Rapid software development

Rapid software development

- Because of rapidly changing business environments, businesses have to respond to new opportunities and competition.
- This requires software and rapid development and delivery is not often the most critical requirement for software systems.
- Businesses may be willing to accept lower quality software if rapid delivery of essential functionality is possible.

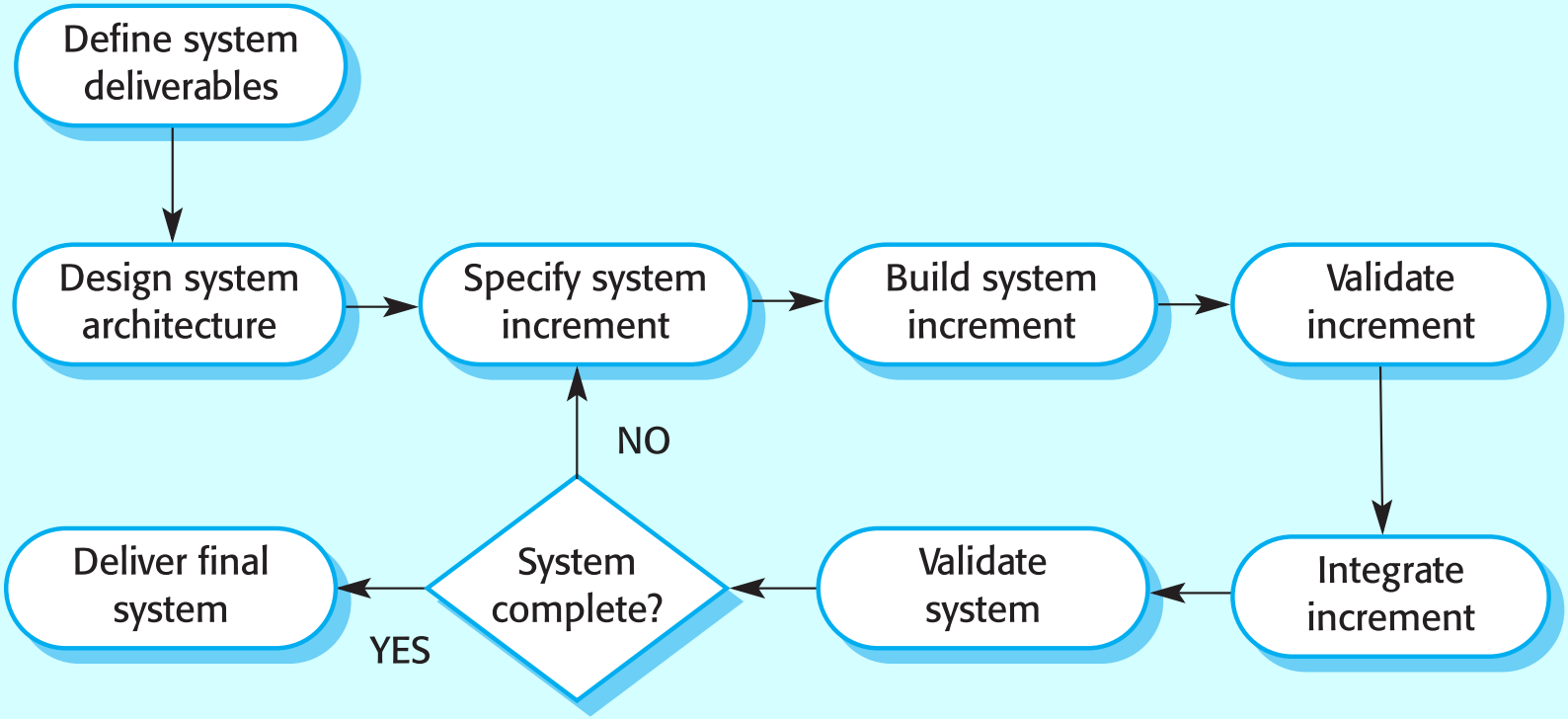
Requirements

- Because of the changing environment, it is often impossible to arrive at a stable, consistent set of system requirements.
- Therefore a waterfall model of development is impractical and an approach to development based on iterative specification and delivery is the only way to deliver software quickly.

Characteristics of RAD processes

- The processes of specification, design and implementation are concurrent. There is no detailed specification and design documentation is minimised.
- The system is developed in a series of increments. End users evaluate each increment and make proposals for later increments.
- System user interfaces are usually developed using an interactive development system.

An iterative development process



Advantages of incremental development

- Accelerated delivery of customer services. Each increment delivers the highest priority functionality to the customer.
- User engagement with the system. Users have to be involved in the development which means the system is more likely to meet their requirements and the users are more committed to the system.

Problems with incremental development

- Management problems
  - Progress can be hard to judge and problems hard to find because there is no documentation to demonstrate what has been done.
- Contractual problems
  - The normal contract may include a specification; without a specification, different forms of contract have to be used.
- Validation problems
  - Without a specification, what is the system being tested against?
- Maintenance problems
  - Continual change tends to corrupt software structure making it more expensive to change and evolve to meet new requirements.

Agile methods

- Dissatisfaction with the overheads involved in design methods led to the creation of agile methods. These methods:
  - Focus on the code rather than the design;
  - Are based on an iterative approach to software development;
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- Agile methods are probably best suited to small/ medium-sized business systems or PC products.

Principles of agile methods

Principle	Description
Customer involvement	The customer should be closely involved throughout the development process. Their role is provide and prioritise new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and design the system so that it can accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

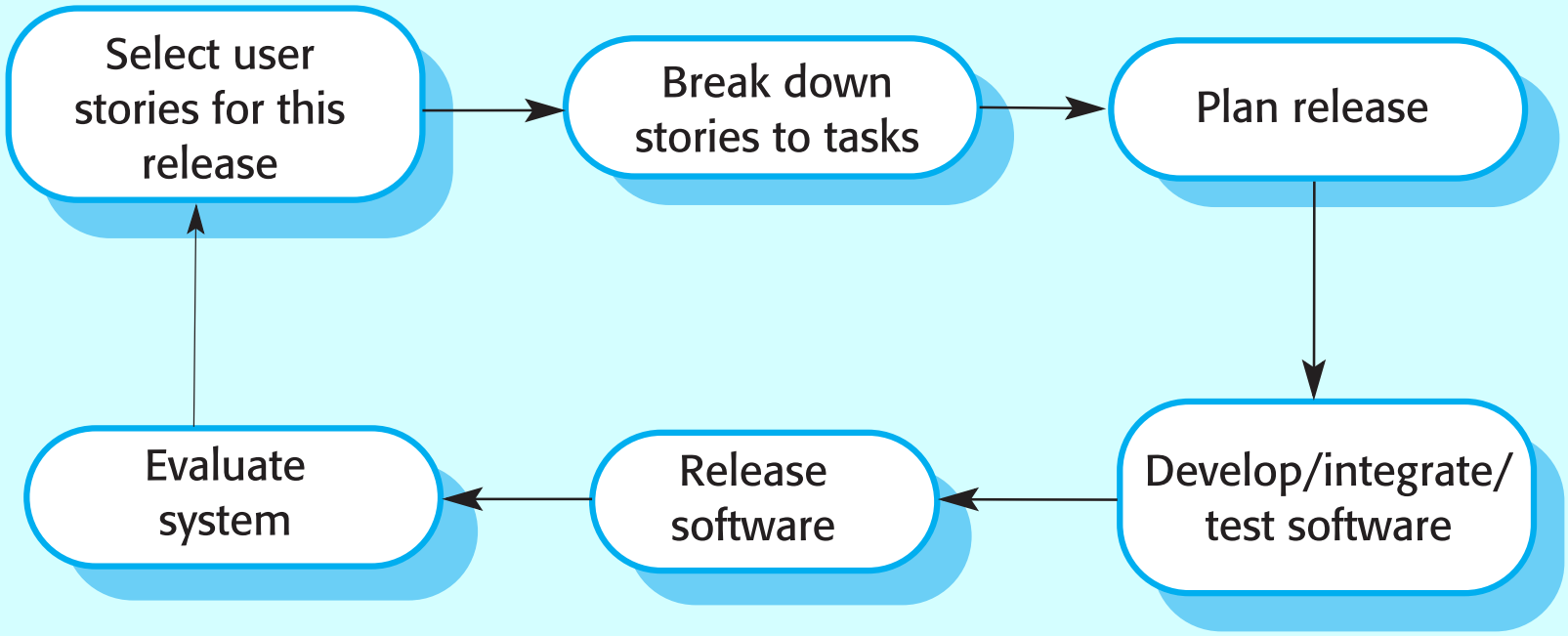
Problems with agile methods

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

Extreme programming

- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

The XP release cycle



Extreme programming practices

Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'.
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices

Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

Requirements scenarios

- In XP, user requirements are expressed as scenarios or user stories.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

Story card for document downloading

Downloading and printing an article

First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card.

After this, you get a copyright form from the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer.

You then choose a printer and a copy of the article is printed. You tell the system if printing has been successful.

If the article is a print-only article, you can't keep the PDF version so it is automatically deleted from your computer.

XP and change

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

Testing in XP

- Test-first development.
- Incremental test development from scenarios.
- User involvement in test development and validation.
- Automated test harnesses are used to run all component tests each time that a new release is built.

Task cards for document downloading

Task 1: Implement principal workflow

Task 2: Implement article catalog and selection

Task 3: Implement payment collection

Payment may be made in 3 different ways. The user selects which way they wish to pay. If the user has a library subscription, then they can input the subscriber key which should be checked by the system. Alternatively, they can input an organisational account number. If this is valid, a debit of the cost of the article is posted to this account. Finally, they may input a 16 digit credit card number and expiry date. This should be checked for validity and, if valid a debit is posted to that credit card account.

Test case description

Test 4: Test credit card validity

Input:
A string representing the credit card number and two integers representing the month and year when the card expires

Tests:
Check that all bytes in the string are digits
Check that the month lies between 1 and 12 and the year is greater than or equal to the current year.
Using the first 4 digits of the credit card number, check that the card issuer is valid by looking up the card issuer table. Check credit card validity by submitting the card number and expiry date information to the card issuer

Output:
OK or error message indicating that the card is invalid

Test-first development

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
- All previous and new tests are automatically run when new functionality is added. Thus checking that the new functionality has not introduced errors.

Pair programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.