Building Adaptable Templates for Large Projects – Rapid Prototyping

By Phillip Kerman

Overview:

Large multimedia projects can mean a lot of work. Creating templates and utilizing rapid prototype techniques is essential to speed project completion. Rapid prototyping can solidify design concepts unlike conventional methods which may not involve the final media format. Templates can provide a consistent structure which is easy for authors to modify, but should not appear repetitive or boring to the user. Both techniques are well worth the investment as they result in reduced production costs.

Presentation Outline:

The ideal which templates and prototypes attempt to reach.

Define terms: "*Prototypes*" and "*Templates*"

> Prototypes are created quickly to solidify design concepts.
> Templates are designed to provide a consistent structure into which authors add content.

Prototypes

> Necessary to solidify agreement among team.
> Must be on computer because that's the final medium.
> *Successive Approximations* as applied to prototyping.
> Cost saving tips (see table on other side).
> Return on Investment argument for prototypes.

Compare: "*Models*" and "*Templates*"

> *Models* are the way the user experiences content.
> *Templates* are the way the author implements content.
>
> Models
>
> > Proven successful models are effective.
> > Models offer the user familiarity.
> > Design models for the user (not the author).
>
> Templates
>
> > Two considerations when designing templates:
> > —Who will implement the content?
> > —How dynamic will the template be?

Four types of templates (with examples of each):

Style guide for author	Engine for author
Runtime style guide	Runtime engine

Steps to designing a template (see table on right).

Summary

Questions from audience.

Steps to designing a template:

—Imagine the ultimate application, with each screen unique.
—Categorize the content. How is it the same and how is it different?
—Define models, trying to incorporate as many variations within each model.
—Create a few prototypes based on representative content.
—Analyze results and repeat as necessary.
—Develop a vocabulary for all team members.
—Build a table for content.
—Add dummy content to a representative template.
—Write "proofing scripts" (especially important for dynamic templates).
—Test at logical milestones to protect yourself from heading too far down the wrong path.

Cost saving tips for prototyping:

Learn to prototype with bad graphics (place holder graphics). *(This is difficult sometimes even for experienced people.)*

Be careful not to let prototype graphics influence final graphics or layout. *(When the final graphics begin to look like the prototype, be sure the artists are not following the prototype without reason.)*

Do prototypes at logical milestones in the production. *(Not just at the beginning, but at logical times when a prototype can help.)*

Don't prototype everything, but rather do a "full path review" (*That is, don't finish every module, section, and page. Rather, one module, one section and a few pages. Note: be sure you pick representative sections.)*

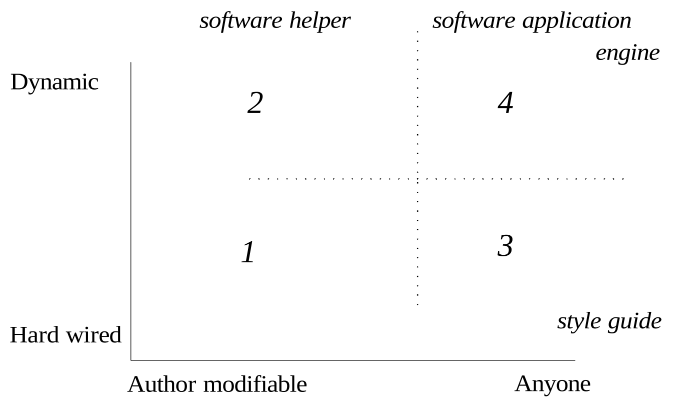
Don't fall in love with your code—learn to throw away the prototype. *(Prototypes lead to a good end, but not directly. When you think you know the end point, start over… your trip to the endpoint will be more direct and efficient.)*

Categorizing templates:

Templates can be categorized on two scales:
—how dynamic (or hard wired)?
—who will implement content (author or anyone)?

1—Style guide for author
2—Engine for author
3—Runtime style guide
4—Runtime engine

As you move higher or further to the right, effort and cost increase However, for large projects the unit cost may diminish—with economies of scale.



Glossary:

Author— The person who adds content.
Code Data Separation— Technique keeping code (or programming) separate from data (or media).
Dummy content— Meaningless text used before the actual content is finished, simulating final layout.
Dynamic— Opposite of static (or hard-wired). Referring to portions changing at *runtime*.
Engine— A template which is so dynamic that it can create multiple instances of itself, or automatically make copies.
Full path— Implementing every detail within one path the user can take.
Localization— Converting a finished project to another language or culture. The trick here is to make the conversion seamless.
Metaphor— Something representing a real-life object or place.
Model— A common or effective way of communicating an idea, e.g. a true-false question.
Place Holder— A temporary media file "holding the place" for the finished media while work continues. (See *dummy content*.)
Proofing scripts— A script (or subroutine) used only during production to "prove" that all possible outcomes function (without manually testing).
Runtime— Time when the user "watches" a project.
Spaghetti code— Programming which follows bad form. Often the result of a prototype that's gone through many changes.
Style guide— A definition of the layout for content. Can include typefaces, font sizes, etc.
Successive Approximations *(applied to prototypes)*— Process of approaching the ideal by designing, building, analyzing… then repeating.
Templates— A *style guide* which includes programming properties which can be varied.

Bibliography / Suggested Reading / Credits / Resources:

Interactivity by Design by Ray Kristof & Amy Satran Adobe Press, 1995 ISBN:1-56830-221-5
The Art of Human-Computer Interface Design Edited by Brenda Laurel, Addison-Wesley 1990 ISBN: 0-201-51797-3

Presentation graphics created by Diana Bauer at Artemus Productions < dbauer@teleport.com >
This document and related links available at http://www.teleport.com/ ~ phillip/toronto/