Notes for Smalltalk Solutions

How to Develop Frameworks

Ralph E. Johnson
University of Illinois at Urbana-Champaign

Department of Computer Science
1304 W. Springfield Ave.
Urbana, IL 61801

johnson@cs.uiuc.edu
(217) 244-0093
http://st-www.cs.uiuc.edu/users/johnson/

Frameworks

Interface design and functional factoring constitute the key intellectual content of software and is far more difficult to create or re-create than code.

Peter Deutsch

A design of an application or subsystem represented by

- 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

Frameworks

Framework prescribes how to decompose a problem.

Not just the classes, but the way instances of the classes collaborate.

- shared invariants that objects must maintain, and how they maintain them
- framework imposes a collaborative model that you must adapt to.

Inversion of control -- "don't call us, we'll call you".

Examples

ApplicationModel>>postBuildWith:

- called after window is built so you can initialize variables
- customize by making subclass
- your code is called by system code

Menus

- you write method that creates menu and give name to model
- you write methods for messages on menu
- model calls method, gets menu, and selects an operation
- customize by making components
- your code is called by system code

Goal of Framework Design

- Minimize amount you have to learn
  - Build application from preexisting components.
  - Use small number of types of components over and over.
  - Write as little code as possible

    ultimate goal is zero code - build program by direct manipulation.

- Maximize power

It shouldn't take a good programmer to build a good program.

Generalizations Require Concrete Cases

People think concretely, not abstractly.

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

Generalization proceeds by

- finding things that are given different names but are really the same,
- parameterizing to eliminate differences,
- breaking large things into small things so that similar components can be found, and
- categorizing things that are similar.

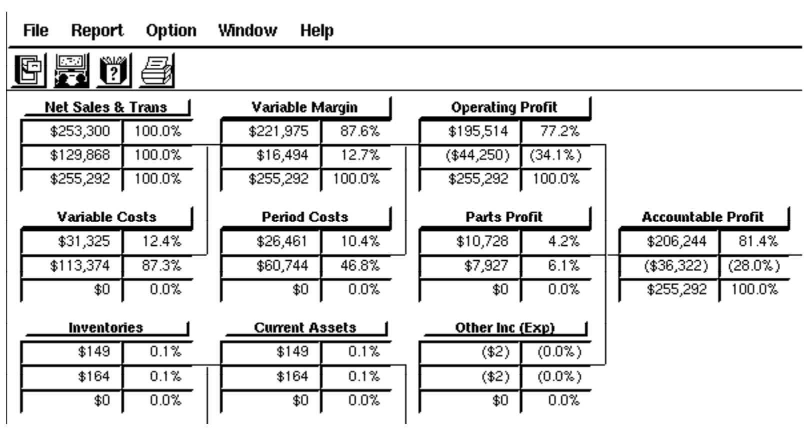
Generalization Occurs Piece by Piece

One insight leads to another.

One improvement shows another problem.

Gradual improvement is rule, but revolutions are memorable.

Dupont Model



Design History for Dupont Model

1) Make VisualWorks widget that is parameterized with aspects for actual, budget, and target.

   Aspects are in the DupontModel.

2) Separate domain model from application model.

   Dupont Model contains a set of domain models, one for each object.

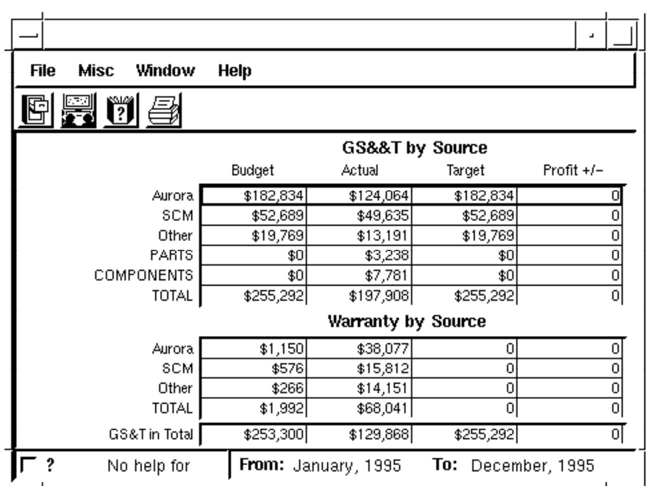
   Aspect methods forward to domain model.

3) Domain object should have actual, budget, and target aspects.

   VisualWorks widget is parameterized with name of domain model.

9

Domain Models



GS&&T by Source

	Budget	Actual	Target	Profit +/−
Aurora	$182,834	$124,064	$182,834	0
SCM	$52,689	$49,635	$52,689	0
Other	$19,769	$13,191	$19,769	0
PARTS	$0	$3,238	$0	0
COMPONENTS	$0	$7,781	$0	0
TOTAL	$255,292	$197,908	$255,292	0

Warranty by Source

	Budget	Actual	Target	Profit +/−
Aurora	$1,150	$38,077	0	0
SCM	$576	$15,812	0	0
Other	$266	$14,151	0	0
TOTAL	$1,992	$68,041	0	0
GS&T in Total	$253,300	$129,868	$255,292	0

10

Separate Domain Model from Application Model

Reduce duplication between DupontModel and summary reports.
Reduce duplication between summary reports.

netsalesBudget
netsalesTarget
netsalesActual
fixedCostBudget
fixedCostTarget
fixedCostActual

→

Netsales	FixedCost
actual	actual
budget	budget
target	target

Split classes and move code
Rename methods
Reorganize class hierarchy

Principles of Framework Design

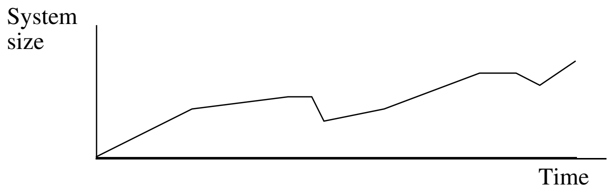
Code is not reusable until it has been reused.

Fixing reuse bugs means changing interfaces.

You can't generalize from one example.

Example-driven Design

Examples $\xrightarrow{\text{Generalization}}$ Framework



13

How to Generalize

To generalize faster:

- get different points of view
- explain/defend current design
- develop catalog of patterns

14

Patterns

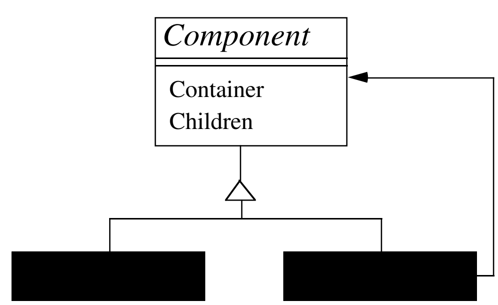
Design Patterns: Elements of Reusable Object-Oriented Software
Gamma, Helm, Johnson, and Vlissides, Addison-Wesley, 1995.

Data Model Patterns: Conventions of Thought
David C. Hay, Dorset House, 1996. Order from 1-800-342-6657

Pattern Languages of Program Design. Ed. Coplien & Schmidt,
Addison-Wesley, 1995. ISBN 0-201-60734-4

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

Composite Pattern



Composite and Component have the exact same interface.

- interface for enumerating children
- Component implements Children() by returning empty set
- usually interface for adding/removing children is private to Composite.

Composite in the Financial Model

Complex queries are made of simpler queries.

CompositeReport lets you build report that contains other reports.

Convert domain objects into TransactionSpace and CompositeValueModel

White-box vs. Black-box Frameworks

White-box framework:
- reuse by inheritance
- must understand details of how framework works

Black-box framework:
- reuse by composition
- only need to understand client interface

White-box framework is easier to design.
Black-box framework is easier to use.

White-box frameworks evolve to become more black-box.

A Reusable History of a Framework

Applications

Whitebox framework

Blackbox framework

Visual builder

History of Financial Modeling Framework

Applications for three business units,

- Application model for each window except Detail Reports, but many reused between business units
- Reusable components: QueryObjects, Report view

Whitebox framework

- ReportModel / ReportValues (application model / domain model)

Blackbox framework

- Componetize ReportModel
- Composite ReportValues

Builder to describe business unit

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).
   - Deciding whether an example is legitimate depends partly on your vision for the framework.

2) Design abstraction that can be specialized to cover examples.
   - Develop theory to explain these examples.

3) Test framework by using it to solve the examples.
   - Each example is a separate application.
   - Performing a test means writing software.

Why Ideal is Never Followed

Analyzing domain requires analyzing individual examples, and analyzing examples 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.

Typically some applications (the one you want to do next) are more important than others (the ones you have already done) so some examples are analyzed more than others.

Old applications work, so there is no financial incentive to convert them to use new software.

Good Way to Develop Framework

Pick two similar applications that need to be developed and that are obviously in your application domain.

Make sure your staff has some people who developed earlier applications in the same domain.

Divide project into three groups:

- framework group
  - Both gives and takes software
  - Considers how other applications would reuse framework
  - Develops documentation and training for framework
- 2 application groups
  - Try to reuse as much software as possible
  - Complains about how hard software is to use

Typical Way to Develop Framework

Realize that many applications are similar.

Develop next application in that domain in an OO language.

Try to divide software into reusable and nonreusable parts.

Develop second application reusing as much software as possible.

Surprise! Framework is not very reusable.

Fix it.

Develop third application reusing as much software as possible.

Surprise! Framework is not completely reusable.

Fix it.

...

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

Another Strategy

Design framework - prototype several small applications.

Build real application.

Refactor and extend framework and old applications.

Build real application

Refactor and extend framework and old applications.

...

HotDraw

Object-oriented graphics package for Smalltalk
written by Ward Cunningham and Kent Beck at Tektronix
http://st-www.cs.uiuc.edu/users/brant/Hotdraw.st

Framework for 2-D graphics

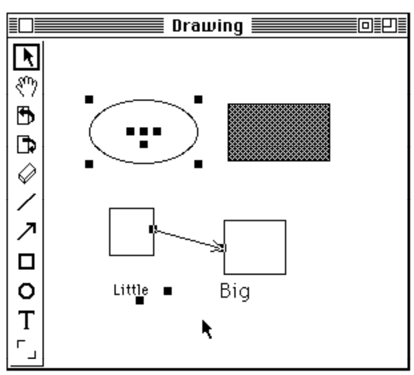
- Drawing tool
- Diagramming inspector
- Oscilliscope user interface prototyper
- Electronic schematic editor.
- PERT chart editor.

Components of framework

- Drawing -- a collection of figures
- Figure -- a visible component of a drawing
- Tool -- a user interface "mode"
- Handle -- a small box that controls an aspect of a figure

27

HotDraw example



28

Drawing

Drawing keeps track of figures
 add, add:behind:, remove:, bringToFront:, sendToBack:

Drawing can be displayed
 displayOn:

Drawing can be animated (default does nothing)
 step

Keeps track of "damaged region" for efficient redisplay.

Figure

A figure knows its location on the screen.
 origin, offset

A figure can display itself.
 displayOn:

A figure knows the drawings and other figures that depend on it and notifies them when it changes.

A figure can create handles for itself.
 handles

Tool

Tool has an icon (for pallette) and a cursor (for canvas).

Tool acts like a part of the controller. The drawing controller delegates all operations to the current tool, so changing the current tool changes the behavior of the editor.

Most Tools only define a few operations.

Operations performed by controller

activate
deactivate

backspace
controlCharacter:
type:

press
pressBackground
pressFigure:
release

Handle

A handle is a little black boxthat is attached to a figure and that does something to the figure when you press it.

One operation -- invoke: aView
-- perform special function
-- default is to follow cursor

Other Classes

DrawingEditor

Responsible for creating the pallette of tools.

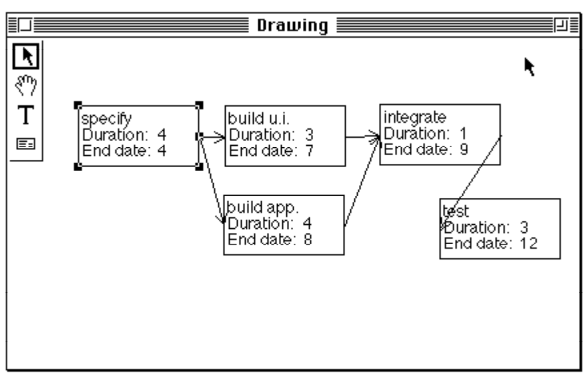
DrawingView

DrawingController

Responsible for knowing the current tool and directing events towards it.

In general, you have to subclass DrawingEditor to define the tool pallette and (sometimes) the class of a new Drawing, but you never have to subclass the other two.

PERT Chart



PERT Chart

New classes:

PERTEvent: figure with a duration and a completion time.

- subclass of ComposedFigure with TextFigures for labels and numbers
- can edit title, duration with text tool
- add handle to connect events
- connection causes constraint between completion time of one PERTEvent and starting time of the other
- internal constraints between starting time, duration, and completion time of a PERTEvent

PERTChartEditor: subclass of DrawingEditor that specifies the palette.

Problem with HotDraw

Tools are hard to build

- have to decide which methods to redefine
- usually duplicate code
  - code for following mouse
  - code in other tools

Change Tool to be compositional

- icons for pallette and cursor
- table mapping events to Commands

Commands

Command knows its figure and its tool

effect - evaluate Command

undo - undo effect of Command

MouseCommands
- initialize: aPoint
  - gets called at start of gesture
- moveTo: currentPoint
  - gets called repeatedly throughout gesture as mouse is moved

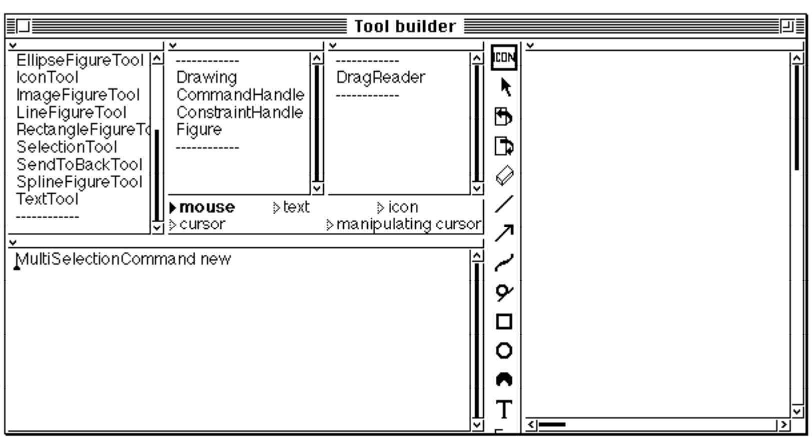
TextCommands
- initialize
- keyboardEvent: anEvent

Handle is now just a Figure with an embedded Command.

Tool doesn't have to be subclassed, just a table of Commands.

HotDraw can now support undo.

ToolBuilder



Black-box Frameworks

Replacing reuse by inheritance with reuse by object composition makes it easier to make programs without defining new classes.

Ultimate framework lets you plug objects together with visual language.

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.

Process of Framework Design

Framework is generalization of *implementation* of several problems.

- study solution domain as well as problem domain
- need to look at real programs
- need a wide range of carefully chosen examples
- generalization is the same process that occurs in all object-oriented design
  - find common abstractions
  - figure how to decompose problem into standard components
  - parameterize

Example-driven: the examples you choose determine your framework

Scheduling Development

Software is not reusable until it has been reused.

Reusing software that is not reusable will point out reusability defects.

1) Domain analysis, collect examples	(small group)
2) Make preliminary design	(small group)
3) Implement examples	(many groups)

In addition to the framework design group, there probably needs to be a group for each example, unless the examples are small.

4) Fix framework and examples	(many unhappy groups)

As usual, every order of magnitude increase in the number of users will illustrate some (reusability) defects that you haven't detected before.

Refactoring

Refactoring
- -- changing the structure of a program, but not its function.
- -- the most common way to fix reusability problems.
- -- refactoring can usually be viewed as "applying a pattern to a design"

Smalltalk Refactory - tool for refactoring programs on Visualworks

http://st-www.cs.uiuc.edu/users/brant/Refactory/RefactoringBrowser.html

Strategic Concerns

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

- Framework development requires long term strategic commitment.
- Pick frameworks that will give you competitive advantage.
- Start small and work up.
  - get experience with OOP
  - requires good abstractors, who must be selected and trained
  - build small frameworks first
  - generalize existing systems

Customers are Crucial

Get customers involved early, and incorporate their feedback.

Make your initial customers succeed.

Customers tell you what needs to be changeable and what is hard to understand.

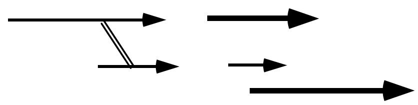
First set of customers are really part of the development team.

- their input is critical
- they will suffer from redesign of framework
- they must feel that their contribution to framework is important

Reuse Scenarios

Ideal: Successive versions of your framework meet the needs of a growing customer base.

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



45

Dealing with Iteration

Don't claim framework is useful until your customers say it is. This will take two or three versions.

Keep customer base small while framework is evolving.

A successful framework is a living framework, evolve it to meet new customer needs.

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

46

Documentation and Training

Software must be understandable to be usable.

Improving documentation can make software more reusable.

Base documentation on examples.

Budget time to debug documentation and training.

Framework is culture as well as software.

NIH vs. TILI

Problem with reuse is NOT fault of customer.

It is hard to make software reusable.

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