

## Software Prototyping (プロトタイピング)

---

- ◆ Animating and demonstrating system requirements

## Uses of system prototypes

---

- ◆ The principal use is to help customers and developers understand the requirements for the system
- ◆ The prototype may be used for user training before a final system is delivered
- ◆ The prototype may be used for back-to-back testing

## 注) Back-to-Back Testing

---

- ◆ 訳語？
- ◆ 22章で説明予定
- ◆ 複数のバージョンのシステムの出力を比較するテスト
- ◆ 目的
  - デグレード(degrade: 以前動いていた機能が新しい版で動かなくなる現象)を防ぐ
  - テスト作業自体の簡便化

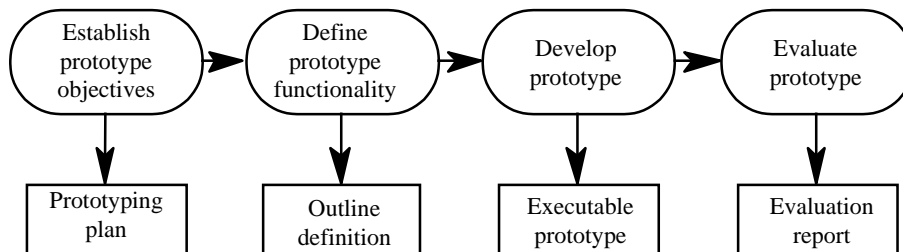
## Prototyping benefits

---

- ◆ Misunderstandings between software users and developers are exposed
- ◆ Missing services may be detected
- ◆ 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

## Prototyping process

---



©Ian Sommerville 1995

Software Engineering, 5th edition. Chapter 8

Slide 7

## Prototyping objectives (発展型と使い捨て型)

---

- ◆ The objective of **evolutionary prototyping** 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.

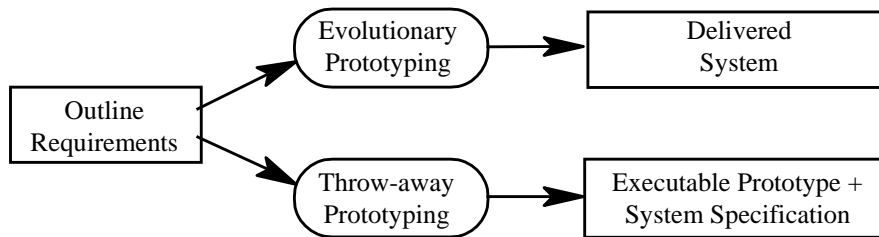
©Ian Sommerville 1995

Software Engineering, 5th edition. Chapter 8

Slide 8

## Approaches to prototyping

---



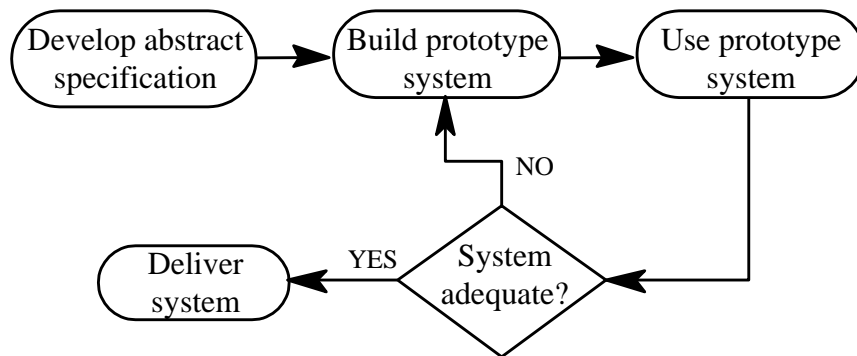
## Evolutionary prototyping

---

- ◆ Must be used for systems where the specification cannot be developed in advance, e.g., AI systems and user interface systems
- ◆ Based on techniques which allow rapid system iterations
- ◆ Verification is impossible as there is no specification. Validation means demonstrating the adequacy of the system

## Evolutionary prototyping

---



## Evol. prototyping problems

---

- ◆ Existing management processes assume a waterfall model of development
- ◆ Continual change tends to corrupt system structure so long-term maintenance is expensive
- ◆ Specialist skills are required which may not be available in all development teams
- ◆ Organisations must accept that the lifetime of systems developed this way will inevitably be short

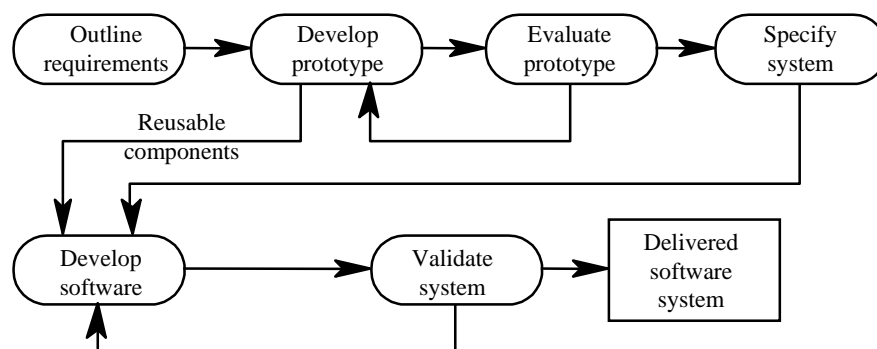
## Throw-away prototyping

---

- ◆ 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

## Throw-away prototyping

---



## 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

## Prototyping techniques

---

- ◆ Executable specification languages
- ◆ Very high-level languages
- ◆ Application generators and 4GLs
- ◆ Composition of reusable components

## Executable specification languages

- ◆ The system is specified in a formal language
- ◆ This specification is processed and an executable system is automatically generated
- ◆ At the end of the process, the specification may serve as a basis for a re-implementation of the system

## Problems with this approach

- ◆ Graphical user interfaces cannot be prototyped
- ◆ Formal specification development is not a rapid process
- ◆ The executable system is usually slow and inefficient
- ◆ Executable specifications only allow functional requirements to be prototyped



## Very 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

## Prototyping languages

---

Language	Type	Application domain
Smalltalk	Object-oriented	Interactive systems
LOOPS	Wide spectrum	Interactive systems
Prolog	Logic	Symbolic processing
Lisp	List-based	Symbolic processing
Miranda	Functional	Symbolic processing
SETL	Set-based	Symbolic processing
APL	Mathematical	Scientific systems
4GLs	Database	Business DP
CASE tools	Graphical	Business DP

## Smalltalk

---

- ◆ Very powerful system for prototyping interactive systems
- ◆ Object-oriented language so systems are resilient to change
- ◆ The Smalltalk environment objects are available to the prototype developer
- ◆ The system includes support software such as graphical user interface generation tools

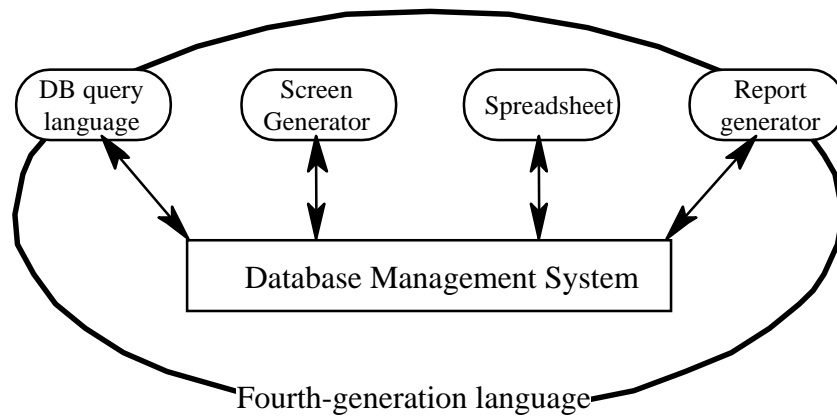
## Fourth-generation languages

---

- ◆ 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
- ◆ Cost-effective for small to medium sized business systems

## 4GLs

---

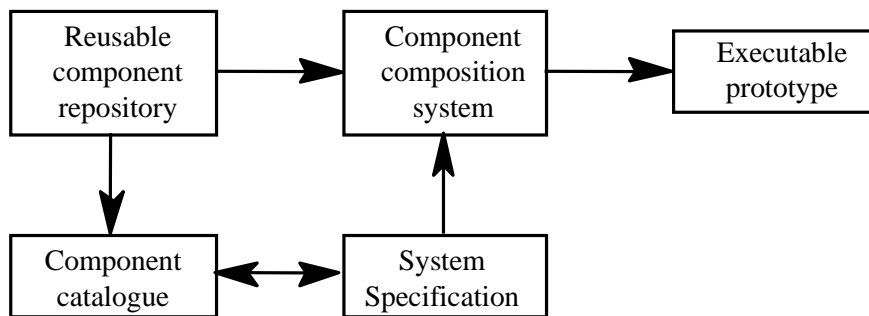


## Prototyping with reuse

---

- ◆ The system is prototyped by ‘gluing’ together existing components
- ◆ Likely to become more widely used as libraries of objects become available
- ◆ Needs a composition language such as a Unix shell language
- ◆ Visual Basic is largely based on this approach

## Reusable component composition



©Ian Sommerville 1995

Software Engineering, 5th edition. Chapter 8

Slide 27

## User interface prototyping

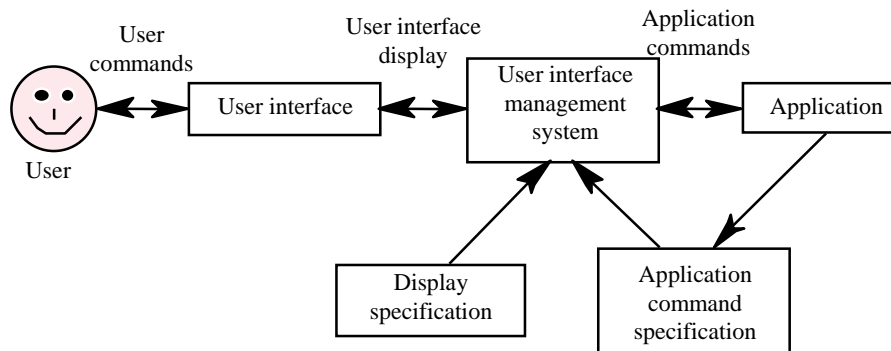
- ◆ It is impossible to pre-specify the look and feel of a user interface in an effective way. prototyping is essential
- ◆ UI development consumes an increasing part of overall system development costs
- ◆ Prototyping may use very high level languages such as Smalltalk or Lisp
- ◆ User interface generators may be used to 'draw' the interface and simulate its functionality

©Ian Sommerville 1995

Software Engineering, 5th edition. Chapter 8

Slide 28

# User interface management system



UIMS: アプリケーションとUIの分離