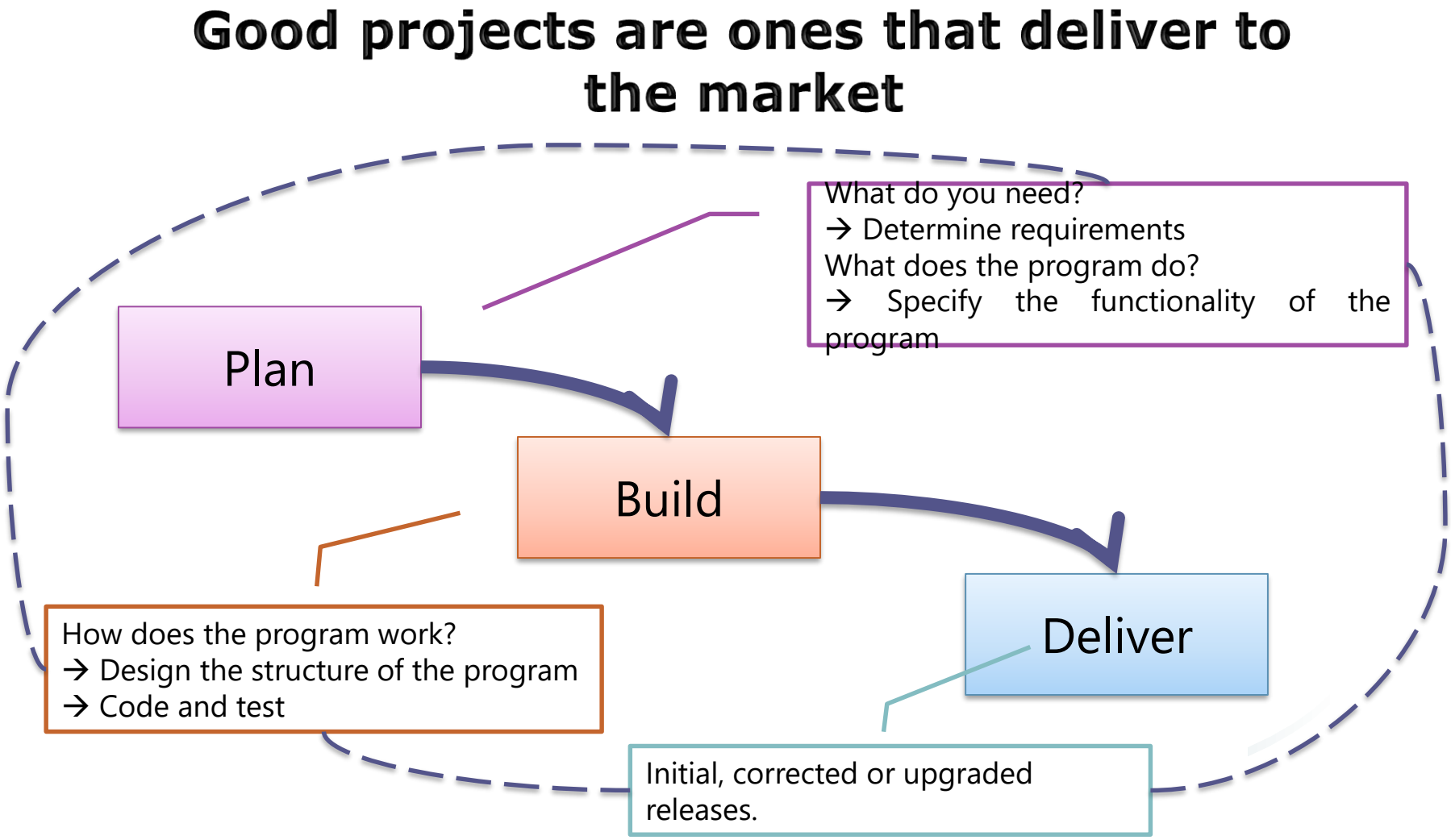
Rapid software development

Dr. Vu Thi Huong Giang

Traditional Development Process



Rapid software development

- Because of rapidly changing business environments, it is often impossible to arrive at a stable, consistent set of system requirements

  →A traditional model of development (completely specifying the requirements, then designing, building and testing the system) is impractical
- An approach to development based on iterative specification and delivery is the only way to deliver software quickly

Covered topics

- Agile method
- Extreme programming
- Rapid application development
- Rapid prototyping

Objectives

- After this lesson, students will be able to:
  - Recall the principles and problems of agile methods.
  - Prepare story cards, task cards, test cards for a XP project.
  - Point out the characteristics of RAD processes
  - Show possible prototyping approaches for a specific software.

I. AGILE METHOD

1. Concept
2. Principles
3. Problems

1. Concept

- Agility is the ability to create and respond to change in order to profit in a turbulent business environment.
- Agile methods are created to cope with the overheads involved in design methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development (incremental development)
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements (incremental delivery)
- Agile methods are probably best suited to small/medium-sized business systems or PC products

2. 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

3. 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 characterizes agile methods
- Prioritizing 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
- Suitable for the development of small or medium-sized business systems and personal computer products
- Not well suited to
  - large-scale systems development with the development teams in different places and where there may be complex interactions with other hardware and software systems
  - critical systems development where a detailed analysis of all of the system requirements is necessary to understand their safety or security implications.

I. EXTREME PROGRAMMING (XP)

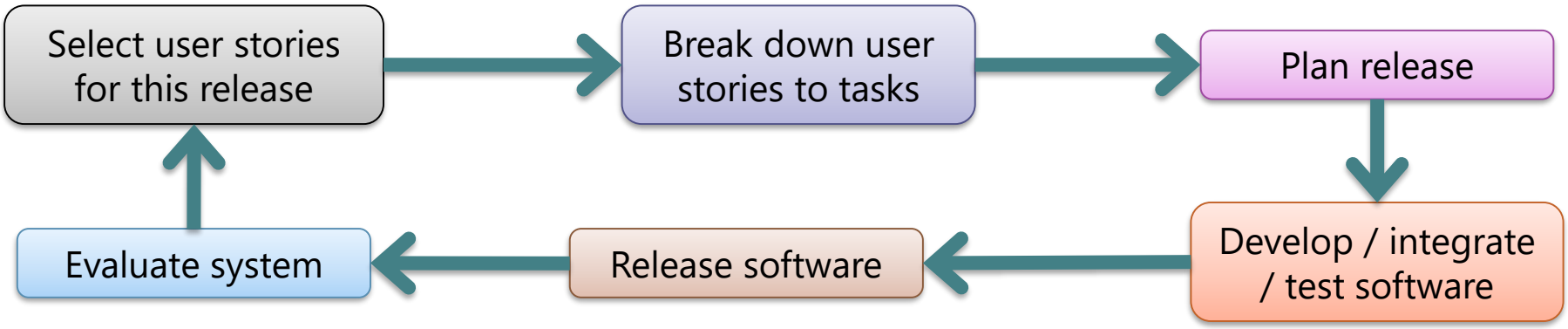
Perhaps the best-known and most widely used agile method

1. Idea of XP
2. The XP release cycle
3. XP and agile principles
4. Problem with XP

1. Idea of Extreme programming (XP)

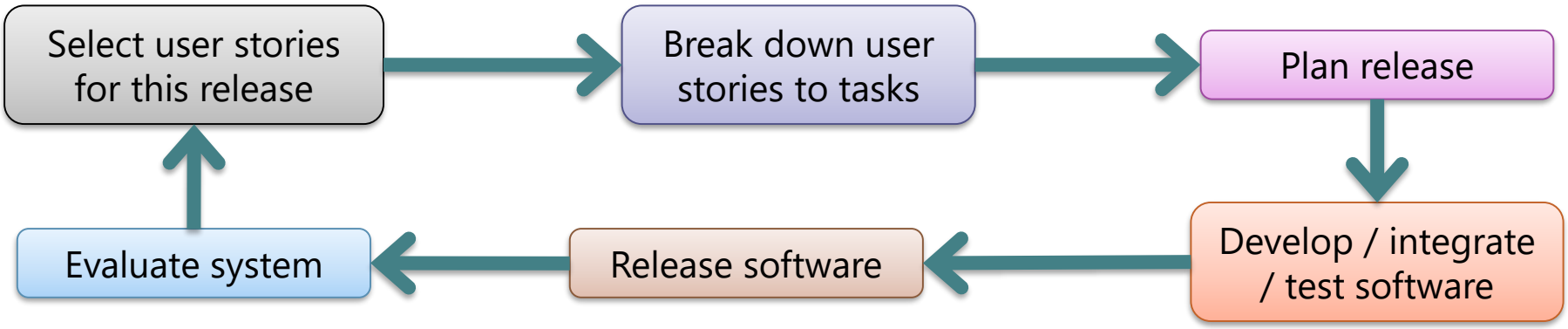
- Extreme Programming takes an ‘extreme’ approach to iterative development
  - New versions may be built several times per day
  - Increments are delivered to customers every 2 weeks
  - All tests must be run for every build and the build is only accepted if tests run successfully

2. The XP release cycle



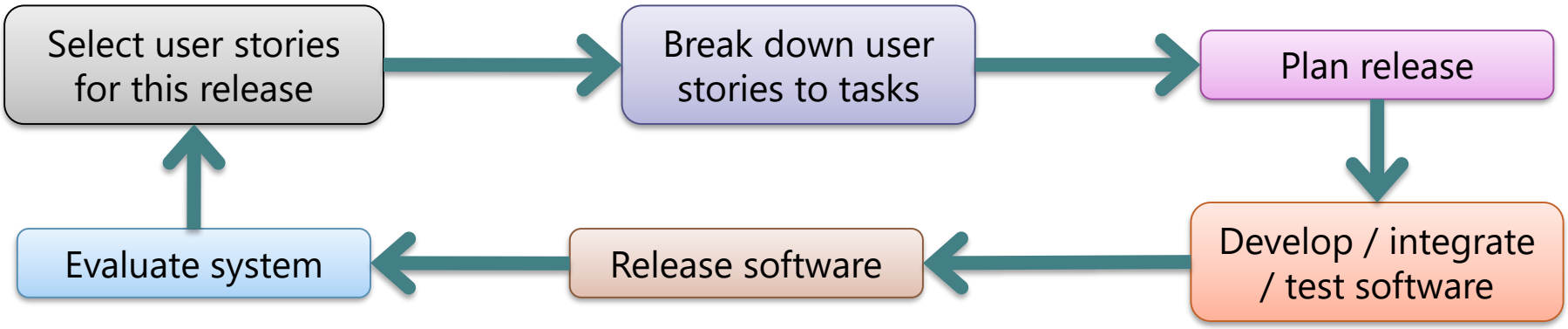
Incremental planning	Requirements are recorded on Story Cards Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'.
Small Releases	The minimal useful set of functionality that provides business 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.

The XP release cycle



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.
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.

The XP release cycle



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.

3. XP and agile principles

Principle	Description in XP
Customer involvement	full-time customer engagement with the team: •To help develop stories that define the requirements •To help prioritize the features to be implemented in each release •To help develop acceptance tests which assess whether or not the system meets its requirements
Incremental delivery	small, frequent system releases
People not process	**pair programming** collective ownership process that avoids long working hours
Embrace change	regular system releases **test-first development** continuous integration
Maintain simplicity	**constant refactoring of code**

3.1. 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
- This serves as an informal review process as each line of code is looked at by more than 1 person
- This 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

Requirements scenarios: Story card for document downloading

- In XP, user requirements are expressed as scenarios or user stories; these are written on cards.
- The development team break these stories 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

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, though a company account or by credit card.
Alter this, you get a copyright form from the system to fill in.
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 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.

Working: Banking system

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 organizational 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

Working: Banking system

3.2. 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

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

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 expire date information to the card issuer

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

Working: Banking system

3.3. Refactoring

- XP proposes refactoring to make changes easier when they have to be implemented.
- Refactoring is the process of constant code improvement where code is reorganized and rewritten to make it more efficient, easier to understand, etc
- Refactoring is required because frequent releases mean that code is developed incrementally and therefore tends to become messy
- Refactoring should not change the functionality of the system
- Automated testing simplifies refactoring as you can see if the changed code still runs the tests successfully

4. Problems with XP

- Customer involvement
  - It may be difficult or impossible to find a customer who can represent all stakeholders and who can be taken off their normal work to become part of the XP team
  - For generic products, there is no ‘customer’ - the marketing team may not be typical of real customers
- Architectural design
  - The incremental style of development can mean that inappropriate architectural decisions are made at an early stage of the process
  - Problems with these may not become clear until many features have been implemented and refactoring the architecture is very expensive
- Test complacency
  - It is easy for a team to believe that because it has many tests, the system is properly tested
  - Because of the automated testing approach, there is a tendency to develop tests that are easy to automate rather than tests that are ‘good’ tests

II. RAPID APPLICATION DEVELOPMENT

1. Concept
2. Characteristics of RAD processes
3. RAD environment tools

1. Concept

- Agile methods have received a lot of attention but other approaches to rapid application development have been used for many years
- These are designed to develop data-intensive business applications and rely on programming and presenting information from a database

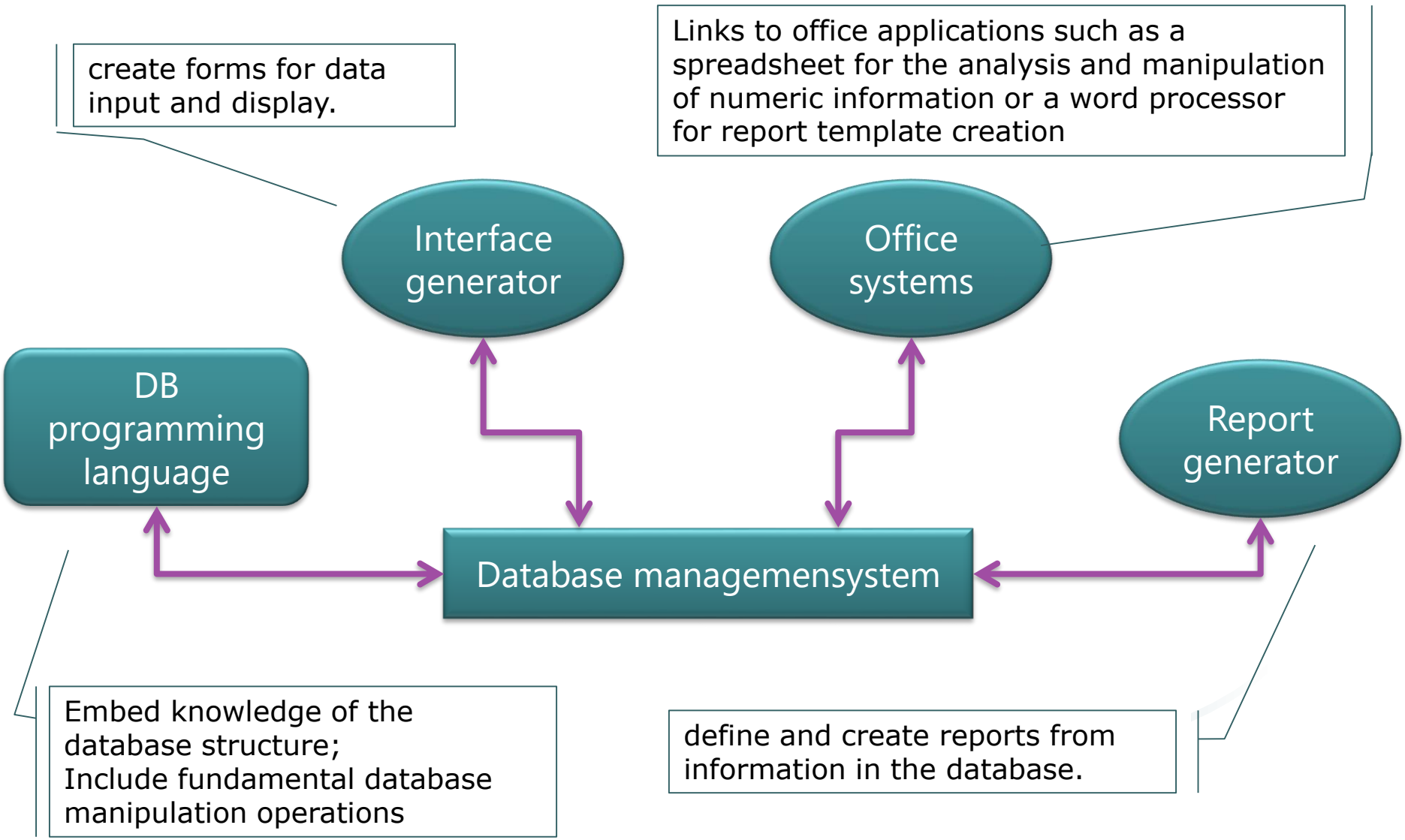
2. Characteristics of RAD processes

- The processes of specification, design and implementation are concurrent
- The user requirements document defines only the most important characteristics of the system
- There is no detailed specification
- Design documentation is minimized
- 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

3. RAD environment tools

- Database programming language
- Interface generator
- Links to office applications
- Report generators

A RAD environment



Interface generation

- Many applications are based around complex forms and developing these forms manually is a time-consuming activity
- RAD environments include support for screen generation including:
  - Interactive form definition using drag and drop techniques
  - Form linking where the sequence of forms to be presented is specified
  - Form verification where allowed ranges in form fields is defined

COTS (Commercial Off the Shelf) reuse

- An effective approach to rapid development is to configure and link existing off the shelf systems
- For example, a requirements management system could be built by using:
  - A database to store requirements
  - A word processor to capture requirements and format reports
  - A spreadsheet for traceability management

Database programming languages

- Domain specific languages for business systems based around a database management system
- Normally include a database query language, a screen generator, a report generator and a spreadsheet
- May be integrated with a CASE toolset
- The language + environment is sometimes known as a fourth-generation language (4GL)
- Cost-effective for small to medium sized business systems

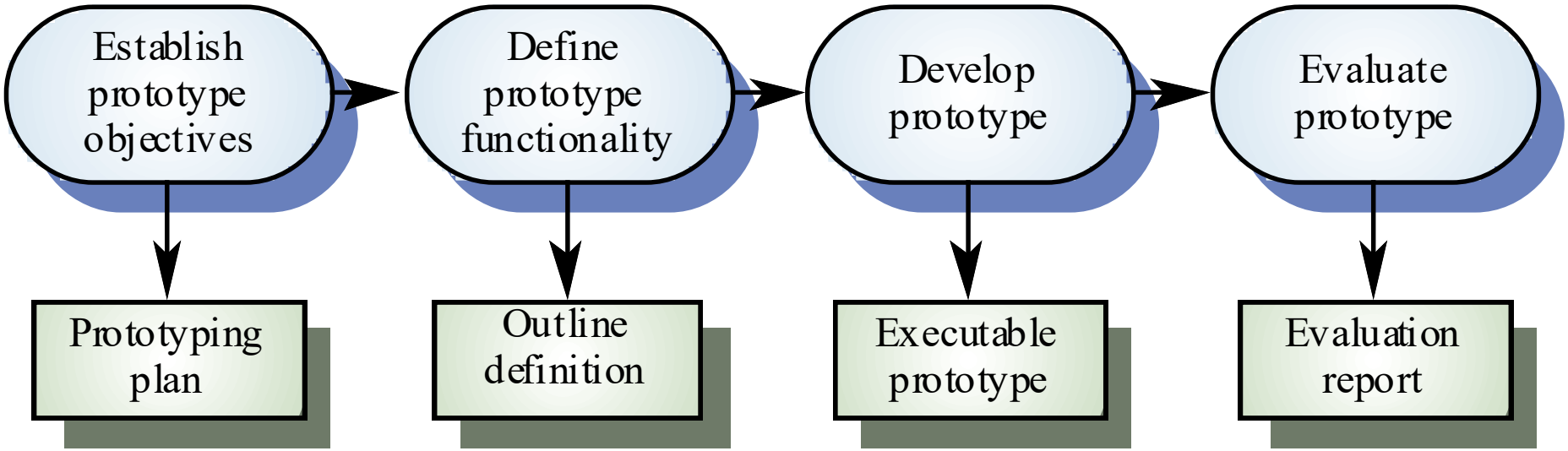
III. Rapid prototyping

1. Concepts
2. Techniques

1.1. Prototype

- A prototype is an initial version of a software system that is used to demonstrate concepts, try out design options and, generally, to find out more about the problem and its possible solutions
- A prototype is used:
  - In the requirements engineering process: help customers and developers understand the requirements for the system
    - Requirements elicitation: Users can experiment with a prototype to see how the system supports their work
    - Requirements validation: The prototype can reveal errors and omissions in the requirements
  - In the system design process: explore particular software solutions and to support user interface design.
  - In the testing process: run back-to-back tests with the system that will be delivered to the customer.
- → Prototyping is the rapid development of a system. Prototyping can be considered as a risk reduction activity which reduces requirements, design and testing risks

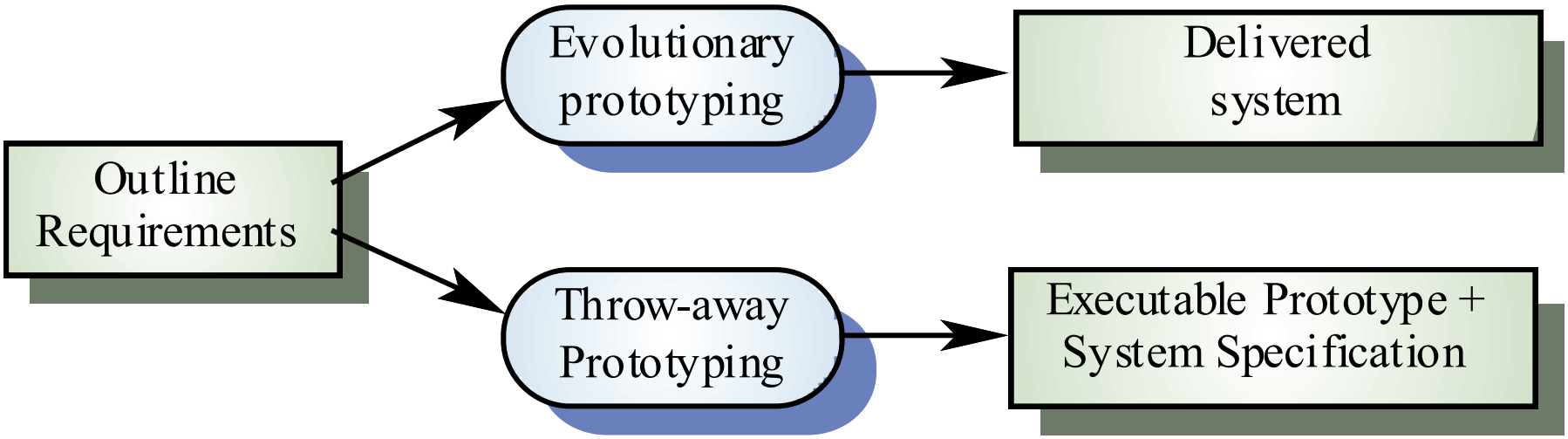
1.2. Prototyping process



1.3. Prototyping benefits

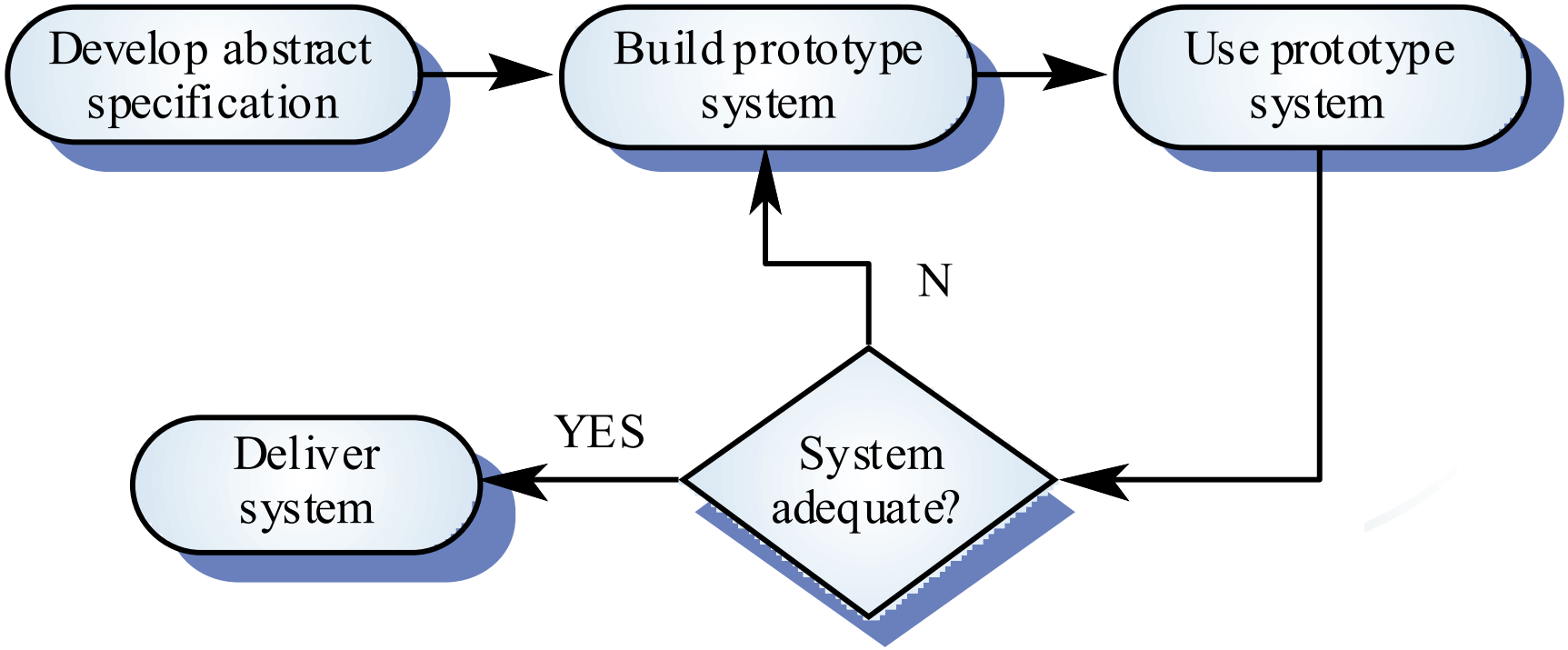
- Misunderstandings between software users and developers are exposed
- Missing services may be detected and confusing services may be identified
- A working system is available early in the process
- The prototype may serve as a basis for deriving a system specification
- The system can support user training and system testing
- Improved system usability
- Closer match to the system needed
- Improved design quality
- Improved maintainability
- Reduced overall development effort

1.4. Approaches to prototyping



1.4.1. Evolutionary prototyping

- Specification, design and implementation are inter-twined
- The system is developed as a series of increments that are delivered to the customer
- Techniques for rapid system development are used such as CASE tools and 4GLs
- User interfaces are usually developed using a GUI development toolkit

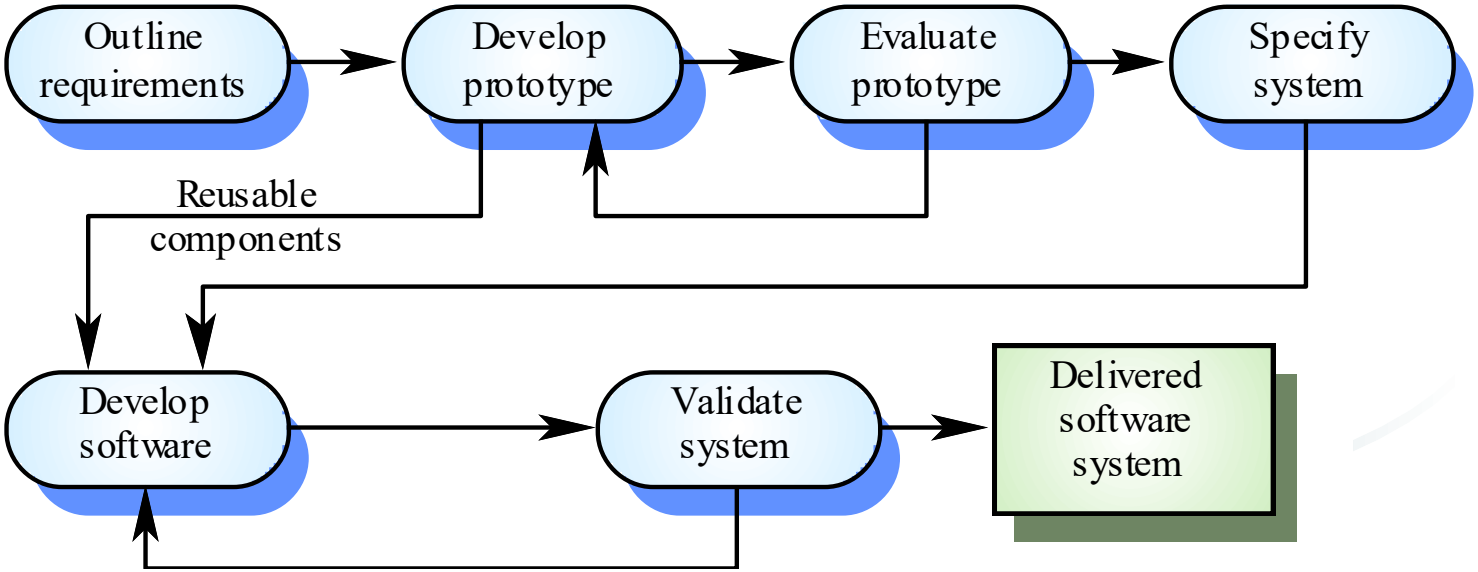


Advantages and Problems

- Advantages:
  - Accelerated delivery of the system
    - Rapid delivery and deployment are sometimes more important than functionality or long-term software maintainability
  - User engagement with the system
    - Not only is the system more likely to meet user requirements, they are more likely to commit to the use of the system
- Problems:
  - Management problems
    - Existing management processes assume a waterfall model of development
    - Specialist skills are required which may not be available in all development teams
  - Maintenance problems
    - Continual change tends to corrupt system structure so long-term maintenance is expensive
  - Contractual problems

1.4.2. Throw-away prototyping

- Used to reduce requirements risk
- The prototype is developed from an initial specification, delivered for experiment then discarded
- The throw-away prototype should NOT be considered as a final system
  - Some system characteristics may have been left out
  - There is no specification for long-term maintenance
  - The system will be poorly structured and difficult to maintain



1.4.3. Prototype delivery

- Developers may be pressurised to deliver a throw-away prototype as a final system
- This is not recommended
  - It may be impossible to tune the prototype to meet non-functional requirements
  - The prototype is inevitably undocumented
  - The system structure will be degraded through changes made during development
  - Normal organisational quality standards may not have been applied

1.4.4. Prototypes as specifications

- Some parts of the requirements (e.g. safety-critical functions) may be impossible to prototype and so don’t appear in the specification
- An implementation has no legal standing as a contract
- Non-functional requirements cannot be adequately tested in a system prototype

2. Rapid prototyping techniques

- Various techniques may be used for rapid development
  - Dynamic high-level language development
  - Database programming
  - Component and application assembly
- These are not exclusive techniques - they are often used together
- Visual programming is an inherent part of most prototype development systems

2.1. Dynamic high-level languages

- Languages which include powerful data management facilities
- Need a large run-time support system Not normally used for large system development
- Some languages offer excellent UI development facilities
- Some languages have an integrated support environment whose facilities may be used in the prototype

Language	Type	Application domain
Smalltalk	Object-oriented	Interactive systems
Java	Object-oriented	Interactive systems
Prolog	Logic	Symbolic processing
Lisp	List-based	Symbolic processing

Choice of prototyping language

- What is the application domain of the problem?
- What user interaction is required?
- What support environment comes with the language?

→ Different parts of the system may be programmed in different languages. However, there may be problems with language communications.

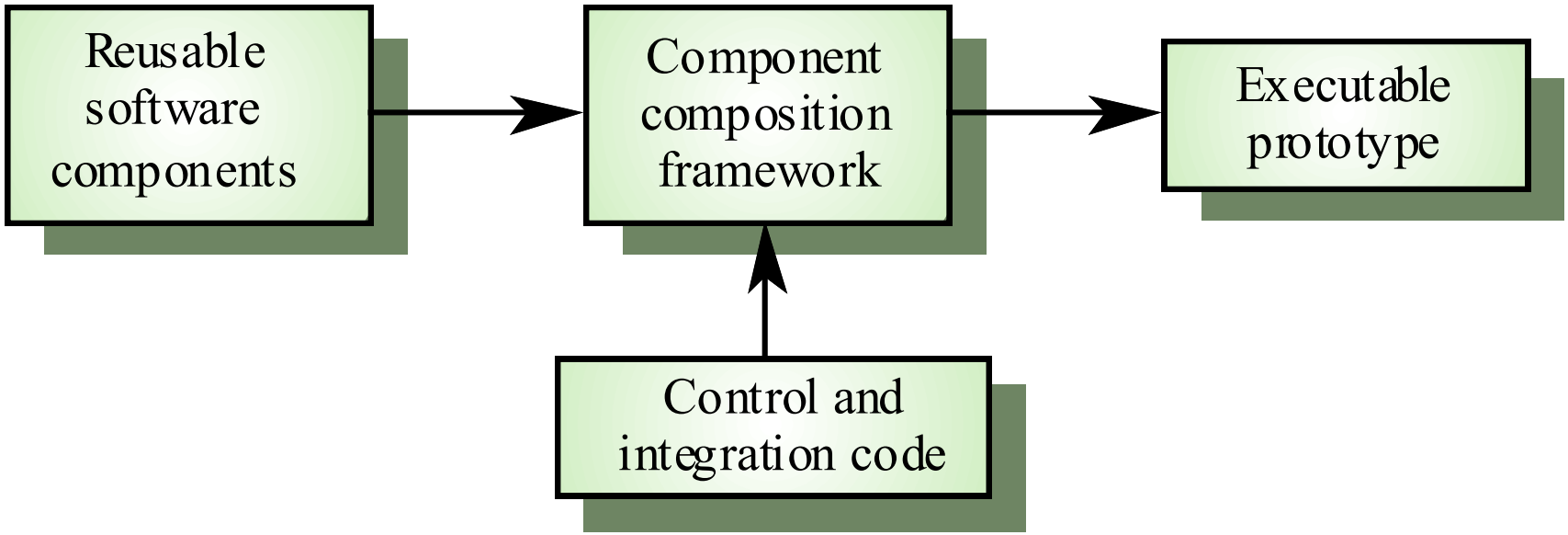
2.2. Component and application assembly

- Prototypes can be created quickly from a set of reusable components plus some mechanism to ‘glue’ these component together
- The composition mechanism must include control facilities and a mechanism for component communication
- The system specification must take into account the availability and functionality of existing components

2.2.1. Prototyping with reuse

- Application level development
  - Entire application systems are integrated with the prototype so that their functionality can be shared
  - For example, if text preparation is required, a standard word processor can be used
- Component level development
  - Individual components are integrated within a standard framework to implement the system
  - Frame work can be a scripting language or an integration framework such as CORBA

Reusable component composition



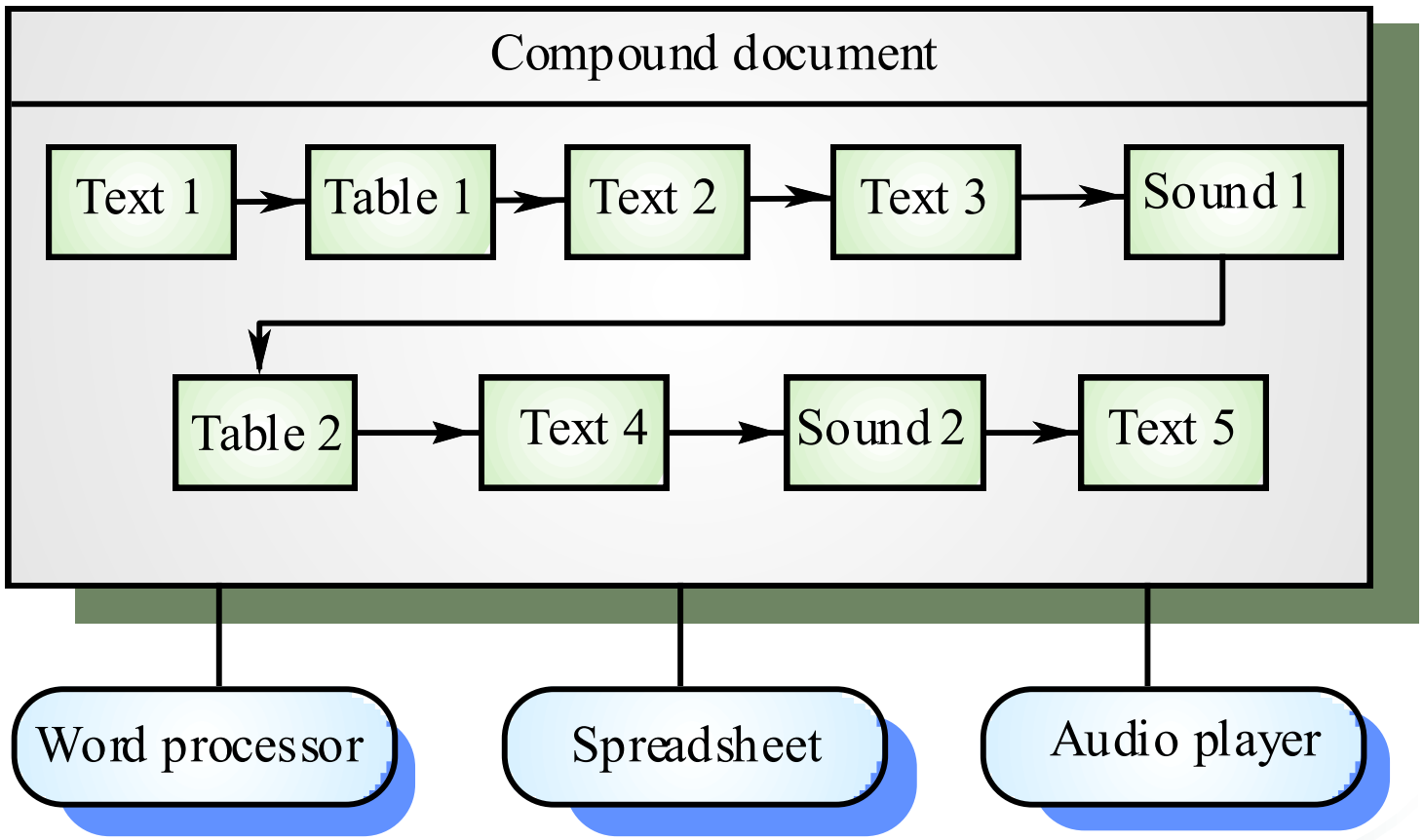
2.2.2. User interface prototyping

- It is impossible to pre-specify the look and feel of a user interface in an effective way. Prototyping is essential.
- UI development consumes an increasing part of overall system development costs
- User interface generators may be used to ‘draw’ the interface and simulate its functionality with components associated with interface entities
- Web interfaces may be prototyped using a web site editor

a. Compound documents

- For some applications, a prototype can be created by developing a compound document
- This is a document with active elements (such as a spreadsheet) that allow user computations
- Each active element has an associated application which is invoked when that element is selected
- The document itself is the integrator for the different applications

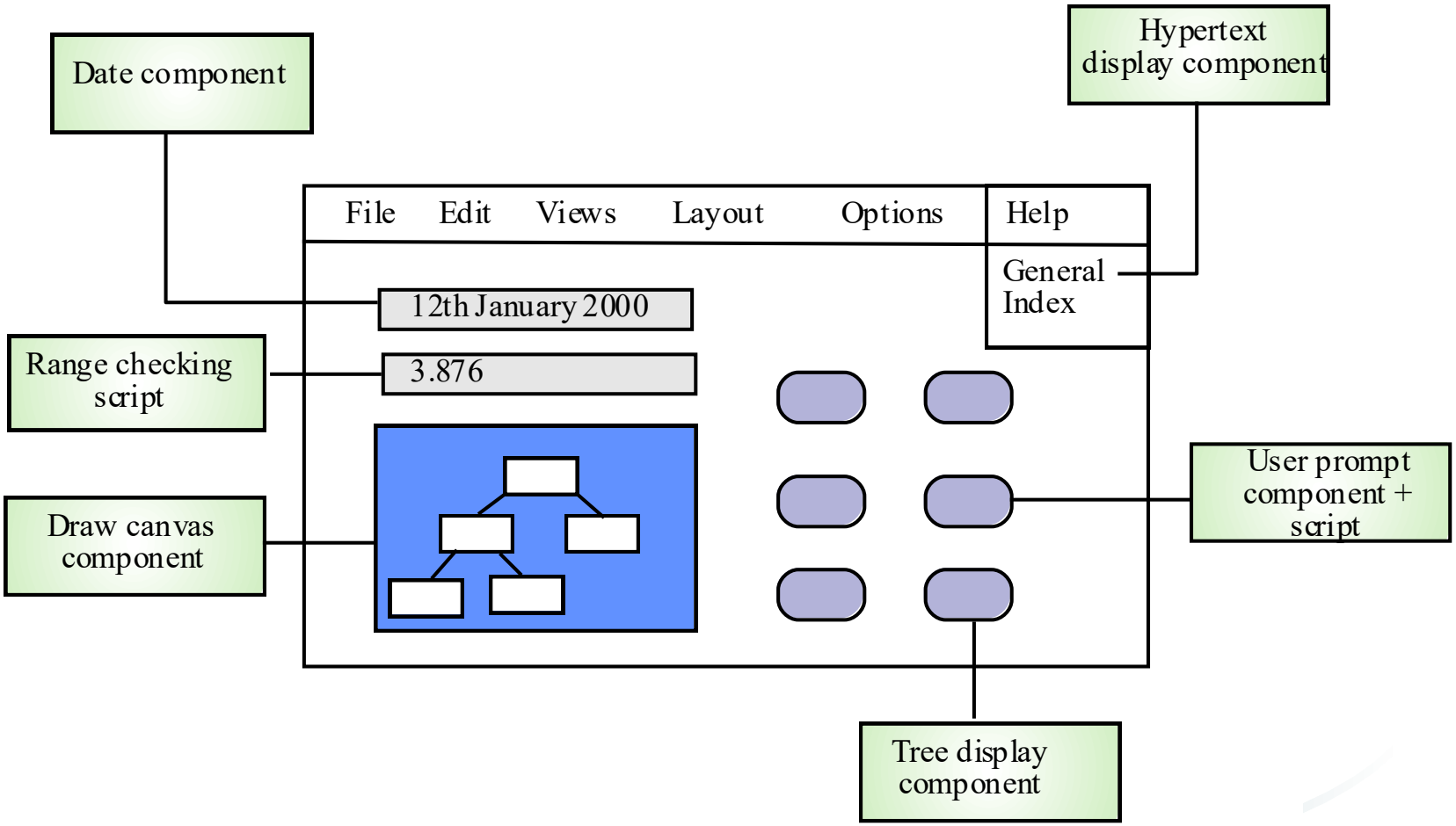
Application linking in compound documents



b. Visual programming

- Scripting languages such as Visual Basic support visual programming where the prototype is developed by creating a user interface from standard items and associating components with these items.
- A large library of components exists to support this type of development
- These may be tailored to suit the specific application requirements

Visual programming with reuse



Problems with visual development

- Difficult to coordinate team-based development
- No explicit system architecture
- Complex dependencies between parts of the program can cause maintainability problems

Quiz and Exercises

- Now let’s go over what you have learned through this lesson by taking a quiz.
- When you’re ready, press Start button to take the quiz