Services, Frameworks, and Components
What is really important to quality rapid application development?

H. James Hoover
Software Engineering Research Lab
Department of Computing Science
University of Alberta
hoover@cs.ualberta.ca
http://www.cs.ualberta.ca/~softeng
http://www.avrasoft.com

IBM CASCON 1998
Workshop on Component-based software composition
Thursday Dec 3

Research supported by NSERC and Teledyne Fluid Systems
Outline

• The story of SizeMaster
• Services, Components, and Frameworks
• A slogan
• RAD issues
• Conclusion
Pressure Safety Valve Engineering

• Teledyne Fluid Systems / Farris Engineering

• Sizing according to ASME and API standards

• Selection according to catalog

• Conformance to engineering practice. e.g. revision control, auditability, accountability

• Combine best of engineering and CS practice
Prevents this:
We needed to build one of these as SizeMaster:
This job is being used to demonstrate SizeMaster Mark 4 functionality. No relationship to any real job is implied.
Clearly RAD project from beginning:

- Weak grasp of new workflow possibilities.
- Vaguely articulated services.
- Radical change in UI
Components + Framework are key to RAD:
How? Delphi IDE for “two-tier” model

- Business objects are stored in a DBMS
- Forms move objects between states in lifecycle.
Delphi IDE for “two-tier” model

• IDE focuses on building forms

• Forms are containers for the components, e.g. UI and DBMS connections

• Business logic is spread over forms
But . . .

RAD does not imply good quality application
Instead often get:

- no encapsulation of business objects, schema overly exposed
- mixing of UI and business logic
- services unarticulated, diffuse in implementation
- no unity over applications in same domain
- ad hoc, brittle architecture, hard to evolve
Why?

Can look at RAD from two perspectives:

• User: business objects and workflow evolution

• Architecture: basic services identification and factoring

Most RAD is architecturally weak, even though components are good.
Slogan

Services + Frameworks + Components

⇒ Quality RAD
What is a component?

• physical, precise thing

• replaceable

• realizes a set of export interfaces, its capabilities

• conforms to a set of import interfaces

• exists in an architectural context or framework

• combine into assemblies, themselves components
What is a service?

• An exported interface that addresses some group of tasks.

• A collection of imported interfaces that establishes the context of the service.

• A service is not a component, it is an architectural element.
A service is not a component

- A component is one way to realize a service, by exploiting the component’s capabilities.

- but a service can be realized by a set of classes in the application framework, or even some piece of code.
What is a framework?

- software architecture + implementation + hooks
- provides generic capabilities in some domain
- custom application specific code added at *hooks*

Examples:
- Inprise Delphi spreadsheet
- DBMS - DB2, Access
- San Francisco
Frameworks

Promise:
Leverage existing design and implementation common to all applications in a domain.

Cost:
Must yield design authority and adapt your application to the overall architectural solution dictated by framework.
Service-centric Framework
Claim: Quality applications are built by

- identifying required services
- allocating services to components
- using frameworks to hook up and build components

This process is iterative and evolutionary.
A workshop is required to assemble components into a product.

The workshop provides:

• service decomposition
• an application framework
• a collection of components
• tools for customization
Problem with unadorned off-the-shelf workshops:

• Few developers are good architects,

• Do not understand the underlying services and frameworks,

• Workshop provides no services, framework, or process guidance
RAD Management Issue

Crucial to *choose the right workshop* for your RAD style.

Basic workshop IDE’s like Delphi, ACCESS, VB, Java or workshop technologies like CORBA, DCOM, are *only good for Waterfall RAD*

What about Evolutionary RAD?
• The success of evolutionary RAD depends crucially on *identifying and factoring services*.

• Services are provided by components.

• Can acquire small and large commodity components from vendors.

• Construct service components by customizing a service-providing framework.

• All services are coordinated by a domain-specific application framework.
Dangers

• Wrong level of abstraction for services

• Bad factoring of services

• Cross-component services, E.g. trouble handling
Conclusion

• functional decomposition isn’t dead yet, we just call them services

• commodity components can be bought

• some large scale ones need to be built

• good domain-specific application framework is key