How to Design Frameworks

Framework is

- the design of an application or subsystem
- expressed as
  - a set of abstract classes and
  - the way objects in those classes collaborate.

Use framework to build application by:

- Creating new subclasses
- Configuring objects together
- Modifying working examples
  (editing scripts)

Inversion of Control

Subroutine library

- User's program calls reused code.
- User designs structure of program.

Framework

- Reused code calls user's program
- Structure of program determined primarily by reused code.

Parts of Framework Application

New classes

Script that specifies classes of components by

- creating components
- connecting components
- parameterizing components

Testing Framework

Classes - Test, TestResult, TestSuite

Use by subclassing Test

Define instance methods to set up test, to perform tests

Define class methods to create a test suite

Model/View/Controller

- Classes - Model, View, Controller, ApplicationModel, ValueModel, etc.
- Use by using GUI builder to make a screen; the GUI builder automatically builds an ApplicationModel and a window-spec that later gets interpreted to build a window.

HotDraw

- Classes - Figure, Drawing, Handle, Tool, DrawingEditor
- Subclass DrawingEditor, Figure, rarely Drawing
- Parameterize Handle, Tool.
- There is a graphical tool for defining new Tools.

White-box vs. Black-box

White-box

Customize by subclassing

Emphasize inheritance

Must know internals

Black-box

Customize by configuring

Emphasize polymorphism

Must know interfaces

(continued)

White-box

Simpler, easier to design

Harder to learn, requires more programming.

Black-box

Complex, harder to design

Easier to learn, requires less programming.

First Rule of Framework Design

Don't. Buy one, instead.

Relevant Principles

- Frameworks are abstractions: people generalize from concrete examples
- Designing reusable code requires iteration
- Frameworks encode domain knowledge
- Customer of framework is application programmer

Generalize from Concrete Cases

People think concretely, not abstractly.

Abstractions are found bottom-up, by examining concrete examples.

To Generalize:

- find things with different names that are really the same,
- parameterize to eliminate differences,
- break large things into small things so that similar components can be found, and
- categorize things that are similar.

Finding Abstract Classes

Abstract classes are discovered by generalizing from concrete classes.

To give classes a common superclass:

- give operations common interface
- move operations with same implementation to superclass
- make operations with different implementation abstract

(continued)

- give them common interface
  - + rename operations so classes use same names
  - + reorder arguments, change types of arguments, etc.
  - + refactor (split or combine) operations

Frameworks Require Iteration

Reusable code requires many iterations.

Basic law of software engineering

If it hasn't been tested, it doesn't work.

Corollary: software that hasn't been reused is not reusable.

Frameworks Encode Domain Knowledge

Frameworks solve a particular set of problems.

Not always application-domain specific, but domain specific. (GUI, distribution, structured drawing editor, business transaction processing, workflow)

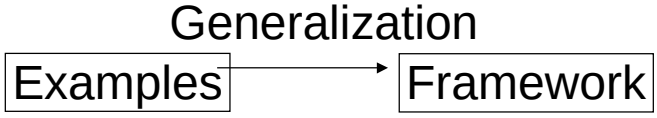
Customers are Programmers

Purpose of framework is to make it easier to build applications.

Apply these slogans to application programmers:

- The customer is always right.
- We are customer-driven.
- Understand your customer.

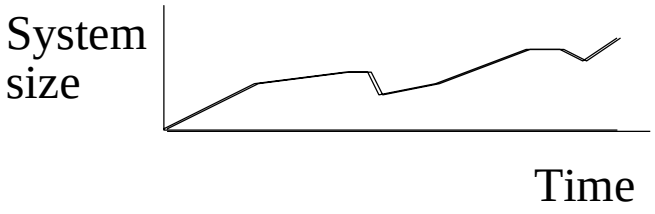
Example-driven Design



Generalization is iterative.

Most changes are small.

A few big changes represent new ways of looking at the problem.



To generalize faster:

- • get different points of view
- • explain/defend current design

Ideal Way to Develop Framework

Analysis → Design → Test

1) Analyze problem domain
   - Learn well-known abstractions.
   - Collect examples of programs to be built from framework. (Minimum of 4 or 5).

Ideal Way to Design Framework

2) Design abstraction that covers examples.

3) Test framework by using it to solve the examples.

- • Each example is a separate application.
- • Performing a test means writing software.

Designing Abstractions

Design phase: look for commonalities, represent each idea once.

Use design patterns

- implies that experience is needed

Insight and ingenuity is always useful, but hard to schedule.

Design Patterns

Design patterns make designs more black-box.

Show how to represent something that changes as an object

- • Strategy -- algorithm
- • Prototype -- products
- • State -- state of an object
- • Mediator -- way objects interact

Using design patterns

Each pattern makes design more complex.

Each pattern makes design more flexible.

Do you need that flexibility?

Is the complexity worth while?

Only use a pattern when it results in a simpler design than the alternatives.

Why Ideal is Never Followed

Analyzing domain requires analyzing individual examples, which is already very hard.

- Only practical if examples have already been analyzed.
- Analyzing and implementing examples is large fraction of the cost of the project.
- People need the concrete feedback of implementation.

Good Way to Develop Framework

Pick two similar applications.

Include developers experienced in the same domain.

One framework group

Two application groups

- framework group
  - Give and take software
  - Consider other applications
  - Explain and teach framework
- 2 application groups
  - Try to reuse as much software as possible
  - Complain about how hard software is to use

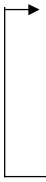
Typical Way to Develop Framework

Notice that many applications are similar.

Develop next application in that domain in an OO language.

Divide software into reusable and nonreusable parts.

Develop next application reusing as much software as possible.



- Surprise! Framework is not very reusable.
- Fix it.
- Develop next application reusing as much software as possible.

Problems with Reuse as a Side-effect

Conflicting goals

- get system out on time
- make it reusable

Hard to pay for reusability

Hard to enforce reusability

Advantages with Reuse as a Side-effect

- Framework developers experience being framework users.
- Only add features that are cost-effective.
- Helps prevent frameworks that are too complex and too abstract.

Another Strategy

- Design framework - prototype several small applications.
- Build real application.
- Refactor and extend framework and old applications.

Summary of Process

Start with examples of desired applications

Iteratively develop abstractions

Test by building applications

More detail

1) Three Examples
2) White-box Framework
3) Component Library
4) Hot Spots
5) Puggable Objects

(continued)

6) Fine-grained Objects
7) Black-box Framework
8) Visual Builder
9) Language Tools

http://st-www.cs.uiuc.edu/users/droberts/evolve.html

Application Generators

Black-box frameworks make it easier to:

- specify application with a picture
- generate code from a picture

Visual programming languages let non-programmers build applications.

Disadvantages of Black-box Framework

Black-box framework tends to have

- more kinds of objects
- more artificial kinds of objects
- more complex relationships between objects
- more objects

Not-quite-ultimate framework forces you to debug more complex system.

Patterns and Refactoring

Refactoring

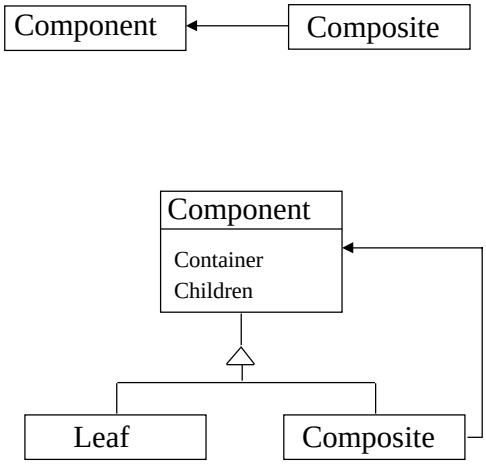
-- changing the structure of a program, but not its function.

-- the most common way to fix reusability problems.

-- making a "hot spot" flexible

-- often is applying a pattern

Refactoring to Help Find Composites



Hints for Framework Design

- Use object composition instead of inheritance
- Incrementally apply patterns / lazy generalization
- Framework should work out of the box

Strategic Concerns

Developing a framework is expensive, so look before you leap.

- Framework development requires long term commitment.
- Pick frameworks that will give you competitive advantage.

Start small and work up.

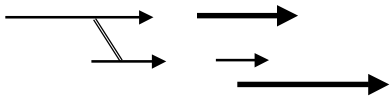
- get experience with OOP
- select and train good abstractors
- build small frameworks first
- generalize existing systems
- keep user base small at first

Customers are Crucial

- Get customers involved early, and use their feedback.
- Make your initial customers succeed.
- First set of customers are really part of the development team.

Reuse Scenarios

Ideal: Framework grows along with customer base.



Actual: Projects may customize the initial framework, and start competing streams of development.

Dealing with Iteration

Don’t claim framework is useful until your customers say it is.

Keep customer base small while framework is evolving.

A successful framework must evolve to meet new customer needs.

Don’t constantly tinker. Plan releases and coordinate with customers.

Documentation and Training

Documentation for framework costs several times usual

- how to use
- how to extend / how it works

Software must be understandable to be usable.

Improving documentation can make software more reusable.

Documentation and Training

- Base documentation on examples.
- Must debug documentation and training.
- Documenting system shows how to change it.
- Framework developers must be intimately involved.

NIH vs. TILI

- Problem with reuse is NOT fault of customer.
- Software is not as reusable as it is claimed.
- It is hard to make software reusable.

Reusable Design is Hard

- Must be abstract and powerful - theory of application domain.
- Must be customizable - theory of what users want to change.
- Must be easy to understand
  - -- simplicity is crucial
  - -- needs good documentation