Agile methods

Objectives

- To explain how an iterative, incremental development process leads to faster delivery of more useful software
- To discuss the essence of agile development methods
- To explain the principles and practices of extreme programming
- To explain the roles of prototyping in the software process

COTS and 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;

Incremental development

- Develop system as a series of builds, each adds new functionality.
  - Concept of iterative enhancement developed at UMD in the 1970s.
- Features:
  - 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.

Rapid software development

- Even builds may be too slow.
- Because of rapidly changing business environments, businesses have to respond to new opportunities and competition.
  - This often requires rapid development and delivery .
  - Businesses may be willing to accept lower quality software if rapid delivery of essential functionality is possible.
  - 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.

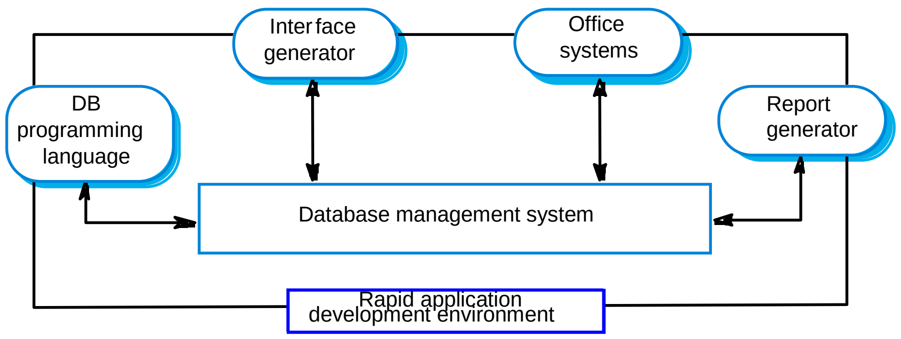
Rapid Application Development (RAD)

- The processes of specification, design and implementation are concurrent. There is no detailed specification and 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.
- 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.

RAD environment tools

- Generally organized around business applications:
  - ❑ 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.

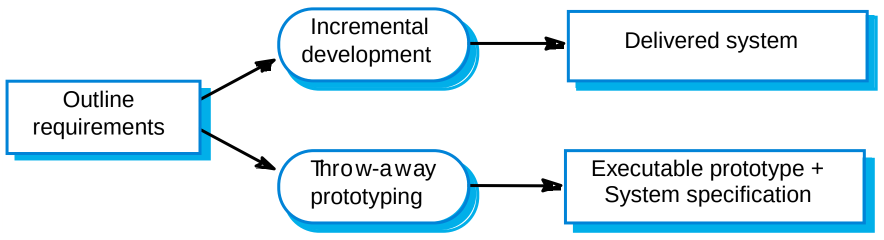
Problems with RAD

- 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.

Prototyping

- For some large systems, incremental iterative development and delivery may be impractical; this is especially true when multiple teams are working on different sites.
- Prototyping, where an experimental system is developed as a basis for formulating the requirements may be used. This system is thrown away when the system specification has been agreed.
  - Problem: Prototype may not be thrown away; it may be all there is!

Incremental development vs. prototyping



Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - ❑ The requirements engineering process to help with requirements elicitation and validation;
  - ❑ In design processes to explore options and develop a UI design;
  - ❑ In the testing process to run back-to-back tests.

Benefits of prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organizational quality standards.

Conflicting objectives

- The objective of incremental development is to deliver a working system to end-users. The development starts with those requirements which are best understood.
- The objective of throw-away prototyping is to validate or derive the system requirements. The prototyping process starts with those requirements which are poorly understood.

Why a need for agile development?

- Waterfall Model required a complete analysis of user requirements.
  - Months of intense interaction with users and customers
  - programmers implement and the complete system is tested and shipped
- But users change their minds.
  - After months, of collecting requirements users still not sure of what they want
  - Requirements tend to change mid-development and difficult to stop the momentum of the project to accommodate the change
  - Copious amounts of documentation need to be kept up to date to accommodate even small changes
- CMM focuses on turning software development into repeatable, defined, and predictable processes, but were found that many of them were, in fact, largely unpredictable and unrepeatable because
  - Applicable first principles are not present.
  - The process is only beginning to be understood.
  - The process is complex.
  - The process is changing and unpredictable.

Lean manufacturing

1. Eliminate waste—eliminate or optimize consumables such as diagrams and models that do not add value to the final deliverable.
2. Minimize inventory—minimize intermediate artifacts such as requirements and design documents.
3. Maximize flow—use iterative development to reduce development time.
4. Pull from demand—support flexible requirements.
5. Empower workers—generalize intermediate documents, "tell developers what needs to be done, not how to do it."
6. Meet customer requirements—work closely with the customer, allowing them to change their minds.
7. Do it right the first time—test early and refactor when necessary.
8. Abolish local optimization—flexibly manage scope.
9. Partner with suppliers—avoid adversarial relationships, work towards developing the best software.
10. Create a culture of continuous improvement—allow the process to improve, learn from mistakes and successes.

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.
- Not “one definition” of agile methods, but a set of related techniques. We’ll discuss XP, SCRUM, Pair Programming. There are others.

Agile Manifesto

- Kenny Beck and 16 others met in 2001:
- Issued “The Agile Manifesto”:
  - ❑ We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
    - Individuals and interaction over process and tools,
    - Working software over comprehensive documentation,
    - Customer collaboration over contract negotiation,
    - Responding to change over following a plan.
  - ❑ That is, while there is a value in the items on the right, we value the items on the left more.

Principles of agile methods

Principle	Description
Customer involvement	The customer should be closely involved throughout the development process. Their role is provide and prioritize 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 recognized 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 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.
- But agile does not mean “no process” as opponents to it claim.

User-based stories (Requirements Elicitation)

- Agile practices are based on the belief that neither the customer nor the developers have full knowledge in the beginning and that the important consideration is having practices that will allow both [the customer and the developer] to learn and evolve as that knowledge is gained—without ongoing recrimination.
- Agile methods believe that the statement of requirements will evolve through the whole project.

Agile requirements engineering

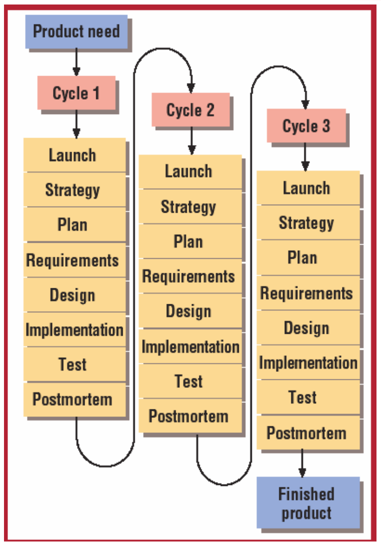
- Frequent personal interactions (customer on site with developer)
- Frequent delivery of software (e.g., every 2 weeks)
- Expressing requirements as features:
  - Use cases to describe features
  - A feature is a small, client-valued function expressed in the form <action><result><object>

    (e.g. calculate the total of a sale)
  - Discussed later

Basics of user stories

- A user story represents a feature customers want in the software.
- A user story is developed by a customer
  - ❑ Customer writes user story on index card.
    - Developers do not write stories
    - Stories must be testable
    - Stories should be independent
  - ❑ Developer estimates development time
  - ❑ Acceptance tests are developed by customer and developer
  - ❑ Iterate. Divide big stories into smaller ones
- User stories are collected
- Stories to be implemented next are discussed after each iteration

Team Software Process



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.

12 rules of Extreme Programming

1. *The planning game*: At the start of each iteration, customers, managers, and developers meet to flesh out, estimate, and prioritize requirements for the next release. The requirements are called "user stories" and are captured on "story cards" in a language understandable by all parties.
2. *Small releases*: An initial version of the system is put into production after the first few iterations. Subsequently, working versions are put into production anywhere from every few days to every few weeks.
3. *Metaphor*: Customers, managers, and developers construct a metaphor, or set of metaphors after which to model the system.
4. *Simple design*: Developers are urged to keep design as simple as possible, "say everything once and only once"
5. *Tests*: Developers work test-first; they write acceptance tests for their code before they write the code itself. Customers write functional tests for each iteration and at the end of each iteration, all tests should run.
6. *Refactoring*: As developers work, the design should be evolved to keep it as simple as possible.

12 rules of Extreme Programming -2

7. *Pair programming*: Two developers sitting at the same machine write all code.
8. *Continuous integration*: Developers integrate new code into the system as often as possible. All functional tests must still pass after integration or the new code is discarded.
9. *Collective ownership*: The code is owned by all developers, and they may make changes anywhere in the code at anytime they feel necessary.
10. *On-site customer*: A customer works with the development team at all times to answer questions, perform acceptance tests, and ensure that development is progressing as expected.
11. *40-hour weeks*: Requirements should be selected for each iteration such that developers do not need to put in overtime.
12. *Open workspace*: Developers work in a common workspace set up with individual workstations around the periphery and common development machines in the center.

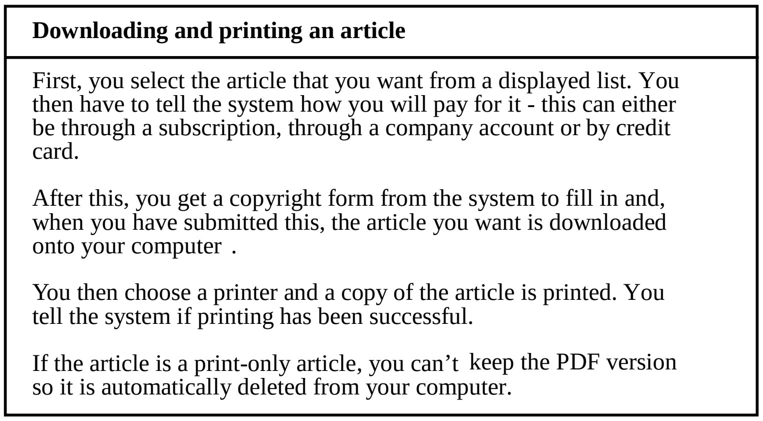
13 Technical Practices of XP

- **Sit together**, the whole team develops in one open space
- **Whole team**, utilize a cross-functional team of all those necessary for success
- **Informative workspace**, place visible wall graphs around the workspace
- **Energized work**, XP teams do not work excessive overtime for long periods of time
- **Pair programming** refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test.
- **User Stories**, the team write short statements of customer-visible functionality
- **Weekly cycle**, weekly meetings held to review progress to date
- **Quarterly cycle**, the whole team should pick a theme or themes of stories for a quarter
- **Slack**, in every iteration, plan some lower-priority tasks that can be dropped if the team gets behind
- …

13 Technical Practices of XP - 2

- …
- **Ten-minute build,** structure project and its tests such that the whole system can be built and all tests run in ten minutes so that the system will be tested often
- **Test-first programming,** all stories have at least one acceptance test. When the acceptance test(s) for a user story all pass, the story is considered to be fulfilled.
- **Continuous integration**, programmers check in the code base completed code and its associated tests several times a day. Code may only be checked in if all its associated unit tests and all of unit tests of the entire code base pass.
- **Incremental design**, rather than an anticipatory detailed design prior to implementation. Invest in the design of the system every day
- Most XP groups use a daily Stand Up meeting to discuss prior day's work, plans for today, and possible problems
  - Why a stand up meeting?

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.

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 organization 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.

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 case description

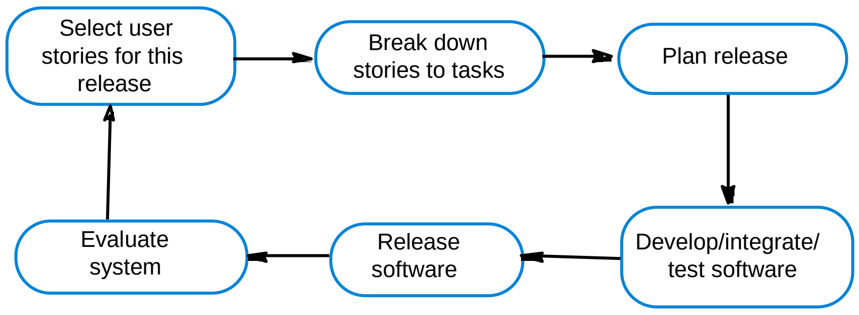
Test 4: Test Credit Card Validity

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

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. (??? – Other studies show 30% cost)

The XP release cycle


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

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.

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.

Heuristics concerning XP

- *XP does not work for all development:*
  - *Team size*: Because the development team needs to be co-located, team size is limited to the number of people that can fit in a single room, generally agreed to be from 2 to 10.
  - *Iteration length*: XP has the shortest recommended iteration length of the Agile Methods under consideration, 2 weeks.
  - *Support for distributed teams*: Because of XP's focus on community and colocation, distributed teams are not supported.
  - *System criticality*: XP is not necessarily geared for one system or another. However, most agree that there is nothing in XP itself that should limit its applicability.

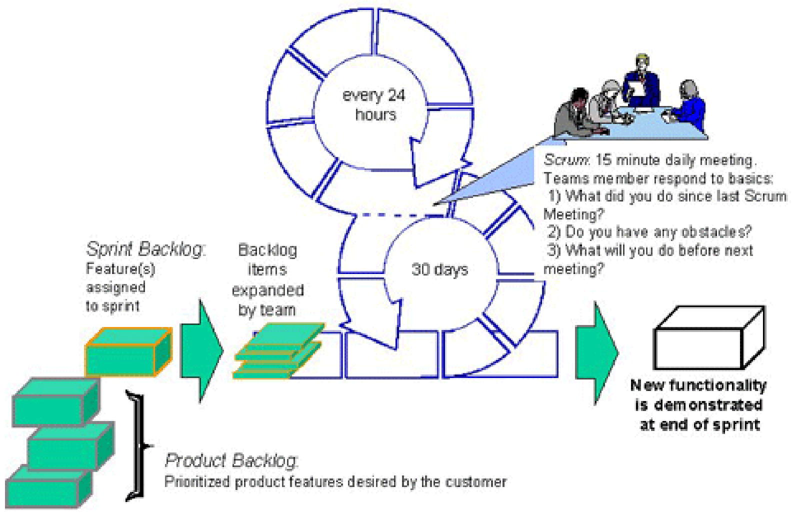
SCRUM -1

- Scrum is an iterative, incremental process for developing any product or managing any work. Scrum is a wrapper for existing engineering practices.
- Without major changes -often within thirty days - teams are building useful, demonstrable product functionality.
- Scrum is a set of interrelated practices and rules that optimize the development environment, reduce organizational overhead, and closely synchronize market requirements with iterative prototypes.

SCRUM-2

- *Pre-sprint planning*: Features and functionality are selected from the release backlog and placed into the "sprint backlog." Since the tasks in the backlog are generally at a higher level of abstraction, pre-sprint planning also identifies a Sprint Goal reminding developers why the tasks are being performed and at which level of detail to implement them
- *Sprint*: Teams are handed their sprint backlog and "told to sprint to achieve their objectives". Tasks in the sprint backlog are frozen and remain unchangeable for the duration of the sprint. Team members choose the tasks they want to work on and begin development. Scrum meetings are held every morning to enhance communication and inform customers, developers, and managers on the status of the project, identify any problems encountered, and keep the entire team focused on a common goal.
- *Post-sprint meeting*: After every sprint, a post-sprint meeting is held to analyze project progress and demonstrate the current system.
- *Team size*: Development personnel are split into teams of up to seven people. A complete team should at least include a developer, quality assurance engineer, and a documenter.
- *Iteration length*: Durations are commonly held at 4 weeks
- *Support for distributed teams*: A project may consist of multiple teams that could easily be distributed.

SCRUM-3


every 24 hours
30 days
Scrum: 15 minute daily meeting.
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

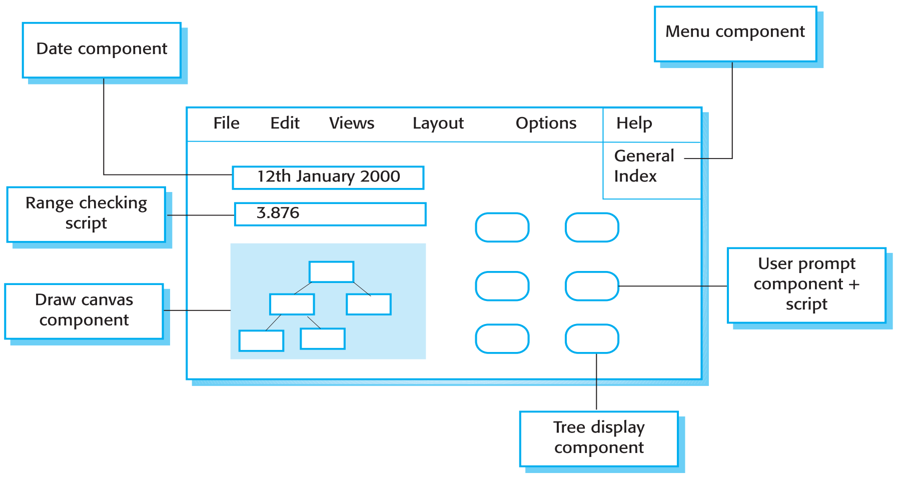
Why SCRUM works

- Small working teams that maximize communication, minimize overhead, and maximize sharing of tacit, informal knowledge.
- Adaptability to technical or marketplace (user/customer) changes to ensure the best possible product is produced.
- Frequent 'builds,' or construction of executables, that can be inspected, adjusted, tested, documented, and built on.
- Partitioning of work and team assignments into clean, low coupling partitions, or packets.
- Constant testing and documentation of a product as it is built.
- Ability to declare a product 'done' whenever required (because the competition just shipped, because the company needs the cash, because the user/customer needs the functions, because that was when it was promised . . .).

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.

Key points

- An iterative approach to software development leads to faster delivery of software.
- Agile methods are iterative development methods that aim to reduce development overhead and so produce software faster.
- Extreme programming includes practices such as systematic testing, continuous improvement and customer involvement.
- The approach to testing in XP is a particular strength where executable tests are developed before the code is written.

Key points

- Rapid application development environments include database programming languages, form generation tools and links to office applications.
- A throw-away prototype is used to explore requirements and design options.
- When implementing a throw-away prototype, start with the requirements you least understand; in incremental development, start with the best-understood requirements.