Software Process

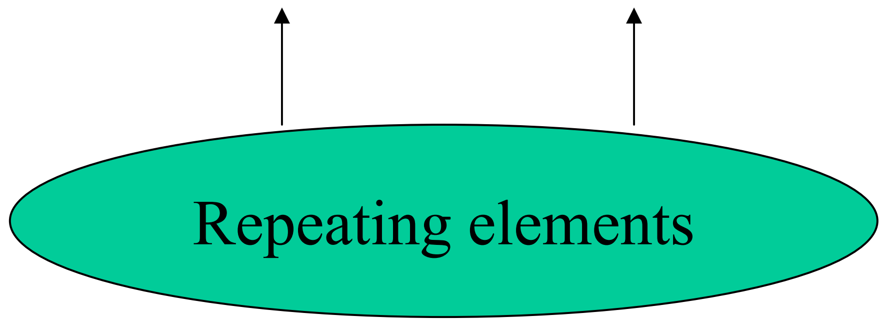
Process: A sequence of activities, subject to constraints on resources, that produce an intended output of some kind.

Any process has these characteristics:

- The process prescribes all the major activities
- The process uses resources, subject to certain constraints,and produces products, both intermediate and final.
- The process can be composed of linked sub-processes.
- Each process activity has entry and exit criteria.
- The activities are organized in a sequence.
- Every process has a set of governing principles.
- Constraints may apply to an activity, resource, or product.

Processes and Activities

Process1 = <A1, A2, A4, A6, A2, A1, A8>



Activities can be modeled in different ways, e.g., as transitions between states.

Activities can be classified as belonging to certain stages.

Software development stages

- Requirements analysis and definitions
- system design
- program design
- program implementation
- unit testing
- system testing
- system delivery
- maintenance

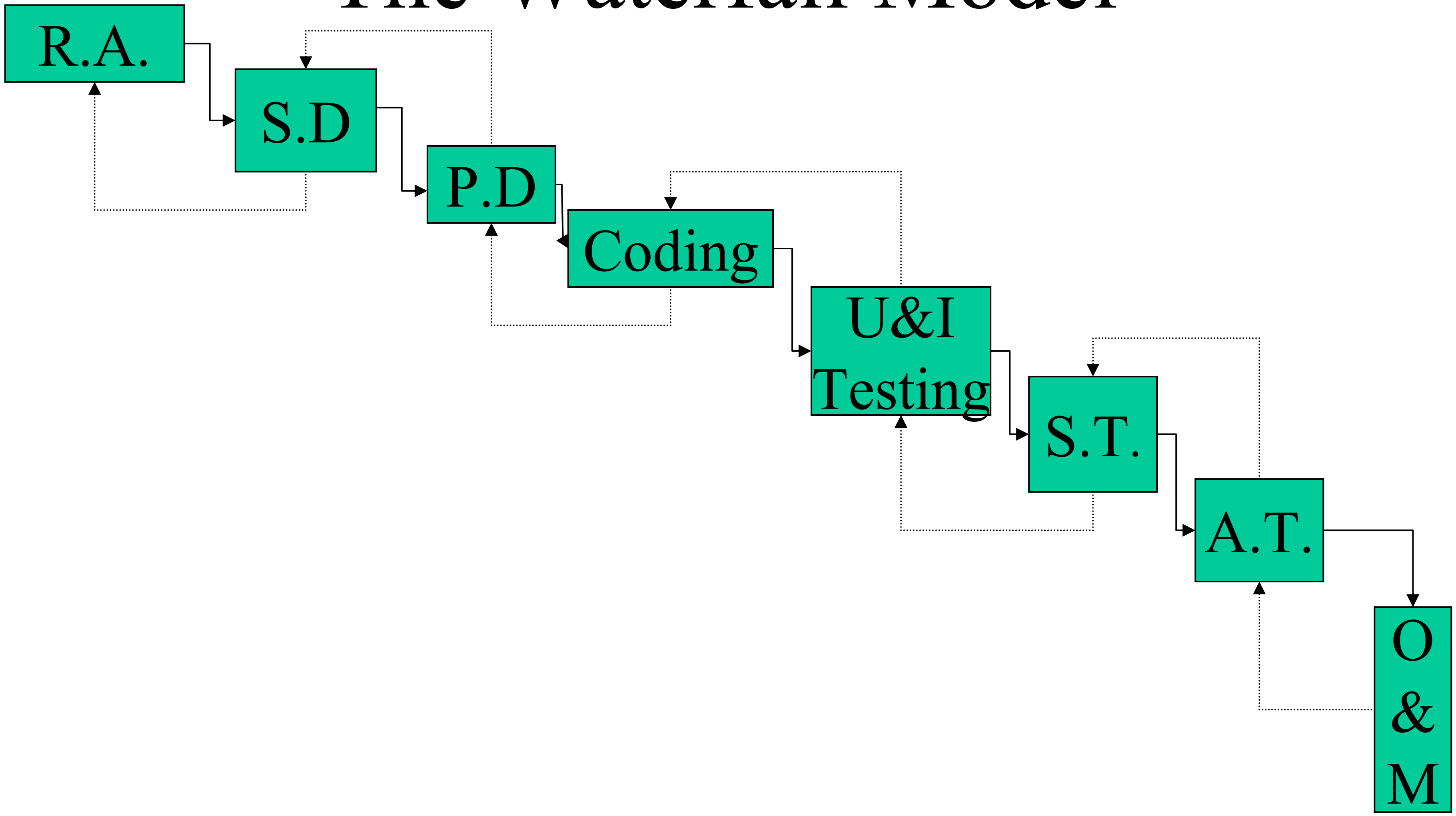
Processes and Development Models

A development model specifies the sequence in which development stages occur.

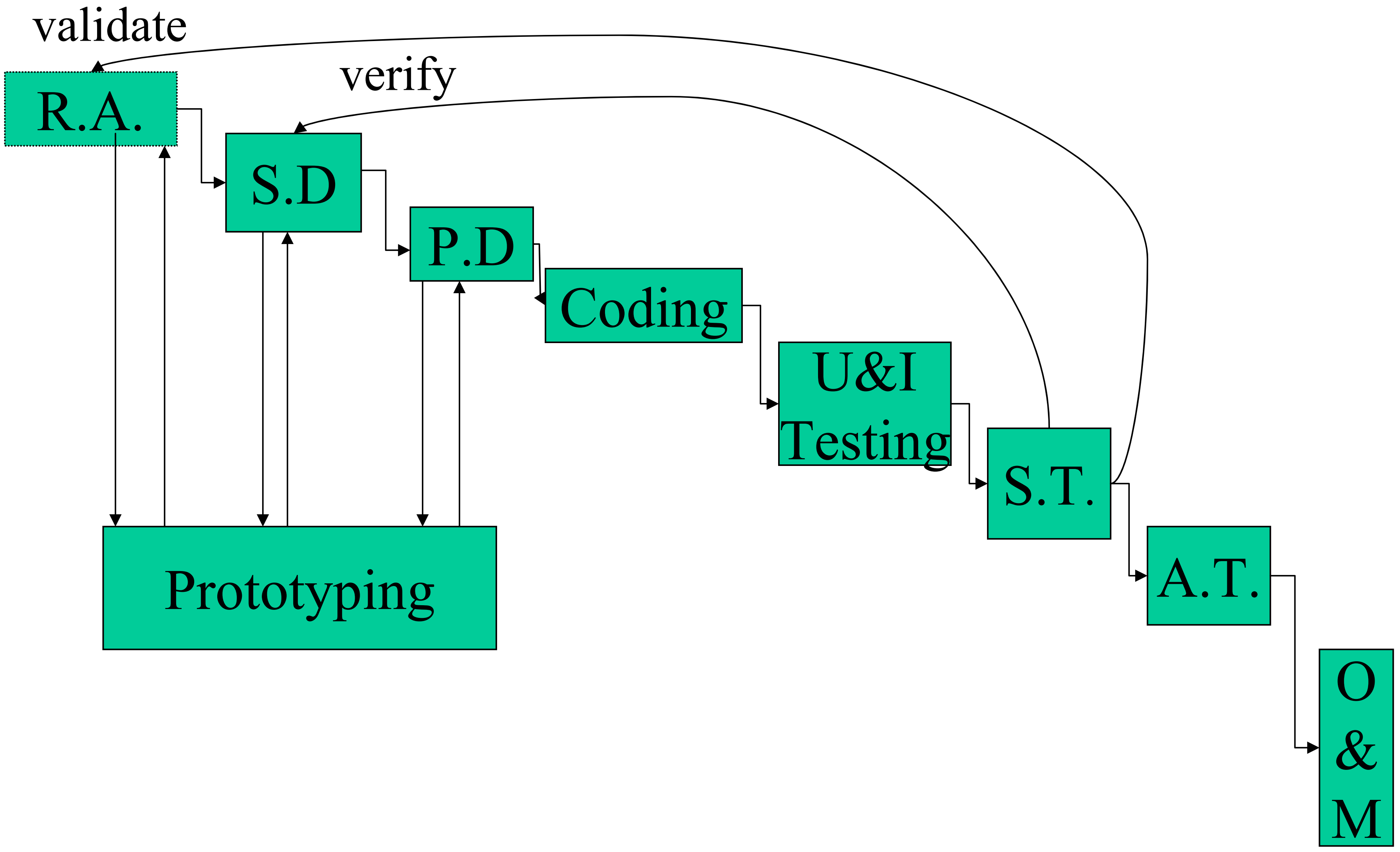
Example:
<S1, S2, S3, S2, S3,…,S2, S3, S4, S5, S4, S6, S7, S8>

A software development model is an abstract specification of a software process. Many concrete processes can satisfy the same model.

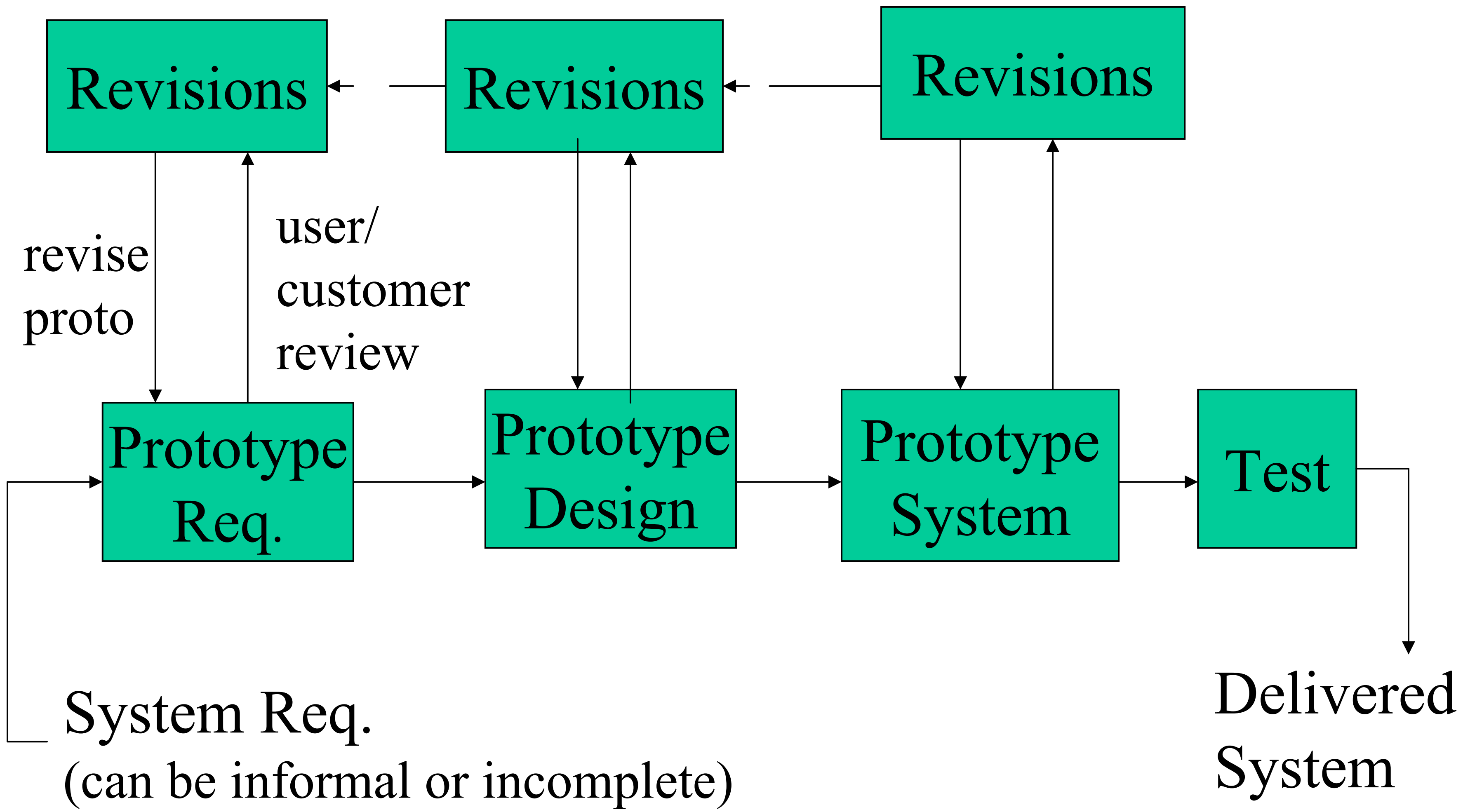
The Waterfall Model



W. Model with Prototyping



Prototyping Model



one-shot, incremental and evolutionary approaches

- one-shot
  - the whole application is implemented in one go
  - For example, the waterfall model
- incremental
  - application is implemented in steps
  - each step delivers a subset of the functions
  - functions in the subset are fully implemented i.e. can be used by client...

one-shot, incremental and evolutionary approaches contd.

- evolutionary
  - the system is implemented via a number of versions
  - each version is ‘exercised’ by users and suggestions for improvement made

‘rules of thumb’ about approach to be used

IF uncertainty is high

THEN use evolutionary approach

IF complexity is high but uncertainty is not

THEN use incremental approach

IF uncertainty and complexity both low

THEN use one-shot

IF schedule is tight

THEN use evolutionary or incremental

combinations of approach

installation

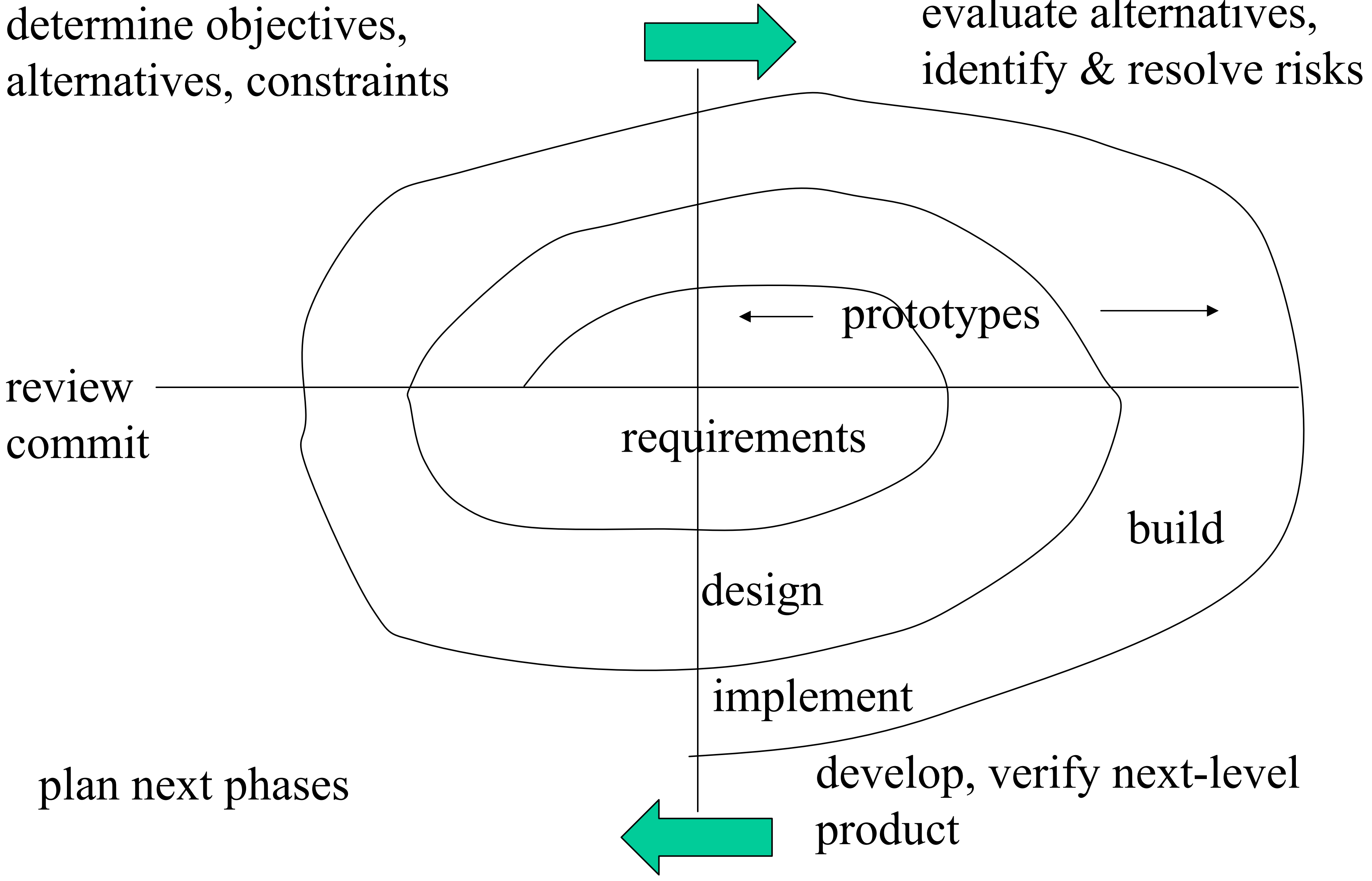
construction

	one-shot	incremental	evolutionary
one-shot	yes	yes	**no**
incremental	yes	yes	**no**
evolutionary	yes	yes	yes

- one-shot or incremental installation - any construction approach possible

- evolutionary installation implies evolutionary construction

spiral model


determine objectives, alternatives, constraints
evaluate alternatives, identify & resolve risks
prototypes
review
commit
requirements
build
design
implement
plan next phases
develop, verify next-level product

spiral model

- could be seen as another view of waterfall model
- at each stage of the development project a greater level of detail is considered
- more knowledge is gathered
- at end of each stage review scope and risk and decide whether to commit to the next stage
- see Boehm’s *A spiral model of software development and enhancement*

‘Agile’ methods

structured development methods have some perceived disadvantages

- produce large amounts of documentation which can be largely unread
- documentation has to be kept up to date
- division into specialist groups and need to follow procedures stifles communication
- users can be excluded from decision process
- long lead times to deliver anything etc. etc

The answer? ‘Agile’ methods?

Dynamic system development method

- UK-based consortium
- DSDM is more a project management approach than a development approach

Nine core DSDM principles

1. Active user involvement
2. Teams empowered to make decisions
3. Frequent delivery of products
4. Fitness for *business purpose*
5. Iterative and incremental delivery
6. Changes are reversible
7. Requirements base-lined at a *high level*
8. Testing integrated with development
9. Collaborative and co-operative approach

Key indicators for DSDM

- visibility of functionality at user interface
- clear identification of all classes of user
- not too much complexity
- not large applications - split into components
- need for time constraints
- flexible high-level requirements

time-boxing

- *time-box* fixed deadline by which *something* has to be delivered
- typically two to six weeks
- MOSCOW priorities
  - Must have - essential
  - Should have - very important, but system could operate without
  - Could have
  - Want - but probably won’t get!

Extreme programming-1

- Associated with Kent Beck - see *Extreme programming explained*
- Developed originally on Chrysler C3 payroll (Smalltalk) project
- *Agile methods* include Jim Highsmith’s *Adaptive Software Development* and Alistair Cocburn’s Chrystal Lite methods

Extreme programming-2

- increments of one to three weeks
  - customer can suggest improvement at any point
- argued that distinction between design and building of software are artificial
- code to be developed to meet current needs only
- frequent re-factoring to keep code structured

extreme programming-3

- developers work in pairs
- test cases and expected results devised before software design
- after testing of increment, test cases added to a consolidated set of test cases

XP vs. Traditional Models

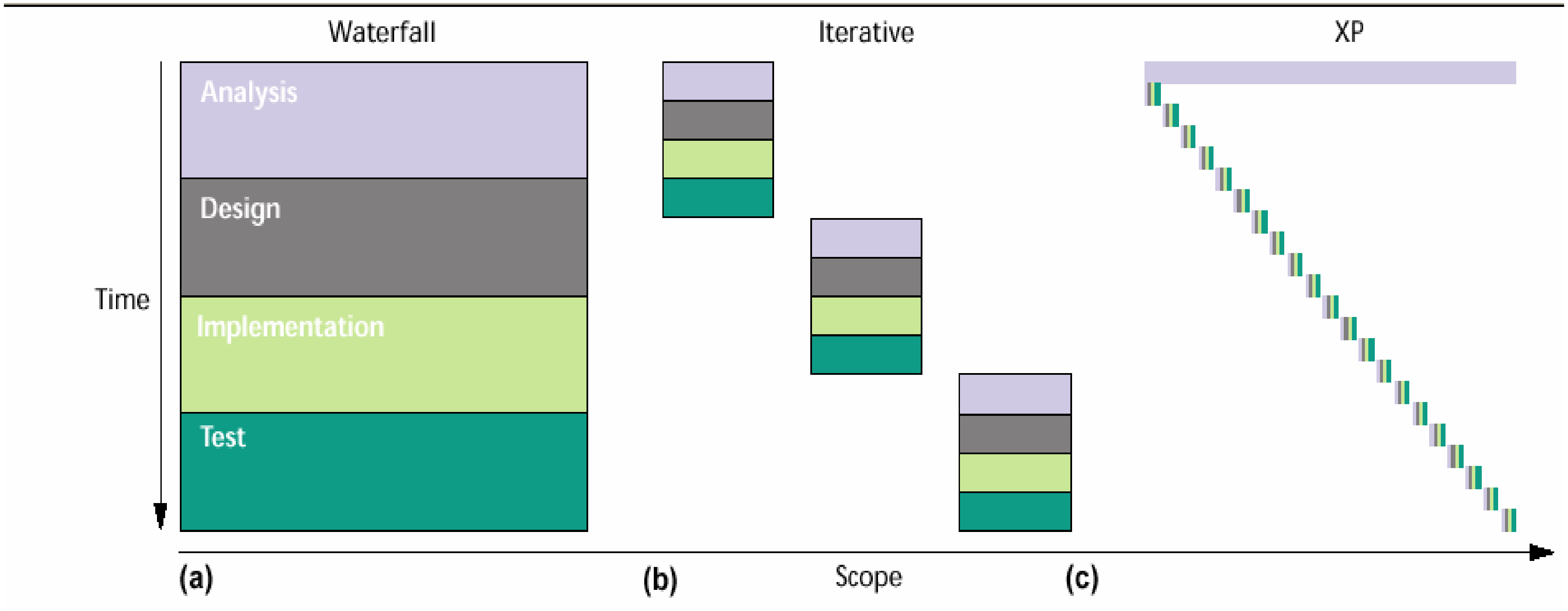


Figure 1. The evolution of the Waterfall Model (a) and its long development cycles (analysis, design, implementation, test) to the shorter, iterative development cycles within, for example, the Spiral Model (b) to Extreme Programming's (c) blending of all these activities, a little at a time, throughout the entire software development process.

When is XP Appropriate?

- Small to medium-size projects.
- The more dynamic the environment, the better.
  - Frequently changing requirements.
  - High risk, tight deadline.
- Build times are short (a few minutes).
- Testing can be automated.

Does XP Really Work?

- Shown to improve quality, increase productivity, and reduce cost.
- Reduces bugs by at least 15%.
- Pair programming benefits outweigh cost.
  - Development time cost of 15%.
  - Improves design and code quality.
  - Reduces staffing risks.
  - “..more enjoyable at statistically significant levels”.

Weaknesses of XP

- Does not scale well:
  - Communication gets complicated.
  - Integration becomes a bottleneck.
- Counter to business culture:
  - Plans are very dynamic.
  - Overtime indicates a problem with the process—not dedication.
- Requires supporting technology:
  - Automated testing.
  - Flattened cost-of-change curve.

Grady Booch's concern

Booch, an OO authority, is concerned that with requirements driven projects:

'Conceptual integrity sometimes suffers because this is little motivation to deal with scalability, extensibility, portability, or reusability beyond what any vague requirement might imply'

Tendency towards a large number of discrete functions with little common infrastructure?

Some Questions about Models

- Why are some sequences of development stages chosen as software dev. models?
- What use are software dev. models?
- Why are management activities such as cost estimation not represented in these models?
- Can you devise a software dev. model in which management activities are explicitly represented?