Chapter 8

Chapter 8 Software Prototyping

Learning Objective

... Animating and demonstrating system requirements

Frederick T Sheldon
Assistant Professor of Computer Science
Washington State University

Objectives

- To describe the use of prototypes in requirements validation
- To discuss evolutionary and throw-away prototyping
- To introduce rapid prototyping techniques
- To explain the need for user interface prototyping

Topics covered

- Prototyping in the software process
- Prototyping techniques
- User interface prototyping
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 process

1. Establish prototype objectives
2. Define prototype functionality
3. Develop prototype
4. Evaluate prototype

- Prototyping plan
- Outline definition
- Executable prototype
- Evaluation report
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

- Develop abstract specification
- Build prototype system
- Use prototype system
- Deliver system
- System adequate?
- NO
- YES

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

Outline requirements → Develop prototype → Evaluate prototype → Specify system → Deliverable software system

Reusable components

Develop software → Validate 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. Therefore, these serve as a form of prototype system
- Intended to combine some of the advantages of prototyping but with a more manageable process and better system structure
Incremental development process

Define system deliverables

Design system architecture

Specify system increment

Build system increment

Validate increment

Integrate increment

Deliver final system

Validate system

Yes

System complete?

NO

Validate increment

Build system increment

Specify system increment

Design system architecture

Define system deliverables

YES

System complete?

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

<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 toolset
- Cost-effective for small to medium sized business systems

4GLs

- DB query language
- Screen Generator
- Spreadsheet
- Report generator
- Database Management System

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

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
User interface management system

User commands → User interface display → User interface management system → Application commands

User interface display

Display specification

Application command specification

Key points

- A prototype can be used to give end-users a concrete impression of the system's capabilities
- Prototyping may be evolutionary prototyping or throw-away prototyping
- Rapid development is essential for prototype systems
- Prototype structures become corrupted by constant change. Hence, long-term evolution is difficult

Key points

- In a throw-away prototype start with the least well-understood parts; in an evolutionary prototype, start with the best understood parts
- Prototyping methods include the use of executable specification languages, very high-level languages, fourth-generation languages and prototype construction from reusable components
- Prototyping is essential for parts of the system such as the user interface which cannot be effectively pre-specified