**Software Prototyping**

- 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.

**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 Objectives

- 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.

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

Build prototype system
Develop abstract specification
Use prototype system

Deliver system
System adequate?

YES

NO

Evolutionary 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.
- Organizations 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 Process**

- Outline requirements
- Develop prototype
- Evaluate prototype
- Specify system
- Develop software
- Validate system
- Delivered software system

**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.

**Incremental Development**
- System is developed and delivered in increments after establishing an overall architecture.
- Users may experiment with delivered increments while others are being developed.
- Intended to combine some of the advantages of prototyping but with a more manageable process and better system structure.
Incremental Development Process

1. Define system deliverables
2. Design system architecture
3. Specify system increment
4. Build system increment
5. Validate increment
6. Integrate increment
7. Validate system
8. Deliver final system
9. System complete?
   - NO
   - YES

Prototyping Techniques

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

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 (e.g., Java Foundation Classes).
- Some languages have an integrated support environment whose facilities may be used in the prototype.
### Prototyping Languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Type</th>
<th>Application domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smalltalk</td>
<td>Object-oriented</td>
<td>Interactive systems</td>
</tr>
<tr>
<td>LOOPS</td>
<td>Wide spectrum</td>
<td>Interactive systems</td>
</tr>
<tr>
<td>Prolog</td>
<td>Logic</td>
<td>Symbolic processing</td>
</tr>
<tr>
<td>Lisp</td>
<td>List-based</td>
<td>Symbolic processing</td>
</tr>
<tr>
<td>Miranda</td>
<td>Functional</td>
<td>Symbolic processing</td>
</tr>
<tr>
<td>SETL</td>
<td>Set-based</td>
<td>Symbolic processing</td>
</tr>
<tr>
<td>APL</td>
<td>Mathematical</td>
<td>Scientific systems</td>
</tr>
<tr>
<td>4GLs</td>
<td>Database</td>
<td>Business DP</td>
</tr>
<tr>
<td>CASE tools</td>
<td>Graphical</td>
<td>Business DP</td>
</tr>
</tbody>
</table>

*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 tool set.
- Cost-effective for small to medium sized business systems.
4GLs

- **DB query language**
- **Screen Generator**
- **Spreadsheet**
- **Report generator**

Fourth-generation language

---

**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**

- **Reusable component repository**
- **Component catalogue**
- **System Specification**
- **Executable prototype**
- **Component composition system**
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.
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, Java, Visual Basic, or Lisp.
- User interface generators may be used to “draw” the interface and simulate its functionality.