软件原型

- 为了需求有效性验证的快速软件开发

目标

- 在不同类型的开发项目中，原型开发的作用和地位；
- 进化式和抛弃式原型开发
- 三种快速原型开发技术，即高级语言开发、数据库编程、组件复用
- 用户界面原型开发

内容

- 软件过程中的原型开发
- 快速原型技术
- 用户界面原型开发

系统原型

- 原型开发是系统的快速开发
- 开发完成的系统总是比需求的系统要差。
- 由于很多系统采用进化式开发方法，原型和正常系统之间的界限变得模糊

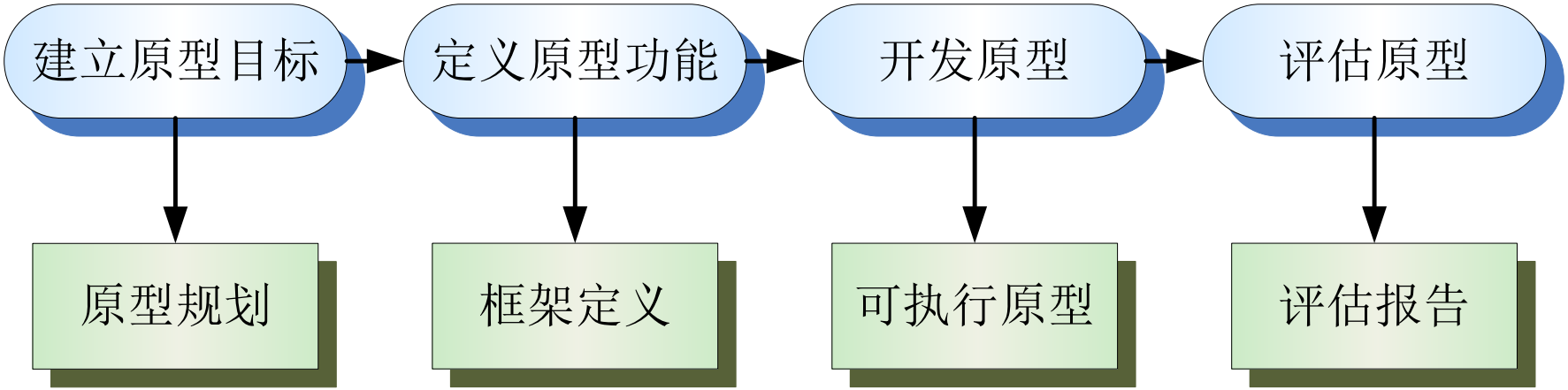
系统原型的作用

- 主要作用是帮助客户和开发人员理解系统需求
  - 需求导出. 用户可以用原型做实验以观察系统是如何支撑他们的工作的。
  - 需求有效性验证. 原型可以暴露出错误和遗漏的东西
- 原型开发可以看作是一个降低需求风险的措施

原型的好处

- 暴露出软件用户和开发人员之间的理解偏差
- 可以发现需求的不完善和不一致
- 在软件过程的早期可以得到一个能工作的系统
- 原型可以作为得出系统描述的基础
- 原型系统可以支持用户培训和系统测试

原型开发过程



原型开发的优点

- 提高系统可用性
- 更加接近真正需要的系统
- 提高设计的质量
- 提高可维护性
- 减少了整体开发投入

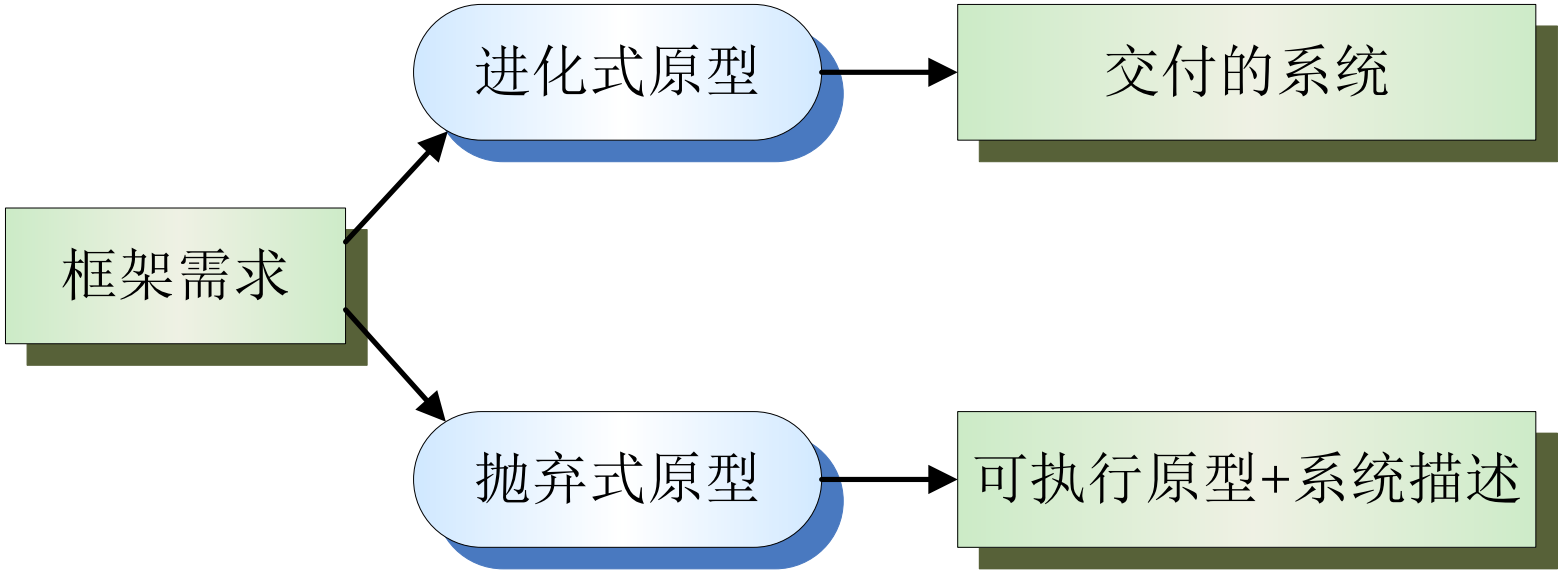
软件过程中的原型开发

- 进化式原型开发
  - 开发一个初始的原型，然后通过几个阶段的精炼修改，得到最终系统
- 抛弃式原型开发
  - 原型是用来发现系统需求，然后是抛弃掉的。正常的系统是重新开发的。

原型开发的目标

- *进化式原型开发*是要交付一个工作系统给最终用户。开发从那些理解得最好的需求开始。
- *抛弃式原型开发*是为了导出和验证系统需求。原型开发过程从那些理解得最差的需求开始。

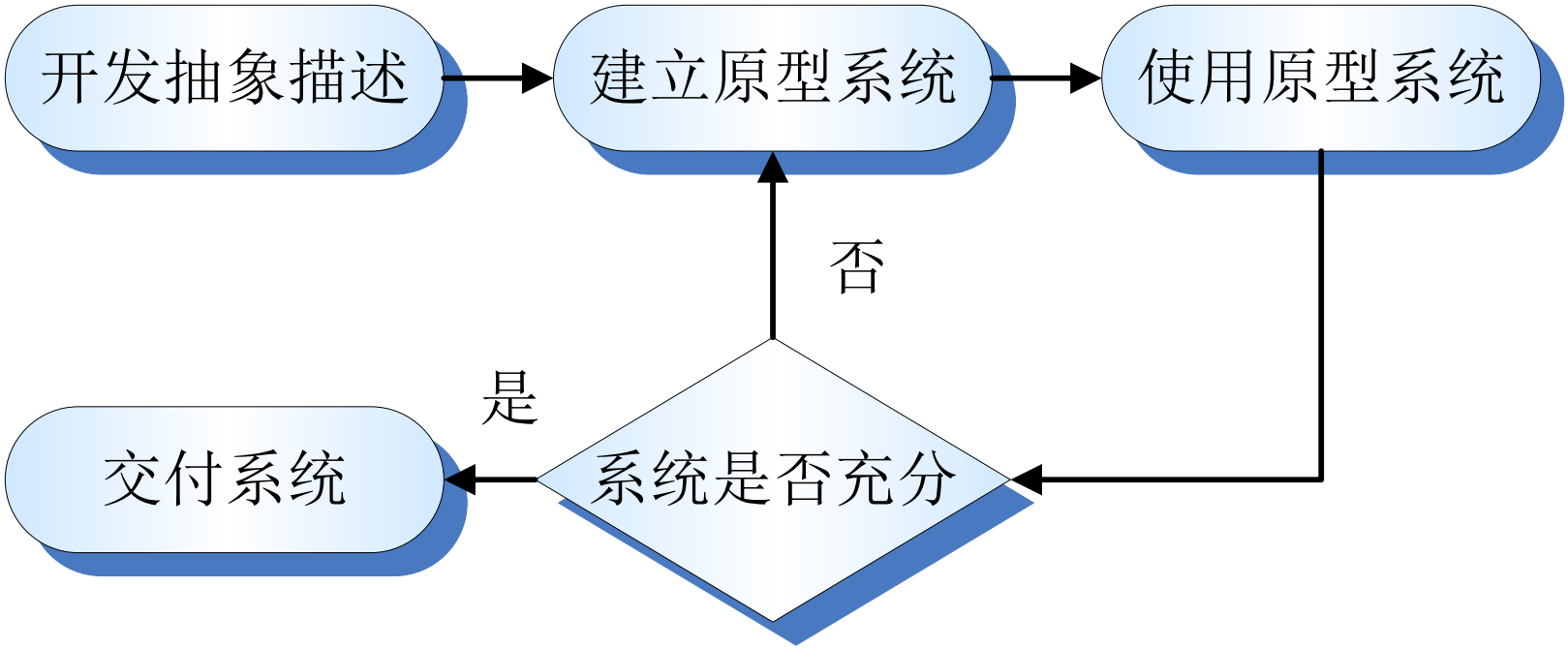
原型开发的路线



进化式原型开发

- 用户系统描述无法再进一步提出的系统。举例来说，有AI系统、人机界面系统
- 基于支持快速系统开发的技术
- 由于不存在对原型的详细描述，所以无法做一致性检验。有效性验证意味着论证系统的充分性

进化式原型开发



进化式原型开发的优点

- 加快系统交付
  - 快速交付和开发有时候比功能完备或者保证长期可维护性更加重要
- 用户的参与
  - 还可使系统更好地满足用户需求，使得用户更加愿意使用系统

进化式原型开发

- 描述、设计和实现交织在一起
- 系统以一系列增量的形式逐步交付给客户
- 使用快速系统开发的技术，如CASE工具和第4代语言（4GL）
- 用户界面通常使用一个GUI开发工具

进化式原型开发的问题

- 管理问题
  - 现有的管理过程都假定是瀑布模型，原型开发太快带来问题
  - 原型开发需要的技术可能不具备、不熟悉
- 维护问题
  - 持续的变更会导致系统崩溃，这样长期的维护会比较困难
- 契约问题
  - 没有完整的系统描述很难拟定一个有关系统开发的合同

原型作为描述的问题

- 需求的某些部分(例如安全性要求很高的功能)可能无法建立原型，这样就无法出现在描述中
- 一个原型系统无法作为法律上的合约
- 在系统原型中，非功能性需求无法充分测试

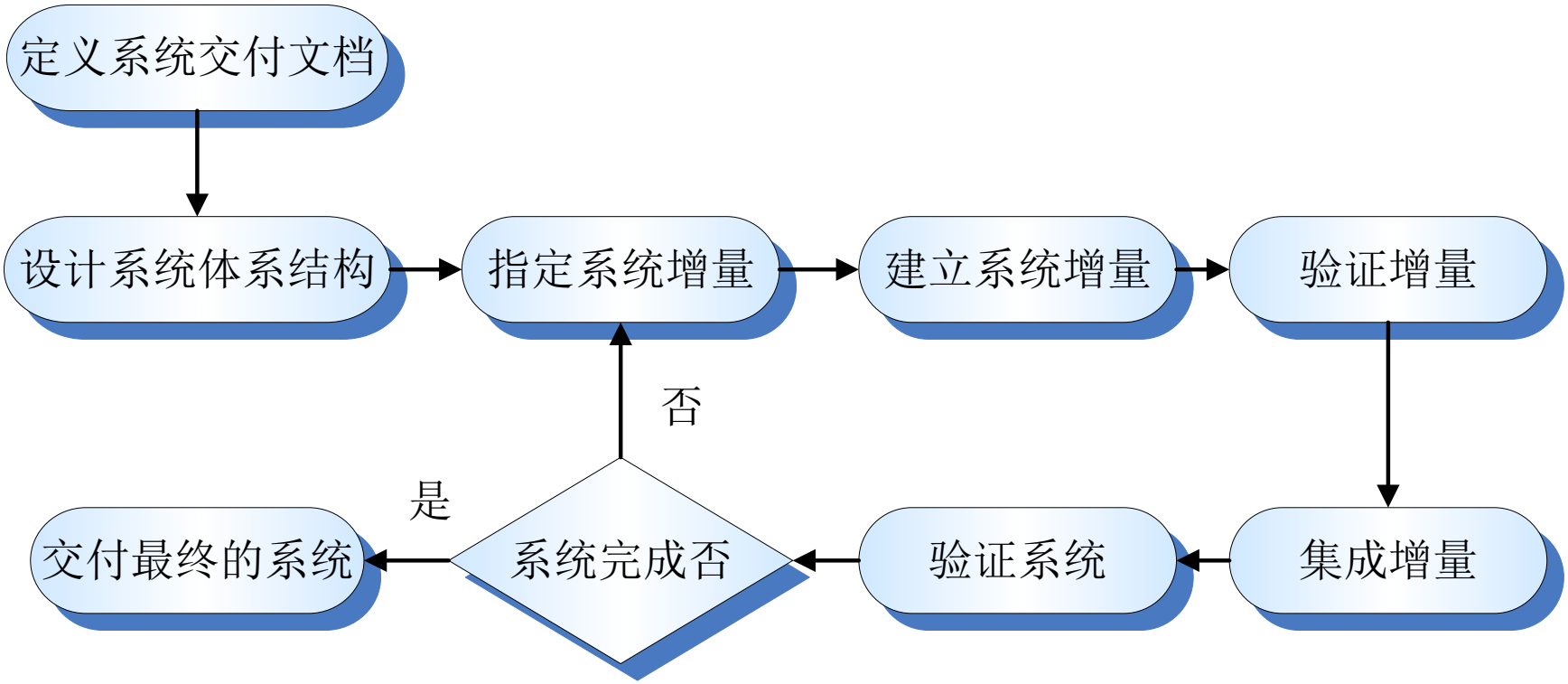
原型只能验证一部分需求

增量式开发

- 系统先建立一个大致的框架，然后以增量形式开发并交付给客户
- 要为每份增量生成需求和描述
- 用户可以在已经提交的增量系统上实验，这一增量系统也可以看成是一种原型系统
- 吸取了原型开发的优点，同时过程更好管理、系统结构更好

原型开发只是不断地改善

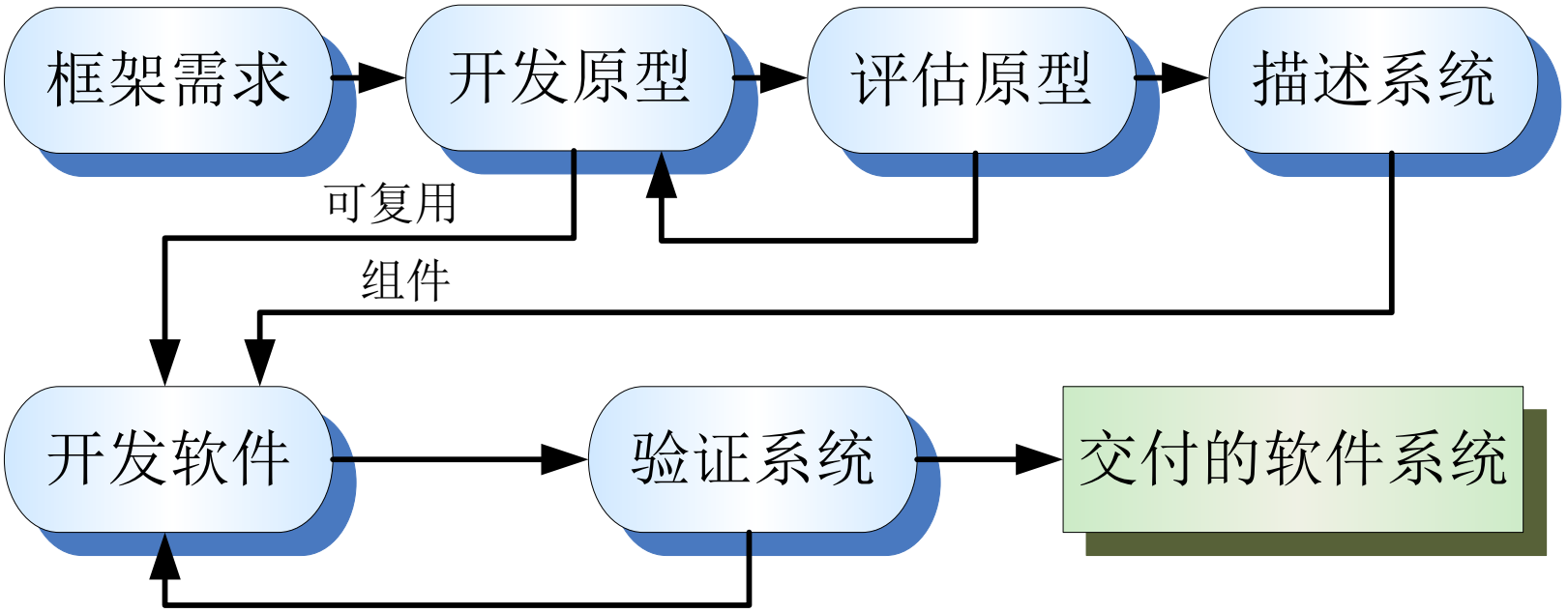
增量式开发过程


定义系统交付文档
设计系统体系结构
指定系统增量
建立系统增量
验证增量
否
是
交付最终的系统
系统完成否
验证系统
集成增量

抛弃式原型

- 用来降低需求风险
- 原型从一个初始描述开发，交付后用作实验，然后抛弃
- 抛弃式原型不能看作是最后系统
  - 有些系统特征可能没有考虑
  - 关于长期维护没有描述
  - 系统结构很差，难以维护

抛弃式原型开发



原型交付

- 开发人员在压力下，可能交付抛弃式原型系统作为用户使用
- 不推荐这种做法
  - 无法调整原型系统以满足非功能性需求
  - 原型系统由于快速开发，必然没有文档
  - 在开发中的变更可能破坏了系统结构
  - 机构内的质量标准对原型往往不加限制

快速原型开发技术

- 不同的技术可能应用于快速开发
  - 动态高级语言开发
  - 数据库编程
  - 组件和应用集成
- 这些技术不是单独的，经常同时使用
- 可视化编程技术应用于大多数原型系统开发中

动态高级语言

- 包含强大的数据管理功能的语言
- 需要一个大型的运行时支持系统，一般不用于大型系统开发
- 某些语言提供了强大的用户界面开发功能
- 某些语言具有集成支持环境

原型开发语言

语言	类型	应用领域
Smalltalk	面向对象	交互式系统
Java	面向对象	交互式系统
Prolog	逻辑	符号处理
Lisp	基于列表	符号处理

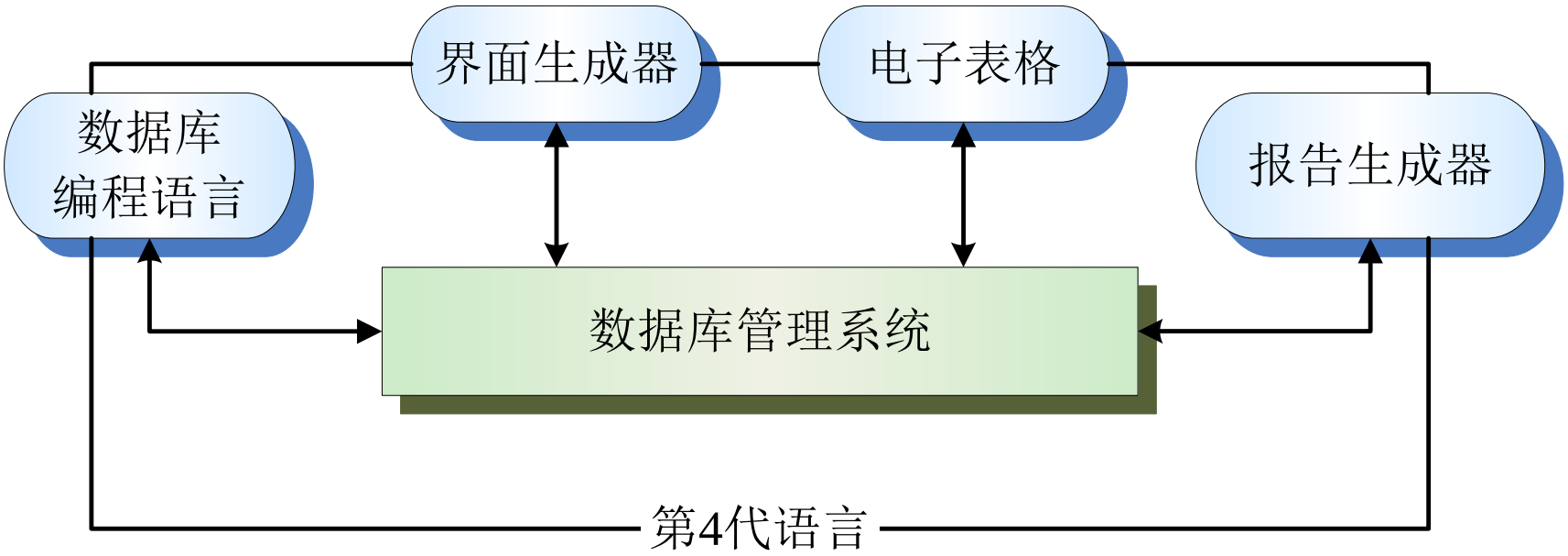
原型开发语言的选择

- 问题的应用领域是什么?
- 需要有什么样的用户交互?
- 语言具有什么样的支持环境?
- 系统的不同部分可能用不同的语言编程。然而，语言间的通信存在问题

数据库编程语言

- 基于数据库管理系统的商业系统领域所特有的语言
- 一般包含一个数据库查询语言、一个界面生成器、一个报表生成器.
- 可能和一个CASE工具集集成在一起
- 语言 + 环境通常叫做第四代语言 (4GL)
- 对中小型商业系统来说是成本效益较好的

数据库编程



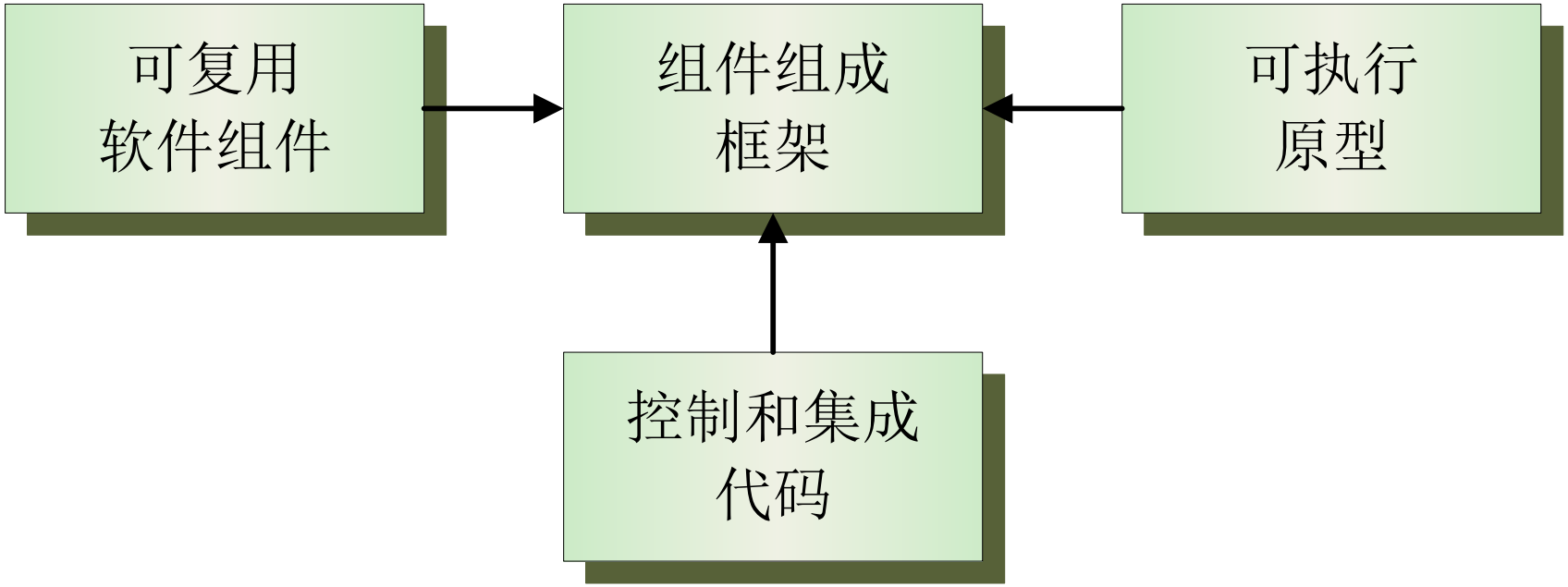
组件和应用集成

- 原型可以从一组可复用的组件、加上一些将组件结合在一起的机制，快速地生成。
- 结合机制应包含控制机制和组件通信的机制
- 系统描述应考虑现有组件的可用性和功能性

基于复用的原型开发

- 应用级开发
  - 整个应用系统集成到原型中，功能可共享
  - 比如，需要文本处理时，可以使用一个标准的文字处理程序
- 组件级开发
  - 个别组件集成在一个标准框架中以完成系统
  - 框架可以是一种脚本语言或者是一个集成的平台，如CORBA

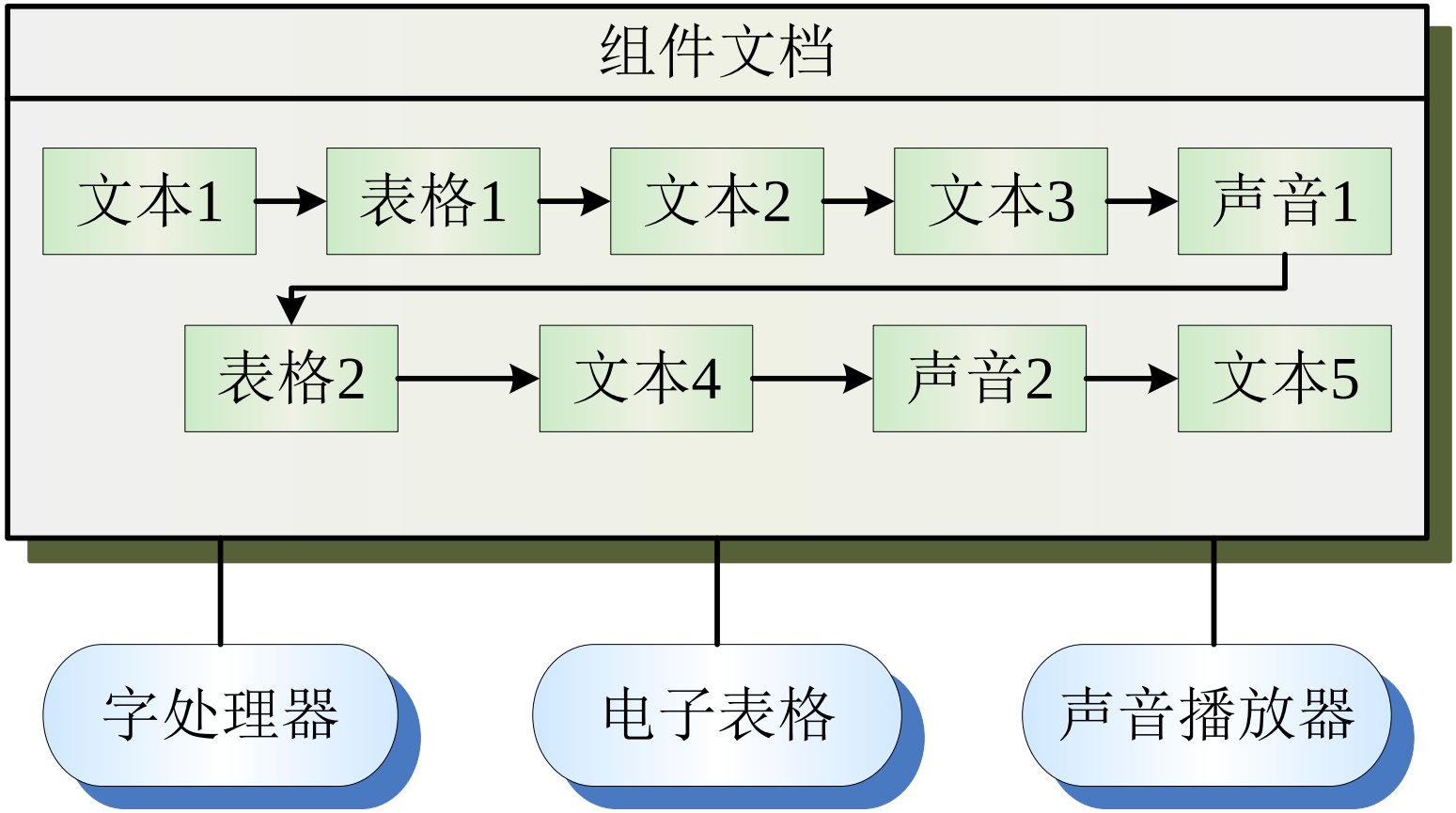
可复用组件的结合



复合文档

- 对某些应用而言，一个原型可以通过开发复合文档来产生
- 这是包含了活动元素（例如电子数据表格）的文档
- 每个活动元素有一个关联的程序，当活动元素被选中时，调用该程序
- 文档本身是不同应用程序的集成

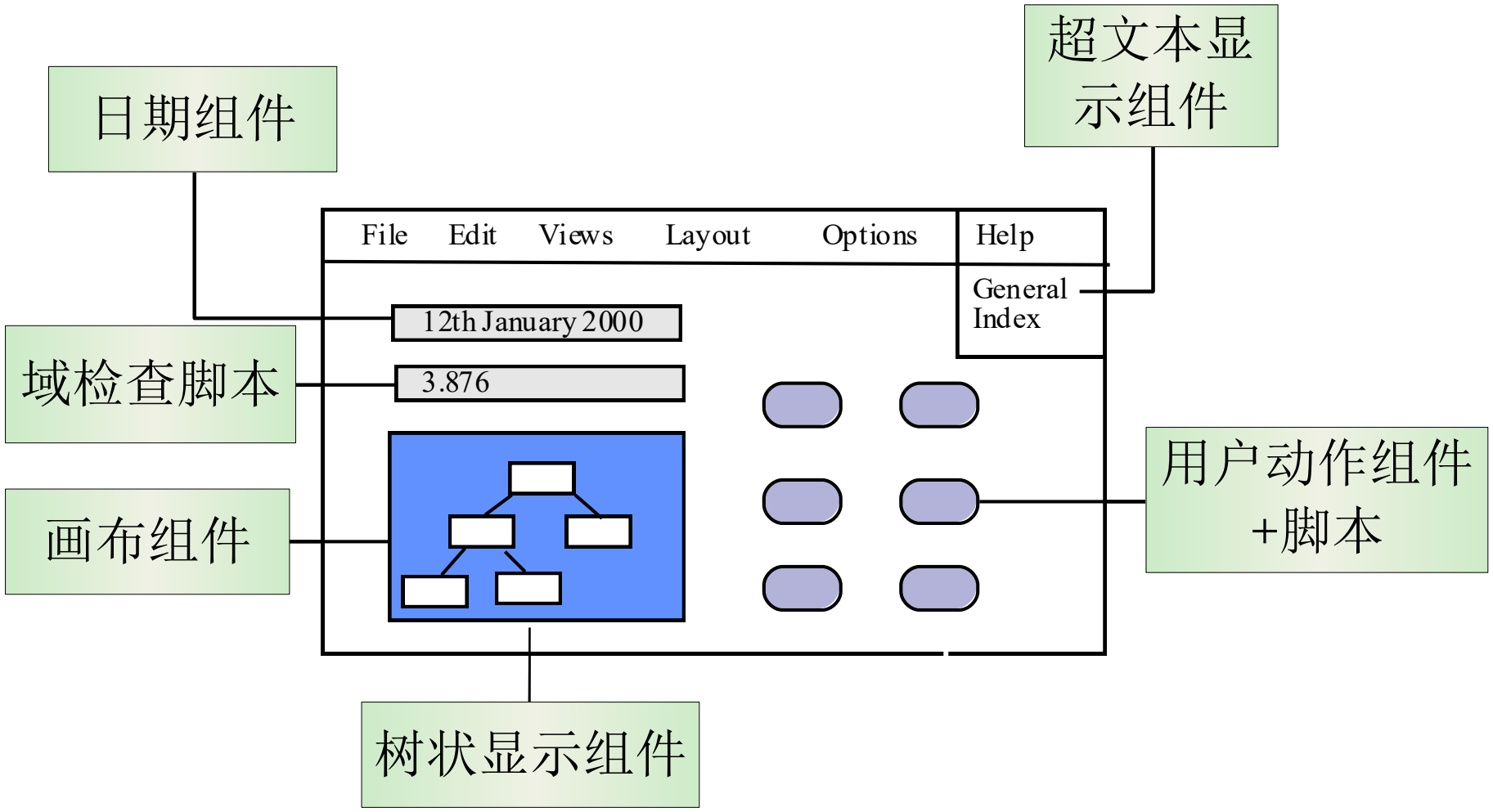
应用程序连接在复合文档中



可视化编程

- 脚本语言如Visual Basic支持可视化编程，通过标准的项目和相关组件生成原型的用户界面
- 一个大型的组件库用来支持这类开发
- 这些组件可能需要裁剪以适合特定的应用需求

基于复用的可视化编程


超文本显示组件
日期组件
File
Edit
Views
Layout
Options
Help
General
Index
12th January 2000
域检查脚本
3.876
画布组件
用户动作组件+脚本
树状显示组件

可视化开发的问题

- 难以协调团队的开发
- 没有一个清晰的系统体系结构
- 程序各部分之间复杂的依赖关系带来维护上的问题

用户界面原型开发

- 很难以一种有效的方式预先描述一个用户界面的外观和使用感觉，原型开发是基本方法。
- 用户界面开发的费用比例在增长
- 用户界面生成器用来画出界面、模拟界面功能
- Web界面应使用网页编辑工具开发原型

要点

- 系统原型能给最终用户关于系统功能的一个直观印象。
- 随着软件交付时间的要求越来越紧、原型开发越来越多
- 抛弃式原型开发是为了理解系统需求而进行的原型开发
- 进化式原型开发是对原型不断改进直到成为最终系统

要点

- 快速开发对原型开发非常重要。这可能要求先放弃部分系统功能，或者放松一些非功能性约束
- 原型开发技术包括使用高级语言、数据库编程以及利用可复用组件的原型构建技术
- 用户界面经常需要使用原型开发技术，因为用户界面不可能通过静态模型有效地预先描述，用户应该参与到原型的评估中来