverage of the project lifecycle and in the kind of software projects they are best suited to.

Evolution of the Methodologies

In the 1960's and early 1970's attempts to build large and complex systems were discouraging as typically they were delivered late, over budget, were unreliable and difficult to maintain. This period of time was known as the software crisis. To address these difficulties a structured approach called the Waterfall was developed.

Waterfall Model

The Waterfall model (now called the **traditional** approach) evolved from the construction industry and was applied to the development of software. The phases[1] of the software development lifecycle are followed in a **sequential** fashion from top to bottom as in a waterfall. No phase was allowed to start until the previous phase was complete. Certain roles (e.g. Customer, Business Analyst, Developer) are only involved during certain activities.

The main advantages are:

- **It is simple and easy to understand:** this is a big advantage for inexperienced developers in the team as the required activities at each stage are well defined;
- **Project managers found it easy to manage:** there are a clear sequence of activities in each stage to follow allowing them to easily draw their Gantt chart and identify milestones for the end of each stage; it was then easier for them to cost and control.

But the disadvantages are many and include:

- **Cannot accommodate changing requirements:** It is not suitable for projects where requirements are at a moderate to high

[1] Note that there are many different versions of the waterfall model. Some use different names for the phases, for example using implementation for installation. Some add extra stages such feasibility and maintenance to show the full system lifecycle.

risk of changing and where they are incomplete or vague. This model is used only when the requirements are very well known, clear and fixed which is more likely in small short projects. It is not good for long and ongoing projects or complex and object-oriented projects.

- **Limited customer involvement:** Customers are only involved in the early stages to define requirements and at the end to test the finished product which increases the risk that the product may not meet the customer's needs.
- **Forward direction of flow only:** Can only go forward through the phases so there is no feedback to guide previous phases.
- **Project management is very authoritarian in style:** this does not help support the creativity of developers or use their skills in the best way.
- **Testing performed at end of lifecycle:** this often leads to errors discovered later in the lifecycle which may involve new design and cost more to change at this stage than if they were discovered at an earlier stage.
- **Working software is delivered late in the lifecycle:** Customers will not know whether the right product is being built and it is hard to gauge progress.
- **Integration is performed at end in a 'Big Bang':** which is risky as design problems may only be uncovered at a late stage which may cost a lot to fix.

A new approach emerged in the **1990s** in response to these problems called Rapid Application Development.

Rapid Application Development (RAD)

RAD[2] **compresses** several steps of the waterfall process into an **iterative** process using prototyping[3] with **high user participation**. The development team delivers a series of fully functional prototypes to the users who evaluate the prototype requesting changes/further refinement. The developers rework the prototype and the process continues in a cycle until the software product **evolves** into the final working product satisfying the customers.

Several prototypes may be developed in parallel for different parts of the system. RAD offers the following advantages:

- **Less up-front planning:** Prototyping is used in place of detailed planning and this helps as often a customer finds it difficult to express their needs/requirements unless they can actually see a working model of the system.
- **Shortens the lifecycle** and enabling **rapid delivery** by speeding up design and implementation. It does this by using special techniques and tools such as:
  - Computer-Aided Software Engineering **(CASE)** tools including **code generators** which enable the automatic generation of programs and database code directly from design documents, diagrams, forms, and reports. There is less manual coding.
  - Joint application design **(JAD)** sessions: **Users, Managers and Analysts work together for several days** in intensive workgroup sessions to specify or review system requirements.
  - **Fourth generation/visual programming languages** such as Visual Basic.
- The construction can be **broken down** into several compartments/modules for which separate prototypes can be developed in parallel.
- Prototypes **facilitate customer requirements determination** as they can visually see part of the working system which is good if the system is covering a new unfamiliar area of the business where the customer finds it hard to express their requirements in a written form.
- **Changing requirements** can be easily accommodated in the evolving prototypes and the solution is more likely to meet the needs of the customer as problems can be discovered earlier in the process.
- **Rapid delivery** to the customer with **high levels of customer involvement** during the complete development cycle reduces the risk of not meeting the customer's requirements. Some business environments are so rapidly changing that by the time the solution is delivered it could be out of date already. High customer involvement also encourages the acceptance of the new system as they are more familiar with it.
- **Progress can be easily seen** through the

[2] The James Martin version of the lifecycle includes four phases called requirements planning, user design, construction and cutover.

[3] The type of prototyping used in RAD is evolutionary as the model develops into the final system but there are many types of prototyping including throw-away in which the prototype is eventually discarded after use.

production of working prototypes not just at the end of the lifecycle, as in the waterfall model. This reassures the customer.

- **There is no big bang integration at end.** There is continuous integration of code. This reduces the risk of finding out that the architectural design is wrong at the end of the project. The design can be restructured at an early stage if necessary when the cost of change is less.

But the disadvantages include:

- **Less management control:** Managing larger projects is very difficult due to lack of planning.
- **Demands more frequent user interaction** throughout project lifecycle and key business people may find it difficult to make that commitment as their time is valuable to the business.
- **Requires highly skilled developers and designers:** they must have wide experience and a large skill set to rapidly respond and adapt the prototype to customer feedback – no time for training courses!
- **Cost of CASE tools** for modelling and automated code generation is high making it less suitable for cheaper projects.
- **Poor design:** the focus on visual prototypes may result in poor architectural design. There may also be problems with programming standards, documentation and maintenance.
- **Lack of scalability.** RAD typically focuses on small to medium-sized project teams.

RAD is popular for web and e-commerce systems which are developed in rapidly changing business environments.

Agile Methodologies

Agile methodologies were formally introduced in 2001 when the Agile Manifesto was published.

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others to 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 a 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.

This manifesto emphasised the difference between agile and traditional methodologies such as the Waterfall model. Agile methodologies are strongly influenced by **RAD** and **prototyping** but are more clearly defined with their own tools and techniques. They can be used for Object Orientated software development projects.

Agile methodologies follow the Software Development lifecycle phases of Analysis, Design, Implementation, Testing and Installation found in the Waterfall model but these are performed in an **iterative and incremental** manner.

- Incremental means that the system is broken down into several smaller parts called increments. Each of these increments works through the software development lifecycle and will be released to the customer as a working subset of the system, frequently.
    - These frequent incremental releases allow the customer to use part of the system at an early stage of development. This allows changes to be suggested and progress to be monitored throughout development reducing the uncertainty and risk of non-delivery. Requirements are implemented in a prioritised fashion enabling the highest business value to be delivered first and to deliver on time.
    - This is in contrast to the waterfall which delivers the whole system at the end at once when change is difficult and expensive.
- Iterative means that within each development phase of an increment there is an iterative cycle. This starts with a vague understanding of the system but as the development progresses this understanding evolves and becomes clearer. So for example we might start with a set of very high level requirements which are quite broad and general. We then produce prototypes (performing analysis, design, code

and testing) for the customer to evaluate and using their feedback we can produce a more refined prototype as we have a more detailed and clearer understanding of their requirements.

- These iterative cycles enable agile projects to respond to rapidly changing customer requirements as there are frequent opportunities for early feedback.

It may be more difficult to estimate how long each stage will take when the system is in the early stages of development so **planning is more difficult**, especially in large projects, but outline plans will evolve and change as the system develops.

- In contrast the waterfall model requires that requirements are defined up front early in the project – they assume that we have all the necessary information to create detailed plans. This may be difficult in certain types of project which use novel technologies or for new business areas with uncertain requirements and where the customer has little understanding of their needs without further exploration.

Agile methodologies are both incremental AND iterative.

The project is broken down into several parts at different levels.

- A project is broken down into **increments** (also called releases).
- Each increment is broken down into **iterations** (also called timeboxes or sprints depending on the methodology).
- Each iteration is broken down into **tasks.**

At each level of breakdown there is feedback in an iterative cycle for example:

- at the end of each increment/release there is feedback from the customer who uses part of the system in a live environment - which helps to plan work for the next increment.
- at the end of each iteration there is a review of the work (sprint review and retrospective in SCRUM) which helps to plan the work in the next iteration.
- at the end of a daily task there is a stand-up meeting where feedback is given which steers the work for the next day.

Within each iteration entire features are developed and **analysis, design, code and test are ALL performed.** These iterations are performed in timeboxes in DSDM, sprints in XP and simply iterations in XP.

- **testing is performed throughout the project** in contrast to the Waterfall model when it is only performed at the end of the lifecycle where defects are more difficult and costly to resolve.

Other differences between agile and traditional methodologies include:

- **Documentation is minimal and evolves** (documents may be said to be ‘living’). Project teams spend more time on development and less on documentation. This is especially noticeable in the XP agile methodology. In contrast in the waterfall model the documentation is extensive and heavyweight.
- Continuous customer/business **involvement** from an early stage helps to **rapidly** and **flexibly** respond to **changing** and **evolving** requirements increasing satisfaction and quality. In contrast the Waterfall model only involves the user at the beginning and end of the project which means they have little opportunity to change the development of the system; this does mean that scope creep is minimised but the product may not be what the customer and the business really needs.
- Teams are encouraged to be **self-directing, pro-active** taking initiatives, be **co-operative** and **collaborative, self-organising** and **cross-functional**. Project managers should be respectful, facilitative, supportive and empowering. Teams will then be more motivated and produce better results. This is particularly seen in the SCRUM stand up meeting where members of the solution team (there are no designated roles) choose tasks to complete based on their own particular skills and the SCRUM Master simply facilitates this process. In contrast in the Waterfall model the Project Manager will delegate tasks to the programmer in an authoritarian manner.
- **Face-to-face communication** is encouraged: through the use of stand-up meetings and

facilitated workshops the whole team is encouraged to use face-to-face communication instead of documentation as it is more efficient and misunderstandings can be quickly addressed. Developers communicate directly with the customer (who may be collocated ▤on-site) who play an important role in offering estimates of work effort for each requirement so informed decisions on what can worked on next can be made quickly. In the waterfall approach the developers and customers are completely separated and the communication is largely through documentation created by the Analyst; this is less responsive, slow, and open to misinterpretation.

- **Continuous Integration:** as the product evolves incrementally the code is continuously integration rather than at the end in one 'Big Bang'.

A Comparison of Three Agile Methodologies

The agile methodologies called DSDM[4] , SCRUM, and XP implement the agile philosophies and principles but have a different focus (DSDM: project, business value; SCRUM: team, empirical; XP: engineering, technical). Here we compare and contrast these methodologies by reference to stages within their lifecycles and their tools and techniques. Sources for lifecycle diagrams are suggested in the resources section of this document.

Project Management

As observed from its lifecycle diagram ▲DSDM is the only methodology of the three that covers the project lifecycle fully from start to finish.5 It details what should be contained in a number of project level documents6 for example:

- In the PRE-PROJECT phase the ▤ Terms of Reference (with objectives, tentative timescale, available funding and outline of scope) is created.
- In the FEASIBILITY stage the ▤Feasibility Assessment is conducted to determine if the project is viable. Throwaway prototyping may be used to understand possible solutions.
- In the FOUNDATION stage several outline documents are created including the:
  - ▤ Business Case: containing business benefits and outline costs.
  - ▤ Management Approach: with project management activities like risk, communication, quality, change, and configuration management.
  - ▤ Solution Architecture Definition: describing for example the computer hardware and network, security and control.
  - ▤ Development Approach: for example coding standards and styles, testing strategies.
- In the POST PROJECT stage there is a ▤Project Review document which measures the benefits of the new project, after a settling in period of say 6-12 months, and compares to those expected from the business case.

Note also:

- ↻SCRUM can simply be slotted into DSDM, being wrapped in its extra project level activities if desired.
- In ✘XP a lot of up-front detailed planning is considered unnecessary and there practically no documentation. The stories themselves are held to represent the scope which is agreed by the customer. The design is said to be visible within the structure of the code itself.

[4] DSDM was published in 1995 and evolved into DSDM Atern in 2007 and into DSDM Agile Project Framework in 2014.
[5] The whole software development project can be considered as a timebox with a start and end date. XP does have a maintenance and death phase.
[6] These are living documents which evolve through the lifecycle which are only outlined at this stage; just provide the minimum information required. Students will not be expected to know the specific names of these documents but should have an appreciation of their overall content so they know what happens in this stage of the lifecycle.

Roles & Teams

In ▲ **DSDM** there are twelve roles from business, project management and technical backgrounds. Eight are further described below:

- **Business Sponsor:** responsible for providing the funds and resources; gives the go-ahead to initiate the project and to move through the feasibility and foundation stages by approving the Terms of Reference, Business Case and Management Foundations documents; not involved in the day-to-day details.
- **Business Visionary:** helps define the high level requirements in the foundations phase making sure they meet the needs of the business; communicates with stakeholders; approves the Prioritised Requirements List.
- **Business Ambassador:** representative of business that will use solution who helps to explore the requirements further during timeboxed development; day-to-day input.
- **Project Manager:** coordinates the project as a whole; leaves the detailed planning within a timebox to the team; produces the Management Foundations document and Delivery Plan.
- **Team Leader:** co-ordinates the work of the solution development team and ensures the increments are delivered on time.
- **Business Analyst:** link between business and solution development team; responsible for producing the Business Case and Prioritised Requirements List document.
- **Solution Developer:** develops prototypes and codes solution to be deployed.
- **Solution Tester:** designs and executes tests, recording results.

In contrast ↻SCRUM is simpler having only three roles:

- **Product Owner:** A representative from the business. They help define, prioritise and groom the product backlog to meet their vision for the new system.
- **Scrum Master:** Guides/facilitates/coaches the team in following the SCRUM methodology helping to remove any obstacles that might slow progress. This is not a traditional project management role as he has no authority to exert control over the team;
- **Development Team:** A collection of people who design, build and test, typically 5-9 in size. There are no specific designated roles such as programmer, tester, and database administrator as in traditional projects. Teams self-organise selecting tasks which suit their skills in the daily SCRUM.

In ✘XP there are many roles; four main roles are briefly outlined below:

- **On-site Customer:** they write user stories and acceptance tests, set priorities and must always be available to answer questions. This requires a big commitment from the business as the customer must be permanently on-site.
- **Programmer:** estimates the effort/cost of each story; breaks stories into tasks; designs unit tests and then codes.
- **Coach:** makes sure the project stays on track; trains team members.
- **Tracker:** monitors programmers taking action if anything goes off track.

Note:

- People can sometimes play more than one role.
- The Business Sponsor + Business Visionary + Business Ambassador in DSDM have similar responsibilities to the Product Owner in SCRUM and the Customer in XP. 7
- The Team Leader in DSDM is broadly equivalent to the Scrum Master in SCRUM and the Coach + Tracker in XP.

[7] ▲ DSDM having a broader range of business representatives has an advantage in that there is less pressure on one person and they will have a broader knowledge at all levels of the business, although the Product Owner and Customer do consult with all the stakeholders (end-users, operational staff, management, etc). Students should have an appreciation of the difference in the spread of roles available in each methodology – it is good to identify from the case study key business people who can play the role of the customer (or similar role).

Requirements Gathering & Specification

All the agile methodologies begin by capturing the requirements at a very high level which are elaborated later.

- In ▲ **DSDM** the gathering of high level requirements begins in the Foundations Phase. They are detailed in the ▤ **Prioritised Requirements List**.
- In ↻ **SCRUM** the requirements are contained in the **Product Backlog;** the format is not specified, but they are often stories.
- In ✘**XP** the requirements are captured as **stories**[8] in the **Exploration** phase with the Customer.

Prioritisation of Requirements

Requirements in agile projects are continually prioritised according to business value with a business representative.

- In ▲ **DSDM** the requirements are prioritised using the **MoSCoW** method in the foundations phase and throughout evolutionary development as the requirements become more detailed and refined. The mnemonic simply means:

  M – Must have requirements

  S – Should have if at all possible

  C – Could have but not critical

  W – Won 't have this time, but potentially later

- In ↻ SCRUM the product owner has the responsibility, in consultation with other stakeholders, of continually prioritising items in the **Product Backlog**[9], ordering them from their highest to lowest business value.
- In ✘XP the customers place the stories in order of priority, possibly as sticky notes on a whiteboard, in the **Planning** phase.

Planning Delivery to the Customer (Release/Deploy/Ship)

- In ▲ DSDM an outline of the planned **releases** are specified in the ▤ **Release Plan** in the **Foundations Phase**. This gives an outline of the releases which are deployed[10] to the customer in a series of increments. Each release will contain one or more development timeboxes to which prioritised requirements are allocated. This is followed by a deployment timebox. At the end of each incremental deployment there is an opportunity for the customer to give feedback (shown by back arrows on the lifecycle) into the planning process.

[8] A story is a simple description of a product requirement which may be written on an index card. It is written in the customer's own language avoiding any technical jargon. It might also describe bugs to be fixed and non- functional requirements.

[9] This is said to be 'living'.

[10] Deployment is the physical act of putting what has been assembled (the release) out into operational use. Several incremental solutions resulting from the timebox development in the previous phase are assembled into a single release, reviewed/checked as a whole, and deployed (put into the live business environment). It also includes such things as performing changeover, training users, setting up data and providing documentation and support to end-users.

- In **SCRUM** a release plan is not specifically shown on the lifecycle as each developmental sprint creates a **'potentially shippable product'** which can be shipped right away or held back and grouped with other sprints depending on what the product owner wants.[11] A subset of the items in the product backlog that can be fitted into a sprint are selected from the top of the product backlog for sprint planning.
- In ✗XP the focus is on continuous small releases to the customer at frequent intervals. These releases are planned in the **Planning Phase**[12] (or game). The developer estimates the cost/effort for each story depending on the difficulty of the story. The customer agrees on which ones to include in a release based on business value. The release plan may simply be a set of user stories on sticky notes stuck on a whiteboard and organised into releases, ordered by priority. Stories from the release are selected for the forthcoming iteration.

Planning the Work in the Time Window

The requirements/features selected for the timebox/iteration/sprint are often broken into simpler development tasks. The duration of each task is estimated to confirm everything can be fitted in the timebox/sprint/iteration.

- In ▲ **DSDM** the timebox should have a mix of Must haves, Should haves and Could haves.
- In **SCRUM** the subset of items from the product back log selected for development is called the **sprint backlog**. Sprints last about 30 days.
- In ✗**XP** this breakdown into tasks takes place in the **Planning** phase. Iterations last 1-3 weeks.

Executing the Work in the Time Window

Each day the solution development team/programmers select tasks to complete each day ensuring they work at a sustainable pace. The execution involves many elements of the software development lifecycle including the analysis, design, code and test of features in a daily iterative cycle.

The cycle begins with the **daily stand up**, also called the **SCRUM**. In this meeting (15 minutes long, same place, same time) everyone stands up to help keep the discussion short. They ask three things: What have I done? What am I going to do? What problems do I have?

- In ▲ **DSDM** the ability to leave out requirements based on MoSCoW prioritisation, reducing scope, helps to ensure that the project keeps on time and to budget.
- The execution is said to be completed in **SCRUM** when it is 'done' in other words when there is a potentially shippable product increment. This definition of 'done' usually means that features have at a minimum been designed, built, tested and documented but the definition of 'done' could be more stringent if the customer desires;
- ✗**XP** goes further and says the work is complete when until the code passes customer acceptance tests.

[11] Release planning could be performed by simply grouping the items in the product backlog into releases.
[12] Planning in XP may be separated into release and iteration planning in the planning game.

Note:

- ✘**XP** is a methodology that is specific for software projects (unlike SCRUM and DSDM that can be used for any type of project). It specifies 12 practices some of which are specifically related to software engineering and are often adopted by other methodologies: **Pair Programming**[13] [two programmers, one task, one workstation]; **Planning Game** [planning meeting for releases and iterations once per iteration]; **Test-Driven Development** [unit tests written before coding]; **Whole Team** [customer uses system and co-located with software development team]; **Continuous Integration** [always work on latest version; upload code every few hours]; **Refactoring** [make architecture simpler and improve design]; **Small Releases** [frequent releases of live functionality]; **Coding Standards** [consistent style and format for source code]; **Collective Code Ownership** [everyone jointly responsible for code]; **Simple Design** [refactor when possible to make code simpler]; **System Metaphor** [story describing how system works]; **Sustainable Pace** [software developers should not work more than 40 hours a week] .
- **Burndown charts** are used to monitor/track progress showing how much work is left to do against time.

Review of the Work in the Time Window

At the end of a timebox/iteration/sprint the working product is demonstrated to the Business Representative who inspects it to see if the requirements have been fulfilled.

- In ↻**SCRUM** there are two specific reviews:
- **Sprint Review**: In this review meeting the **PRODUCT/SYSTEM** is reviewed checking what has been done/not done. Everyone attends this meeting.
- **Sprint Retrospective**: in this review the **PROCESS** that created the product/system is reviewed. The software development team including the Scrum Master attends this meeting.

[13] One is driver in control of the keyboard and mouse; the other is the navigator who watches the driver implementing, identifies defects and gives ideas; swap over. Benefits: higher quality code, faster, more enjoyable, more confidence in work.

Applying Methodologies

When considering which methodology to apply we should consider:

- skills of the Software Development Team (can they cope with evolving prototypes? do they have knowledge of the methodologies used?);
- nature of the user requirements (clear/ ambiguous/changing and volatile?);
- technology used to build the solution (is it new or well established?);
- problem area covered (is it for an existing well established manual system or novel application?)
- proposed system complexity (routine cookie-cutter solutions do not apply) and size;
- need for reliability/safety/security;
- importance of schedule visibility;
- project timescales;
- required documentation (e.g. in a highly regulated industry).

Some guidelines for when use these methodologies are presented in the table below:

Methodology	**Guidelines**
Waterfall	requirements are unambiguous, clear and unchanging (these may be small projects) and therefore can be described in detail at an early stage of development (for example a system which automates an existing manual system); team members are inexperienced as they might find the agile methodologies more difficult to understand; the business is well understood and the technology used to build the solution is well established; where there is a requirement for formal approval in a regulated industry and detailed documentation is required; large scale projects with clear unchanging requirements unsuitable for agile development due to the size of the teams requiring a lot of communication across distributed teams; where reliability is critical (medical/missile control).
RAD	requirements of the system are unknown or uncertain and difficult to explain and are rapidly changing; it is not possible to define requirements accurately ahead of time as the business environment is rapidly changing/volatile; the technology used to build the system is new or the system being employed is highly innovative; team members are skilled to cope with evolving prototypes; developers are skilled in the use of advanced tools; timescale is short; not for critical systems like mission- control; small-medium sized projects.
▲ **DSDM**	requirements can be defined for project at a high level initially; standalone applications; major use can be made of pre-existing class libraries (APIs); performance/reliability is not critical; the required technology is more than a year old; the business would benefit from early delivery of a partial solution; no complex algorithms; main functionality visible at the user interface so users can interact with the prototypes; organisation which benefits from project management practices; projects with tight timescales and budgets where MoSCoW prioritisation can help by reducing project scope; broad range business representatives available for business roles; new-product development and enhancing existing products with innovative new features keeping businesses competitive; good for projects with strict budgets and timescales as MoSCoW prioritisation quite effective at reducing scope.

SCRUM	the developmental phases in SCRUM are very similar to DSDM and can be simply slotted into DSDM. SCRUM is therefore suitable for similar types of project. It does have a significant difference in the way that requirements are prioritised; DSDM uses MoSCoW rather than 'done' which may give DSDM the edge if there is a strict budget and time deadline; SCRUM is useful instead of DSDM when the organisation uses another project management system into which they want to slot an agile methodology rather than using the DSDM project management approach; another difference of course are the roles which may map better to the organisational structure of the business.
✗XP	when requirements are constantly changing or not fully known up front - risky projects with dynamic requirements where detailed planning is not possible; small project teams working from the same site (2 – 10 people) as communication intense and rapid feedback required; highly motivated, stable and experienced teams - there are a lot of software engineering practices and automated tools needed for testing; needs clear communication and feedback from customer who is on-site and fully committed – must be able to be released from usual work in business; automated testing tools are available; good for short time scales – do the simplest thing just to pass acceptance testing; software projects only; frequent deliveries are preferred and the project can be broken down into very small increments for release; not for large mission critical applications; not for large projects as lack of analysis and design makes coordination difficult across distributed teams.

Questions

1. Describe **three** reasons a methodology is used for the software development process.

1. ____

____

____

2. ____

____

____

3. ____

____

____

2. Provide two advantages and two disadvantages when applying the following methodologies for the development of different types of software projects.

Methodology	Advantage	Disadvantage
Waterfall	1. 2.	1. 2.
RAD	1. 2.	1. 2.
DSDM	1. 2.	1. 2.
SCRUM	1. 2.	1. 2.
XP	1. 2.	1. 2.

3. Suggest which methodology would be the most suitable for the following projects. Justify your selection.

Project	**Suggested Methodology**	**Justification**
A system for the Electoral Office in Northern Ireland which maintains a register of voters including those that are postal and proxy, and prints poll cards and ballot papers.		
Uses novel deep learning technology which classifies types of cancer using images of tumour sections from pathology departments in the hospital. Team is composed of highly qualified and experienced programmers and experts with research degrees in the deep learning algorithms.		
A pharmaceutical laboratory management system which stores records of all clinical trial experiments for new drugs.		
A 3D virtual reality chemistry app for an e-learning business which must embrace the high expectations of learners who are used to embracing the most up-date technology online especially game playing.		
An ordering system for a restaurant that is struggling to cope with demand.		
A system for a UK wide hotel chain which includes booking of facilities and room, equipment hire and a shop.		
An online payment and booking system for a local campsite.		

4. Describe one activity that each role performs in the lifecycle of that particular methodology.

Methodology	**Role**	**Activity**
Waterfall	Customer/User	
	Analyst	
	Developer	
RAD	Customer/User	
	Analyst	
	Developer	
DSDM	Business Sponsor	
	Business Visionary	
	Business Ambassador	
	Project Manager	
	Team Leader	
	Business Analyst	
	Solution Developer	
SCRUM	Product Owner	
	Scrum Master	
	Development Team	
XP	On-site Customer	
	Programmer	
	Coach	
	Tracker	

5. Explain the following tools or techniques, giving examples of how they benefit the development process and how they might be used in a specific methodology.

Tools/Techniques	**Usage**	**Benefit**
Evolutionary Prototyping		
Throwaway Prototyping		
Timeboxing		
Facilitated Workshops		
MoSCoW prioritisation		
Daily Stand-Up		
Pair Programming		
User Stories		
Reviews & Retrospectives		
Sustainable pace		

6. Explain the difference between incremental and iterative, giving examples of how it is implemented in various methodologies, and the benefits it brings to the business.

7. Discuss the use of various kinds of prototyping throughout the development lifecycle and their advantage and disadvantages of using them for particular types of software development projects.

On-Line Resources

- DSDM Lifecycle
- XP Lifecycle
- SCRUM lifecycle
- User stories
- User stories grouped for release
- Iterative versus Incremental
- SCRUM Video
- Increments and Timeboxes
- MoSCoW (sticky notes)
- SCRUM sprint retrospective
- Burndown chart
- SCRUM ACDT iterations
- Iterations and Increments
- DSDM Timeboxes and Depolyment – how iterations and increments can be related.