Preview of Award 1439957 - Annual Project Report

Cover |
Accomplishments |
Products |
Participants/Organizations |
Impacts |
Changes/Problems

Cover

Federal Agency and Organization Element to Which Report is Submitted:	4900
Federal Grant or Other Identifying Number Assigned by Agency:	1439957
Project Title:	SHF: Large: Collaborative Research: Science and Tools for Software Evolution
PD/PI Name:	Daniel Dig, Principal Investigator
Recipient Organization:	Oregon State University
Project/Grant Period:	11/18/2013 - 06/30/2016
Reporting Period:	07/01/2013 - 06/30/2014
Submitting Official (if other than PD\PI):	N/A
Submission Date:	N/A
Signature of Submitting Official (signature shall be submitted in accordance with agency specific instructions)	N/A

Accomplishments

* What are the major goals of the project?

We propose to create a change-oriented programming environment (COPE), with its associated science and tools for software evolution. Specifically, we have five major goals:

1. **Understand transformations**. We will study how programmers perceive, recall, and communicate change in complex code. We will study how programmers understand changes and why, when, and where programmers do (not) apply them.

2. **Automate transformations**. Writing and scripting transformations today is hard, e.g., implementing a seemingly simple transformation such as Rename Method refactoring requires writing hundreds of lines of code. We must make this simpler. COPE will provide better ways for average programmers to quickly script their own transformations and for experts to implement sophisticated transformations.

3. **Compose, manipulate, and visualize transformations**. Understanding properties of transformations— e.g., commutativity, dependencies, parameterization—is central to transformation's analysis and reuse. Programmers should be able to take a sequence of transformations representing an optimization or feature or bug fix, and reapply it to different versions of a program or even completely different programs. We will develop novel approaches to manipulate transformations as well as visualize their effects.

4. **Archive and retrieve transformations**. When programs are represented as sequences of transformations, COPE must subsume and generalize the activities of version control. Instead of dealing with text edits, COPE will archive, retrieve, visualize, and merge transformations.

5. **Infer transformations**. We will develop an infrastructure within COPE to infer higher-level changes (e.g., bug fix, security improvement) from lower-level changes, especially code edits. This will allow us to capture high-level program transformations even when the programmer did not specify them.

This is a joint project between the PI Professor Dig at the Oregon State University, the Department of Computer Science at the University of Illinois at Urbana-Champaign with Professors Johnson, Marinov, Bailey, and the Department of Computer Science at the University of Texas, Austin, with Prof Don Batory . We plan to have regular monthly teleconference meetings between the three institutions and one annual in-person meeting between all PIs and supported students. We have several joint publications between the three institutions.

*** What was accomplished under these goals (you must provide information for at least one of the 4 categories below)?**

Major Activities:

In this second year, despite the lead PI’s move from Illinois to Oregon State University, we made important progress on all of the foundational components of COPE. We were able to:

- (i) develop an infrastructure to collect code evolution events such as text edits and test runs in the Eclipse and IntelliJ IDEA IDEs

- (ii) understand how version control systems influence the way how programmers manage software changes (see our ICSE’14a paper)

- (iii) infer higher-level, unknown frequent transformations from text edits (see our ICSE’14b paper)

- (iv) automate new refactorings to help Java developers port their code to the new lambda expressions functional features provided in the Java language (see our FSE’13a paper)

- (v) automate new refactorings and testing tools for helping programmers improve the responsiveness of mobile applications (see our ICSE’14c paper)

- (vi) develop a novel way to automatically test refactoring objects (see our ECOOP'13 paper)

- (vii) develop an infrastructure to browse code changes and offer code completion using temporal code evolution data (see our ICSE’13 NIER paper)

- (viii) develop an infrastructure for testing configurable systems that can change for different configurations (see our FSE’13b paper)

- (ix) develop a novel way to compose higher-level refactorings that introduce design patterns from lower-level refactorings (see our OOPSLA’14 submission)

Specific Objectives:

(i) In order to accomplish our vision where fine-grained code changes become first-class citizens, we need access to fine-grained changes produced by real-world developers. Thus, we developed an infrastructure to leverage the Eclipse and Intellij IDEA IDEs to capture many programming events (e.g., text edits, test runs) as programmers perform them online. Our infrastructure is ready now for deployment and we have two scenarios: one that is targeting open-source developers and the data will be sent to our servers for analysis, and another that is targeting proprietary developers (and the data will be sent to the company’s servers, and we will run analytics under supervision from the company).

(ii) In order to understand how **version control systems (VCS)** influence the way how programmers manage software changes, we conducted an in-depth, large scale empirical study that looks at the influence of distributed VCS (DVCS) on the practice of splitting, grouping, and committing changes. We recruited 820 participants for a survey that sheds light into the practice of using DVCS. We also analyzed 409M lines of code changed by 358300 commits, made by 5890 developers, in 132 repositories containing a total of 73M LOC. (see our ICSE’4a paper)

- (iii) In our ICSE’14b paper we present the first approach that identifies previously unknown frequent code change patterns from a fine-grained sequence of code changes. Our novel algorithm effectively handles challenges that distinguish continuous code change pattern mining from the existing data mining techniques. We evaluated our algorithm on 1,520 hours of code development collected from 23 developers, and showed that it is effective, useful, and scales to large amounts of data.

- (iv) We automated new refactorings to help Java developers port their code to the new lambda expressions functional features provided in the Java language (see our FSE’13a paper)

Java 8 introduces two functional features: lambda expressions and functional operations like map or filter that apply a lambda expression over the elements of a Collection. Refactoring existing code to use these new features enables explicit but unobtrusive parallelism and makes the code more succinct. However, refactoring is tedious: it requires changing many lines of code. It is also error-prone: the programmer must reason about the control-, data-flow, and side-effects. Fortunately, refactorings can be automated.

We designed and implemented **LambdaFicator**, a tool which automates two refactorings. The first refactoring converts anonymous inner classes to lambda expressions. The second refactoring converts for loops that iterate over Collections to functional operations that use lambda expressions.

- (v) We automated new refactorings and testing tools for helping programmers improve the responsiveness of mobile applications (see our ICSE’14c paper).

Asynchronous programming is in demand today, because responsiveness is increasingly important on all modern devices. Yet, we know little about how developers use asynchronous programming in practice. Without such knowledge, developers, researchers, language and library designers, and tool providers can make wrong assumptions.

We present the first study that analyzes the usage of asynchronous programming in a large experiment. We analyzed 1378 open source Windows Phone (WP) apps, comprising 12M SLOC, produced by 3376 developers. Using this data, we answer 2 research questions about use and misuse of asychronous constructs. Inspired by these findings, we developed (i) **Asyncifier**, an automated refactoring tool that converts callback-based asynchronous code to use async/await; (ii) Corrector, a tool that finds and corrects common misuses of async/await.

- (vi) Testing refactoring engines is a challenging problem that has gained recent attention in research. Several techniques were proposed to automate generation of programs used as test inputs for refactoring engines and to help developers in inspecting test failures. However, these techniques can require substantial effort for writing test generators or finding unique bugs, and many of the bugs found resulted from “corner cases” that developers dismissed as being unlikely to occur in practice. Additionally, these techniques do not provide an estimate of how reliable refactoring engines are for refactoring tasks on real software projects.

In our ECOOP’13 paper we evaluate an end-to-end approach for testing refactoring engines and estimating their reliability by (1)~systematically applying refactorings at a large number of places in well-known, open-source projects and collecting failures during refactoring or while trying to compile the refactored projects, (2)~clustering failures into a small, manageable number of failure groups, and (3)~inspecting failures to identify non-duplicate bugs.

- (vii) develop an infrastructure to browse code changes and offer code completion using temporal code evolution data (see our ICSE’13 NIER paper) Modern IDEs make many software engineering tasks easier by automating functionality such as code completion and navigation. However, this

functionality operates on one version of the code at a time. We envision a new approach that makes code completion and navigation aware of code evolution and enables them to operate on multiple versions at a time, without having to manually switch across these versions. We illustrate our approach by discussing several example scenarios where it makes the developer's work easier. We also describe a prototype Eclipse plugin that embodies our approach for a part of functionality, namely code completion and navigation for Java code.

- (viii) Many programs can be configured through dynamic and/or static selection of configuration variables. A **software product line (SPL)**, for example, specifies a family of programs where each program is defined by a unique combination of features. Systematically testing SPL programs is expensive as it can require running each test against a combinatorial number of configurations. Fortunately, a test is often independent of many configuration variables and need not be run against every combination. Configurations that are not required for a test can be pruned from execution.

In our FSE’13b paper presents **SPLat**, a new way to dynamically prune irrelevant configurations: the configurations to run for a test can be determined during test execution by monitoring accesses to configuration variables. SPLat achieves an optimal reduction in the number of configurations and is lightweight compared to prior work that used static analysis and heavyweight dynamic execution.

- (ix) Introducing design patterns into a program by hand is tedious and error-prone. Refactorings help but manual tasks still remain: you must understand available refactorings, determine a precise sequence of refactorings to invoke, and perform these tasks repetitively to a laborious degree.

In our paper [OOPSLA’14sub] we present *Reflective Refactoring (*RR*)*, a Java package to automate the introduction of classical design patterns (Visitor, Abstract Factory, *etc*), their inverses and variants. We encoded 78% of classical design patterns as R2 scripts; for the remaining 22% it is unclear of their role in a refactoring tool. (The State design pattern, for example, is commonly implemented in Model Driven Engineering tools, where a statechart is translated to code.

Although this pattern is definitely automatable, it is unclear whether it should belong in a refactoring tool.) RR will be the basis for future work on refactoring the codebase of SPLs.

Significant Results:

- Using the data from our empirical study [ICSE’14a], we uncovered some interesting facts. For example, (i) commits made in distributed repositories were 32% smaller than the centralized ones, (ii) developers split commits more often in DVCS, and (iii) DVCS commits are more likely to have references to issue tracking labels. This clearly shows that the tooling infrastructure that developers use has an effect on how developers manage changes.

- In our ICSE’14b paper, we analyzed some of the mined code change patterns and discovered ten popular kinds of high-level program transformations. More than half of our 420 survey participants acknowledged that eight out of ten transformations are relevant to their programming activities.

- We evaluated our LambdaFicator refactoring tool [FSE’13a] by using 9 open-source projects where we have applied the two refactorings 1263 and 1709 times, respectively. The results show that LambdaFicator is useful: (i) it is widely applicable, (ii) it reduces the code bloat, (iii) it increases programmer productivity, and (iv) it is accurate.

- Our empirical evaluation of the Asyncifier and Corrector tools for introducing asynchronous patterns [ICSE’14c] shows that these tools are (i) applicable and (ii) efficient. Open-source developers accepted 314 patches generated by our tools.

- By using our approach [ECOOP’13] on the Eclipse refactoring engines for Java and C, we already found and reported 77 new bugs for Java and 43 for C; the Eclipse developers already fixed at least 7 of these bugs and confirmed 62 more as real bugs. Despite the seemingly large numbers of bugs, we found these refactoring engines to be relatively reliable, with only 1.4% of refactoring tasks failing for Java and 7.5% for C. In summary, the results show that our approach can be easily adopted for multiple programming languages, that applying refactorings to real software projects can discover new, important bugs, and that refactoring engines are more reliable than one would expect from some previous studies.

- Experimental results [FSE’13b] on 10 SPLs written in Java show that SPLat substantially reduces the total test execution time in many cases. Moreover, we demonstrate the scalability of SPLat by applying it to a large industrial code base written in Ruby on Rails.

- We believe our approach for code-completion [ICSE’13NIER] opens a new line of research that adds a novel, temporal dimension for treating code in IDEs in the context of tasks that previously required manual switching across different code versions.

- Our approach for scripting design patterns via our Reflective Refactoring (R2) is effective and efficient. In one application, R2 automatically created a visitor with 276 visit methods by invoking 554 Eclipse refactorings in a few minutes – an achievement that could not be done manually. We demonstrate the generality and scalability of R2, illustrate its productivity potential, and explain why refactoring speed and correctness are critical issues for scripting in next-generation refactoring engines.

Key outcomes or Other achievements:

- 7 papers accepted at the top conferences in Software Engineering, and 1 under review

- 1 award paper at ICSE'14, and another one nominee at ICSE'14

- discovered and reported several hundred bugs in open-source projects

- one of our tools, LambdaFicator, ships with the official release of the NetBeans IDE, which is used by millions of Java developers.

- *A Study and Toolkit for Asynchronous Programming in C#*. Semih Okur, David Hartveld, Danny Dig, and Arie van Deursen. Proceedings of ICSE'14. Acceptance ratio: 20% (99/499). **ACM SIGSOFT Distinguished Paper Award**

- *Mining Fine-Grained Code Changes to Detect Unknown Change Patterns*. Stas Negara, Mihai Codoban, Danny Dig, and Ralph Johnson. Proceedings of ICSE'14 . Acceptance ratio: 20% (99/499). **ACM SIGSOFT Distinguished Paper Award Nominee**

- *How Do Centralized and Distributed Version Control Systems Impact Software Changes?* Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. Proceedings of ICSE'14. Acceptance ratio: 20% (99/499).

- *Crossing the Gap from Imperative to Functional Programming through Refactoring.* Alex Gyori, Lyle Franklin, Danny Dig, and Jan Lahoda. Proceedings of FSE'13. Acceptance ratio: 20% (51/251).

- *Temporal code completion and navigation .* Yun Young Lee, Sam Harwell, Sarfraz Khurshid, Darko Marinov. Proceedings of ICSE'13 NIER

- *SPLat: lightweight dynamic analysis for reducing combinatorics in testing configurable systems.* Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don S. Batory, Sabrina Souto, Paulo Barros, Marcelo d'Amorim. Proceedings of FSE'13, Acceptance ratio: 20% (51/251).

- *Systematic Testing of Refactoring Engines on Real Software Projects.* Milos Gligoric, Farnaz Behrang, Yilong Li, Jeffrey Overbey, Munawar Hafiz, Darko Marinov. Proceedings of ECOOP'13. Acceptance Ratio: 25% (29/116).

- *Scripting Refactorings in Java to Introduce Design Patterns*. Jongwook Kim, Don Batory, Danny Dig. Under Review at OOPSLA'14.

* What opportunities for training and professional development has the project provided?

Carrying out the research activities provided professional training to four PhD students at OSU, 3 PhD students at Illinois, and 1 PhD student at Texas. We expect that once we have a more complete version of COPE running, we will use it for the undergrad Software Engineering classes at OSU, Illinois, and Texas.

* How have the results been disseminated to communities of interest?

We have published the results in top conferences in Software Engineering. We have also released several tools as free, open-source, and have advertised them at professional venues, such as conferences, visits to other universities (e.g., CMU, North Carolina State U, etc).

* What do you plan to do during the next reporting period to accomplish the goals?

We will also conduct interviews with programmers at software companies to find out how programmers perceive, recall, and communicate changes

We will start working on deploying our transformation-centric approach on existing version control systems like Git and

SVN. Rather than imple-menting a brand new version control system (with the danger of never being adopted by practitioners) we will enhance the capabilities of existing version control systems, providing orthogonal views of changes at different levels of abstraction.

We will consolidate the inference area, with the aim of detecting other classes of transformations, the first target are inferring development cycles, bug fixes, performance improvements. For this, we will use a hybrid of static/dynamic analysis, as well as machine learning techniques.

We will improve upon Reflective Refactoring to make it easier for other programmers to script their own custom transformations.

Products

Books

Book Chapters

Conference Papers and Presentations

Semih Okur and David Hartveld and Danny Dig and Arie van Deursen (2014). *A Study and Toolkit for Asynchronous Programming in C#*. ICSE. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Alex Gyori and Lyle Franklin and Danny Dig and Jan Lahoda (2013). *Crossing the gap from imperative to functional programming through refactoring*. ESEC/SIGSOFT FSE. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Caius Brindescu and Mihai Codoban and Sergii Shmarkatiuk and Danny Dig (2014). *How Do Centralized and Distributed Version Control Systems Impact Software Changes?*. ICSE. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Lyle Franklin and Alex Gyori and Jan Lahoda and Danny Dig (2013). *LAMBDAFICATOR: from imperative to functional programming through automated refactoring*. ICSE Demo. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Stas Negara and Mihai Codoban and Danny Dig and Ralph Johnson (2014). *Mining Fine-Grained Code Changes to Detect Unknown Change Patterns*. ICSE. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Chang Hwan Peter Kim and Darko Marinov and Sarfraz Khurshid and Don S. Batory and Sabrina Souto and Paulo Barros and Marcelo d'Amorim (2013). *SPLat: lightweight dynamic analysis for reducing combinatorics in testing configurable systems*. ESEC/SIGSOFT FSE. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Milos Gligoric and Farnaz Behrang and Yilong Li and Jeffrey Overbey and Munawar Hafiz and Darko Marinov (2013). *Systematic Testing of Refactoring Engines on Real Software Projects*. ECOOP. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Yun Young Lee and Sam Harwell and Sarfraz Khurshid and Darko Marinov (2013). *Temporal code completion and navigation*. ICSE. . Status = PUBLISHED; Acknowledgement of Federal Support = Yes

Inventions

Journals

Licenses

Other Products

Other Publications

Patents

Technologies or Techniques

Thesis/Dissertations

Stas Negara. *Towards a Change-oriented Programming Environment*. (2013). University of Illinois at Urbana-Champaign. Acknowledgement of Federal Support = Yes

Websites

Lambdaficator refactoring tool
http://refactoring.info/tools/LambdaFicator/

homepage of the Lambdaficator refactoring tool

asynchronous programming toolkit and data
http://www.learnasync.net/

homepage of the asynchronous programming toolkit and data

repository of VCS study data
http://cope.eecs.oregonstate.edu/VCStudy/

repository of VCS study data

Participants/Organizations

What individuals have worked on the project?

Name	Most Senior Project Role	Nearest Person Month Worked
Dig, Daniel	PD/PI	2
Bailey, Brian	Co-Investigator	1
Batory, Don	Co-Investigator	2
Johnson, Ralph	Co-Investigator	1
Marinov, Darko	Co-Investigator	1
Brindescu, Caius	Graduate Student (research assistant)	12
Codoban, Mihai	Graduate Student (research assistant)	12
Gyori, Alex	Graduate Student (research assistant)	5
Hilton, Michael	Graduate Student (research assistant)	12
Kim, Jongwook	Graduate Student (research assistant)	12
Negara, Stas	Graduate Student (research assistant)	3

Okur, Semih	Graduate Student (research assistant)	6
Shmarkatiuk, Sergii	Graduate Student (research assistant)	12
Simons, Connor	Graduate Student (research assistant)	12

Full details of individuals who have worked on the project:

Daniel Dig
Email: digd@eecs.oregonstate.edu
Most Senior Project Role: PD/PI
Nearest Person Month Worked: 2

Contribution to the Project: Supervising the overal activities

Funding Support: This NSF grant

International Collaboration: Yes, Netherlands
International Travel: No

Brian Bailey
Email: bailey@illinois.edu
Most Senior Project Role: Co-Investigator
Nearest Person Month Worked: 1

Contribution to the Project: HCI expertise

Funding Support: industry gifts

International Collaboration: No
International Travel: No

Don Batory
Email: dsb@cs.utexas.edu
Most Senior Project Role: Co-Investigator
Nearest Person Month Worked: 2

Contribution to the Project: leader at Texas

Funding Support: industry gifts

International Collaboration: No
International Travel: No

Ralph Johnson
Email: rjohnson@illinois.edu
Most Senior Project Role: Co-Investigator
Nearest Person Month Worked: 1

Contribution to the Project: expertise on frameworks

Funding Support: industry gifts

International Collaboration: No
International Travel: No

Darko Marinov
Email: marinov@illinois.edu
Most Senior Project Role: Co-Investigator
Nearest Person Month Worked: 1

Contribution to the Project: leader at Illinois

Funding Support: industry gifts

International Collaboration: No
International Travel: No

Caius Brindescu
Email: brindesc@eecs.oregonstate.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 12

Contribution to the Project: research on VCS, infrastructure for recording changes

Funding Support: none

International Collaboration: No
International Travel: No

Mihai Codoban
Email: codobanm@eecs.oregonstate.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 12

Contribution to the Project: research on mining frequent code changes

Funding Support: none

International Collaboration: No
International Travel: No

Alex Gyori
Email: gyori@illinois.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 5

Contribution to the Project: research on testing

Funding Support: industry gifts

International Collaboration: No
International Travel: No

Michael Hilton
Email: hiltonm@eecs.oregonstate.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 12

Contribution to the Project: lead student architect at OSU

Funding Support: none

International Collaboration: No
International Travel: No

Jongwook Kim
Email: jongwook@cs.utexas.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 12

Contribution to the Project: reflective refactoring

Funding Support: none

International Collaboration: No
International Travel: No

Stas Negara
Email: snegara2@illinois.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 3

Contribution to the Project: algorithm for mining frequent changes

Funding Support: industry gifts

International Collaboration: No
International Travel: No

Semih Okur
Email: okur2@illinois.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 6

Contribution to the Project: refactoring for improving responsiveness

Funding Support: industry gifts

International Collaboration: No
International Travel: No

Sergii Shmarkatiuk
Email: shmarkas@eecs.oregonstate.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 12

Contribution to the Project: research on VCS, infrastructure on storing research data

Funding Support: none

International Collaboration: No
International Travel: No

Connor Simons
Email: simmons6@illinois.edu
Most Senior Project Role: Graduate Student (research assistant)
Nearest Person Month Worked: 12

Contribution to the Project: HCI

Funding Support: none

International Collaboration: No
International Travel: No

What other organizations have been involved as partners?

Name	Type of Partner Organization	Location
Oracle	Industrial or Commercial Firms	Czeck Republic

Full details of organizations that have been involved as partners:

Oracle

Organization Type: Industrial or Commercial Firms
Organization Location: Czeck Republic

Partner's Contribution to the Project:
Collaborative Research

More Detail on Partner and Contribution: Jan Lahoda from Oracle office in Prague helped us to integrate and release LamdaFicator in the NetBeans IDE which is produced by Oracle

---

Have other collaborators or contacts been involved? Yes

Impacts

What is the impact on the development of the principal discipline(s) of the project?

Our LambdaFicator tool ships with the official release of the NetBeans IDE which is used by millions of Java developers everyday.

Our Asyncifier and Corrector tools found hundreds of new bugs in real-world, mature open-source applications. The developers accepted 314 patches generated by these tools.

The approach for testing refactorings has found 120 new bugs in production quality refactoring tools from the Eclipse

IDE. The developers of Eclipse have started to fix some of these bugs.

The inference techniques are the basis of elevating software changes at higher levels of abstraction.

Besides the impact on research and development of software engineering, we expect to use the techniques and results in the undergrad education at Oregon State University, University of Illinois, and University of Texas.

What is the impact on other disciplines?

Nothing to report.

What is the impact on the development of human resources?

The project is providing a rich education environment for future scientists. One of the PhD students involved (Stas Negara) graduated in Aug 2013 and joined Google.

What is the impact on physical resources that form infrastructure?

Nothing to report.

What is the impact on institutional resources that form infrastructure?

Nothing to report.

What is the impact on information resources that form infrastructure?

Nothing to report.

What is the impact on technology transfer?

LambdaFicator is now incorporate in the official release of the NetBeans IDE for Java developers. All tools in our toolset are released freely for public use.

What is the impact on society beyond science and technology?

Just as machinery fostered the industrial revolution, machinery automating software evolution tasks will foster a revolution in software technology. By embracing the view that change (transformations) lies at the heart of software development, US software education and industry will take a critical step toward maintaining its supremacy long-term.

Changes/Problems

Changes in approach and reason for change

Nothing to report.

Actual or Anticipated problems or delays and actions or plans to resolve them

Nothing to report.

Changes that have a significant impact on expenditures

Nothing to report.

Significant changes in use or care of human subjects

Nothing to report.

Significant changes in use or care of vertebrate animals

Nothing to report.

Significant changes in use or care of biohazards

Nothing to report.