

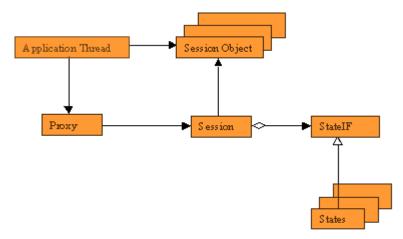
Using Osprey for

Understanding Object-Oriented Software

MITRE-Sponsored Research

Project Number: 51MSR102-A3

Principal Investigator:	Penny Chase
Task Lead:	David Harris
Project Staff:	Angel Asencio, Sam Cardman, Ellen Laderman, Suzi Lubar, Scott Mardis



Introduction

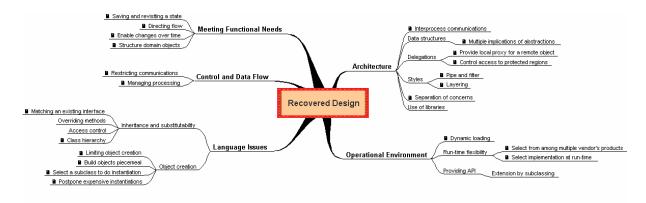
In the spring of 2000, we were asked to help with an assessment of object-oriented software. A MITRE sponsor needed to assess the quality of a C++ and Java software system. Specifically, the sponsor wanted to know if the software could be reused. This need was typical of the many needs that motivate analyses of software. Examples

include maintenance questions, reuse questions, code upgrades, component integrations, and analysis of hidden and explicit requirements.

We knew that commercial and open source resources for analyzing object-oriented software were limited. There were few tools available that fully parsed C_{++} or Java and these tended to be very expensive. Additionally, due to indirection caused by inheritance and polymorphism, it is difficult for these tools to correctly compute data flow and control flow.

Although the use of object-oriented languages is becoming pervasive, the indirection mentioned above and the lack of suitable tools make analysis an imprecise, error-prone art form. Thus, we recognized that the development of tools to understand object-oriented software would be an important area for research and development.

The obvious candidates for understanding object-oriented software structures are design patterns. Members of a design pattern community have developed best-practice descriptions for creating object-oriented software. These descriptions are agreed upon knowledge that bridges between software constructs, their rationale, and their consequences for code quality. If we know that developers used a particular pattern (intentionally or implicitly), we have a good idea of what forces (see those in the figure as examples) influenced the artifact's designers and how they chose to harness those forces in the design and implementation.



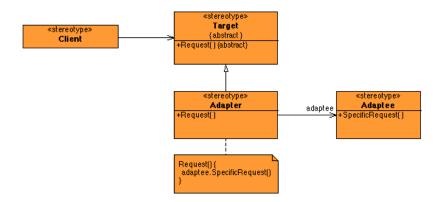
Forces influencing design

We developed a prototype software system named **Osprey**¹. To identify occurrences of design patterns in the software, we developed 104 recognizers – declarative descriptions of the constraints that must be met if one is to conclude that a pattern is present in the code. Then, given a program to analyze, Osprey matches the relationships among that program's structures and the pre-built design pattern templates.

For example, the *Adapter* design pattern is used to adapt a new service to meet the interface of a pre-existing set of services. One might have a set of services that works with x-y coordinates, but the program needs to take advantage of a service that produces results in distance-and-bearing coordinates. The way to achieve this is to use an *adapter*. A key component of the *Adapter* pattern is that there must be some mechanism for the

¹Osprey stands for Object-oriented Software Pattern REcoverY

adapter to know what is being adapted. Hence, Osprey identifies potential adapters that know about other structures and change the results that are generated by these other structures – in our example, changing distance-and-bearing results to x-y coordinates.



Adapter Pattern Template

With Osprey, analysts can look at programs in a more informed way. They are able to make statements such as "The purpose of class A is to adapt the results produced by class B in order to match the interface defined by class C." They can also make statements about code quality. We identified source code qualities (e.g. reliability, evolvability, and performance) effected by the use of design patterns and added this information to Osprey. For example, the presence of the pattern *AbstractFactory* supports maintainability.

Osprey generates HTML documentation based on all of the analysis it does on the source code. The generated documentation consists of:

- code-level reports (classes, methods),
- recovered design patterns,
- software quality and design rationale inferred from the use of patterns, and
- mappings from high-level expectations down to the source code that meets those expectations.

Osprey provides a way for analysts to access data through design concerns and design patterns. One page enumerates all the pattern participants for the pattern instances in the code (see the figure below for a partial view of the design pattern recognition results).

ile Edit	View Go	Communicato	R.C.								Help
5	5 3	14	2 🖻	-	-	a	121				
Back F	orward Relead		Search Netscape	Print	Security	Shop	Stop				
E Boo	ikmarks 🔏 L	ocation: [ftp:/	/transfer.mit	re.org/SI	1 °30-39/	39/24239	/Transfer/	ansvers/ACE	/Results-A	ACE-release	htal
in.											_
				Pattern I	nstances	for ACE					
Partitionin			Architecture	Structure	Creat						
iters.	Schehler.	Madiator.	Arniter,	Exception.	Pettery Mathe						
as Like Streets	<u>n.</u>	Elds.	CompanyationApprotec.		Eingle Deads						
engenite.		Entrat.	Communitation	Present Linkson		Dennell					
and the state of t	-	Standbart.	Perstanent	Briteher	Prototype,						
		ChanOrd, runnihil its	100000	Print.	Bull der						
		HattaleActions	Bracher	Tirtue Press	Alloutout	HTL.					
		Tister	AttesterConsector	Parate.	Chieropert.						
		Extent Argenotes	Autior@birtt	Bridge.							
		1910000	AttiveColeitFrancesm								
		Summer.									
		Displacion Robi		Entris & Angles							
	thod										
uctoryMet ecogFacto	aryMethodWi	Terington Reset Protei		Entrice & Barbor	tore detai	la (2940)					
uctoryMet ecogFacto		Terington Reset Protei		EntriceAdulter	iore detai			product			
actoryMet ecogFacto a	aryMethodWi	biseter		yMethod) m	nore detai		L.May Sectors				
actoryMet ecogFacto a act_sta	aryMethodWit	biseter	x pattern Factor	yMethod) m factory	iore detai		T.May Benete No.				
actoryMet ecogFacto a act_sta act_out	atoria alatrat_r _HoursC1 Listing_+C-+1	biseter	or pattern Factor AGE Artin Me Paint Many	yMethod) m factory	iore detai		T. Hay Leasts				
actoryMet ecogFacto a act_sta act_out	atoria alatrat_r _HoursC1 Listing_+C-+1	biseter	or pattern Factor AGE Artin Me Paint Many	yMethod) m factory	tore detai		T. Hay Leasts				
utoryMet ccogFacto 4 ACL.Na ACL.Sca acL.Sca acL.Sca	abstract_or abstract_or _NonreC1 t_brokeC1 tor	Relation Rest Made	ar pattern Factor AUL Artis 310 Juli 6.177 AUL Artis 310 Juli 6.177	Endine Adaptor yMethod) an factory or z=- E		an an an an an an	T. Hay Leasts				
utoryMet ecogFacto 4 ACL.Na ACL.Sa acL.Sa	abstract_or abstract_or _NonreC1 t_brokeC1 tor	Relation Rest Made	or pattern Factor AGE Artin Me Paint Many	Endine Adaptor yMethod) an factory or z=- E		an an an an an an	T. Hay Leasts Jo mit amendian T. DOT.AAD				
actoryMet a a act.m act.or apleCreat accoSimple	abstract_or abstract_or _NonreC1 t_brokeC1 tor	Relation Rest Made	ar pattern Factor Art Alter Jug Juick Art Roy Series - C Arter tern SkupleCreat	Endine Adaptor y Method) an factory or E Z E		an an an an an an	T. DUT AND				
a sctsu actsu actsu actsu accessingi ia	abstract_or abstract_or _NonreC1 t_brokeC1 tor	Television MCompetitors (fo water Returns (for pat factor	ar pattern Factor Art Alter Jug Juick Art Roy Series - C Arter tern SkupleCreat	Endine Adaptor y Method) an factory or E Z E	tals (s to	at at pr at pr at s	T. Stay Leonte Jie aut amerika I. DOT Allo	E			
actoryMet eccogFactor a ACL.Na aCL.NA ACL.NA	ar yM ethod Wil akitract_n , Manne_n 1. Jeanse tor le Creator With	Returbs Resolutions (Competitors of for realise Returns of four part fasts 	ar pattern Factor Art Alter Jug Juick Art Roy Series - C Arter tern SkupleCreat	Endine Adaptor y Method) an factory or E Z E	tals (1 %c	a a pr a a materia pr a a materia a sup r materia a sup r materia sup r sup r materia sup r materia sup r materia sup r sup sup r sup r sup r sup r sup sup r sup sup sup r sup r sup r sup sup sup sup sup sup sup sup sup sup	t Jug Joon Jo no amplia ant amplia bot amplia bot amplia bot amplia	E			
actoryMet accogFacto a act_m a	an y Method Wil abstract_ar 	Returbs Resolutions (Competitors of for realise Returns of four part fasts 	ar pattern Factor Art Alter Jug Juick Art Roy Series - C Arter tern SkupleCreat	Endine Adaptor y Method) an factory or E Z E	tals (1 %	a pr a d Hates Dans classrootes	Provide a second	E			

Design Pattern Web-page

Using Osprey Output

Analysts can start with program structure, with design pattern instances, with quality issues, with vulnerability concerns, or with design concepts. From any of these starting points, analysts can navigate to design patterns and to color-coded source code files. The table below lists the reports Osprey is able to generate, their category, a brief description of their contents, and the source language(s) for which they are generated.

Analysts who want an in-depth understanding of how the program is structured can start with design patterns. Analysts who wish to assess the software quality for source selection, reusability, or interoperability can start with quality reports or perhaps reports that show use of operating system services. Analysts who need to assess the potential vulnerabilities of the code can start with any of a collection of vulnerability reports.

Report Name	Category	Description	Source
			Language
Executive	Summary	Summary of source code statistics and	All
Summary		major observations	
Class Details	Source Code Information and	Class information including: methods,	C++, Java
	Recovered Design Patterns	inheritance, and pattern participation	
Class Hierarchy	Source Code Information	Class inheritance hierarchy	C++, Java
Class Instantiation	Source Code Information	Who creates a class and classes it creates	C++, Java
Methods	Source Code Information	Properties of the program's methods	C++, Java
Struct Details	Source Code Information	Lists of structs defined in the program	С
Procedures	Source Code Information	Properties of the program's procedures	С
Operating System Calls	Source Code Information	List of operating system calls	C, C++
Pattern-based	System Qualities	Links to program quality information	C++, Java
Qualities		based on pattern usage	
Pattern	System Qualities	Pattern impacts on program qualities and	C++, Java
Alternatives		alternative pattern suggestions	
Inheritance Faults	System Qualities	Potential variable initiation flaws due to polymorphism	C++, Java
Design Goals	Design Analysis	List of potential design goals and supporting pattern usage	C++, Java
Design Facts and Evidence	Design Analysis	Facts about the source code design	C++, Java
Design Index	Design Analysis	Index into pattern descriptions	C++, Java
Potential	Security Concerns	Programming constructs that could lead	C++, C
Vulnerabilities		to security violations	
Vulnerabilities by File	Security Concerns	File security evaluation	C++, C
Port Access	Security Concerns	List of ort numbers referenced in the source code	C++, C
Vulnerable OS Calls	Security Concerns	List of vulnerable OS calls used in program being analyzed	C++, C
Pattern Deviations	Security Concerns	Deviant implementations of design patterns	C++, Java
Pattern Instances	Recovered Design Patterns	List of pattern instances	C++, Java
Patterns within File Structure	Recovered Design Patterns	Pattern participation organized by file	C++, Java
Pattern Diagrams	Home	Basic pattern descriptions	C++, Java

Reports on the Source Code

Conclusions

Our research program makes progress in two areas of assisting analyst/users in software understanding tasks. We developed recognizers that identify instances of

software design pattern use in source code and we applied inference mechanisms to recovered information in order to provide insight into the design process itself. We have demonstrated the utility of design pattern recovery and the feasibility for automatic recovery of design rationale and software qualities of object-oriented software.

Additional Information

Osprey is available for use in analysis of software systems – either as a service we provide or as a tool that can be used at your site.

Readers who would like to know more about Osprey technology may wish to read *"Relating Expectations to Automatically Recovered Design Patterns"* by A. Asencio, S. Cardman, D. Harris, and E. Laderman, published in the Proceeding of the Working Conference on Reverse Engineering, Richmond, Virginia, 2002 and available from the authors.

We can be reached by email at osprey@mitre.org.