Architecturally Significant Requirements

- Play a strategic role in driving architectural design
- Often critical to the success (or failure of a system).
- Often represent quality concerns such as performance, portability, reliability etc.
- **Non-functional Requirements** (NFRs) are often overlooked in the requirements specification process.

Example: A medical device used to perform laser surgery must be highly responsive.
Talk Outline

- Architecturally Significant Requirements and their impact on architectural design.
  - Focus on agile projects
  - Examples from TraceLab project
- Establishing and utilizing trace links between quality concerns and code
  - Patterns of traceability
  - Archie tool
- Recovering architectural knowledge
  - Machine learning techniques

Working with ASRs

- In practice ASRs (especially NFRs) are often not elicited and are not clearly specified.
  - Many Software Requirements Specifications simply don’t include NFRs.
  - Similarly, many agile projects fail to include ASR-related user stories.
- Is there a better way?
- In our TraceLab project we adopted a persona-driven approach which enabled us to discover architecturally significant requirements early in the project and to use our knowledge to make informed decisions about architectural design and implementation.
ASRs in TraceLab

- TraceLab is a US $2 Million Project funded by the National Science Foundation.
- Developed by collaborators at DePaul University, College of William and Mary, Kent State Univ., and Univ. of Kentucky.
- Intended to empower future traceability research through facilitating innovation and creativity, increasing collaboration between traceability researchers, decreasing the startup costs and effort of new traceability research projects, and fostering technology transfer.
- Provides an environment in which researchers can design and execute experiments, share components and datasets, and comparatively evaluate results in a controlled setting.
**Competing Tradeoffs**

We want to write components in C#.

We're programming this thing and we say that we should just program for Windows and everyone will have to use Windows.

It's going to work on Mac, right?

I don't want to do any programming.

I want to keep my R and MatLab scripts.

I'm not learning another language.

We should get this into the hands of our users early so we can get early feedback.

It better be as fast as running experiments that I write myself.

I just want it to install and run easily.

I have to run it on my desktop as I have proprietary data.

I'm willing to share with others, but not until after I've published.

Can you offer it as a service?

We only need to cut across the chaos.

We only have 3 years to deliver everything!!

**Traditional HCI Personas**

We decided to represent the conflicting needs through developing a set of architecturally-savvy personas.

Traditionally persona construction involves surveying users, classifying them, formulating hypotheses of use, validating, creating scenarios, and finally designing personas.

Too time consuming for our project i.e. too much upfront effort that would retard the achievement of our goals.

Solution: **Persona sketches.**
Meet Karly...

**Karly** is a new PhD student. She is interested in tracing requirements to software architecture. She has contacts with a local company who will allow her to access their data for her experiments; however this data is proprietary (i.e. protected by a NDA) and so she cannot share it with anyone else. She predicts that it will take her about 6 months to set up her traceability environment, but then she discovers TRACY. Karly is quite a good programmer, but is much more interested in the process side of her research.

**My user stories:**
1. I need to be able to maintain confidentiality of my data.
2. I need to be able to create my own components and integrate them with existing experiments.
3. I need to be able to setup new benchmarks for comparative purposes.
4. I need to be able to program components in C#.
Meet Jack.

Jack is married and has two young children. He has recently been hired by the TRACY project into the role of Software Architect/Developer. He has 6 years of experience as a software developer and 2 years as a lead architect in a successful gaming company. He has taken the job on the TRACY project because he is excited by the challenge of working in a research oriented project.

Jack is very motivated to build a high quality product. Jack has never worked in an academic research setting before. He is very collaborative and is looking forward to working with the other developers, academics, and students on the project.

My user stories:
1. I need to develop the TraceLab framework in a language which supports rapid prototyping.
2. I need the framework language to easily interface with, and call, components written in other languages.
3. I need the platform to provide natural support for the separation of model and view components.
4. I need libraries to support GUI development.

Meet the full ensemble...

Tom

Glen
Age: 23
MS Student at Hillsbury College
Glen is an MS student who has been helping his advisor to build TraceLab components. He has never contributed to an open source project before, so he needs to figure out how to make contributions to TraceLab. Glen is very collaborative and is looking forward to working with the other researchers on the project.

Wayne
Age: 46
Technical Project Mgr
ABC Corp
Wayne is the technical manager for a very large systems engineering project. He could be described as an early adopter, as he prides himself in keeping an eye out for good ideas that could help his organization. Wayne wants to improve the efficiency of traceability practices in his organization and is interested in using TraceLab.

Mary
Age: 51
NSF Program Officer
Mary is the funding officer for the grant. She is concerned that the project delivers on time and ultimately meets all major goals in terms of adoption, research advancements, and technology transfer.
Understand key concerns

<table>
<thead>
<tr>
<th>Decision</th>
<th>Platform/Language</th>
<th>Tom</th>
<th>Janet</th>
<th>Karly</th>
<th>Jack</th>
<th>Mary</th>
<th>Wayne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pertinent user stories:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>US 1</td>
<td>The system must run on multiple platforms</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>US 2</td>
<td>Users must be able to write and integrate components from multiple languages</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>US 3</td>
<td>The source language of each component must be invisible at runtime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>US 4</td>
<td>The selected language/platform must support rapid framework prototyping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>US 5</td>
<td>The selected GUI must deliver ‘hottie dazzle’</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
</tbody>
</table>

Architectural Decisions

<table>
<thead>
<tr>
<th>Decision</th>
<th>Platform/Language</th>
<th>Tom</th>
<th>Janet</th>
<th>Karly</th>
<th>Jack</th>
<th>Mary</th>
<th>Wayne</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD 1</td>
<td>Build framework using Visual Studio.net and C#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD 2</td>
<td>Develop the initial Windows-specific GUI in WPF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD 3</td>
<td>Utilize MVVM (model view model) architectural pattern, so that (a) the GUI is loosely coupled and can be later implemented using GTK or Windows Forms and compiled for multiple platforms, and (b) the TraceLab engine can be compiled using Mono for porting to Linux and Mac environments</td>
<td>½</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process steps:
1. Analyze persona needs.
2. Identify primary drivers.
3. Extract all related user stories.
4. Assign to personas.
5. Brainstorm architectural design solutions and evaluate leading contenders.
6. Evaluate against personas.

Design solutions for key concerns

<table>
<thead>
<tr>
<th>Decision</th>
<th>Platform/Language</th>
<th>Tom</th>
<th>Janet</th>
<th>Karly</th>
<th>Jack</th>
<th>Mary</th>
<th>Wayne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural Decisions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD 1</td>
<td>Build framework using Visual Studio.net and C#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD 2</td>
<td>Develop the initial Windows-specific GUI in WPF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD 3</td>
<td>Utilize MVVM (model view model) architectural pattern, so that (a) the GUI is loosely coupled and can be later implemented using GTK or Windows Forms and compiled for multiple platforms, and (b) the TraceLab engine can be compiled using Mono for porting to Linux and Mac environments</td>
<td>½</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Risks

<table>
<thead>
<tr>
<th>Decision</th>
<th>Platform/Language</th>
<th>Tom</th>
<th>Janet</th>
<th>Karly</th>
<th>Jack</th>
<th>Mary</th>
<th>Wayne</th>
</tr>
</thead>
<tbody>
<tr>
<td>R 1</td>
<td>The Mono library may not support latest features of C#; Better support for Linux than Mac.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R 2</td>
<td>Build first for Windows solution may lead to multiple GUIs to maintain in the long run.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Long running OS project. Initial tests showed adequate support. Mitigate risk through frequent Mono compiles throughout the project.

Personal impacts

<table>
<thead>
<tr>
<th>Decision</th>
<th>Platform/Language</th>
<th>Tom</th>
<th>Janet</th>
<th>Karly</th>
<th>Jack</th>
<th>Mary</th>
<th>Wayne</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI 1</td>
<td>Tom &amp; Mary’s needs are partially met through this solution. In the long-term researchers will be able to use TraceLab in Linux, but early releases will run on Windows only.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PI 2</td>
<td>All other persons impacted directly by platform/language decisions are positively impacted by this decision.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process steps:
7. Identify architectural risks associated with the proposed solution and their mitigations.
8. Consider and document impacts upon personas.
Architectural design

Supports build-now/port-later decision

Decision 2: Workflow architecture

Options
- Pipe-and-filter
- Services
- Precedence graph + Blackboard
Decision 2: Workflow architecture

Our approach is generalizable..

We created five Architecturally Savvy Personas for a Mechatronics Traceability project that we are working on with Siemens.

The personas highlighted different kinds of concerns from those highlighted by the TraceLab personas.

Elaine is a mechanical engineer with over 20 years of experience working for Company X. She is in charge of modeling the mechanisms for a railway gate. Her model needs to integrate with other models that describe the signaling process for the railway system. Elaine is aware that the crossing gate must comply to a number of regulatory codes and she would like to be able to view the relevant codes from within her model. Elaine has access rights to update her model and to read requirements.

Elaine: Age 50
Mechanical Engineer

Fast trace retrieval
Access control
Extensibility to new case tools
Interoperability of data formats
Remote access
Trace GUIs as plugins

John is the compliance officer for company X. His job is to ensure that all regulatory codes are met by the delivered product and to generate reports to demonstrate this. He is a very detail-oriented person and takes great pride in his job. No products have ever been recalled under his watch for non-compliance purposes.

John: Age 50
Compliance Officer

Fast trace retrieval
Access control
Extensibility to new case tools
Interoperability of data formats
Remote access
Trace GUIs as plugins

Stanley, Age 50
Compliance Officer

Fast trace retrieval
Access control
Extensibility to new case tools
Interoperability of data formats
Remote access
Trace GUIs as plugins
**SCRUM+ ASPs**

1. Identify preliminary personas

2. Elaborate individual personas and explore quality concerns

3. Explore architectural decisions and trade-offs

4. Select features plus their associated architectural components for the Sprint backlog

5. Update personas

6. Evaluate solution with respect to persona's goals

7. Daily scrum meeting

8. Sprint-sized architectural chunks associated with specific features

9. Backlog tasks expanded by team

10. Construct software, including architecture, incrementally

11. Deliver potentially shippable product

---

**So what did we learn?**

- Emerging and analyzing quality concerns early allowed us to make more informed architectural decisions.
- Sketching out architecturally savvy personas (ASPs) enables us to think about quality concerns in a more tangible way.
- Our approach fits naturally into the SCRUM-like process we had adopted for the project.
- A light-weight approach for integrating NFR-thinking into a fast-paced, agile, development environment.
Talk Outline

- Architecturally Significant Requirements and their impact on architectural design.
  - Focus on agile projects
  - Examples from TraceLab project

- Establishing and utilizing traceability links between quality concerns and code
  - Patterns of traceability
  - Archie tool

- Recovering architectural knowledge
  - Machine learning techniques

Change Cycle: Ideal World

Ideal World: Architectural information is documented during the Architectural design phase and is updated regularly to reflect the current system architecture.

Slide used courtesy of Mehdi Mirakhorli
Change Cycle: Real World

Real World: Architectural information is outdated and does not reflect the current architecture of the system.

*Slide used courtesy of Mehdi Mirakhorli*

Architectural Degradation

1. Intended and implemented architecture diverge.

2. Architecture violations (i.e. strict layering bypassed, or pipe-and-filter pipeline violated); cyclic dependencies; dead code; code clones; metric outliers etc.

System becomes brittle starts to erode.
Requirements traceability is the ability to describe and **follow the life of a requirement**, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases."


We can use the Softgoal Interdependency Graph (SIG) notation to capture the goal refinements that lead to our architectural decisions.
Tracing Concerns to Code

Only certain kinds of architectural decisions are traceable to code.

Customized Views

A custom view shows the impact of the architectural decision to pass data using serialization, on higher level quality concerns.
Customized Views

A persona/user perspective upon architectural decisions.

Some decisions occur across multiple projects

Can we find better ways to trace quality concerns to code when common architectural decisions are made?
Some decisions recur across projects

Due to complexity of the problem, we tackled tactics first.

- Tactics are pervasive in fault-tolerant and/or high-performance systems.
- Tactics seem to have an interesting relationship to change.

Tactic Occurrence Across Projects

Tactics tend to be found in safety-critical, and/or other kinds of performance-centric systems.
Tactic Traceability Patterns


Archie...
Talk Outline

- Architecturally Significant Requirements and their impact on architectural design.
  - Focus on agile projects
  - Examples from TraceLab project

- Establishing and utilizing traceability links between quality concerns and code
  - Patterns of traceability
  - Archie tool

- Recovering architectural knowledge
  - Machine learning techniques

Trace Retrieval

In contrast, architectural concerns are often NOT unique in individual systems – so we can train our traceability engine to recognize them across projects.
Our tactic detector uses a previously designed classifier – now implemented in TraceLab.

**Classification Approach**

1. Normalizes the frequency with which term $t$ occurs in the requirement with respect to the length of the requirement.

\[
Pr_Q(t) = \frac{1}{N_Q} \sum_{r \in S_Q} \frac{freq(r, t)}{|r|} \cdot \frac{N_Q(t)}{N(t)} \cdot \frac{NP_Q(t)}{NP_Q}
\]

2. Computes the likelihood that requirement $r$ traces to Query $q$.

\[
Pr_Q(r) = \frac{\sum_{t \in T_Q} Pr_Q(t)}{\sum_{t \in T_Q} Pr_Q(t)}
\]

# Towards Automation

<table>
<thead>
<tr>
<th>Tactic Name</th>
<th>Document trained indicator terms</th>
<th>Code trained indicator terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heartbeat</td>
<td>heartbeat, fault, detect, msgag, period, watchdog, send, tactic, failur, aliv</td>
<td>heartbeat, ping, beat, heart, hb, outbound, puls, hsr, period, isolin</td>
</tr>
<tr>
<td>Scheduling</td>
<td>priori, schedul, assign, process, time, queue, robin higher, weight, dispatch</td>
<td>schedul, task, priori, prcb, sched, thread, rip, weight, tsi</td>
</tr>
<tr>
<td>Authentication</td>
<td>authent, password, kerbero, sasl, ident, biometric, verifi, prove, ticket, purport</td>
<td>authent, credenti, challenge, kerbero, auth, login, otp, cred, share, sasl</td>
</tr>
<tr>
<td>Resource Pooling</td>
<td>thread, pool, number, worker, task, queue, executor, creat, overhead, min</td>
<td>pool, thread, connect, sparow, nbp, processor, worker, timewait, jdbc, ti</td>
</tr>
<tr>
<td>Audit Trail</td>
<td>audit, trail, record, activ, log, database, access, action, monitor, user</td>
<td>audit, trail, wizard, pwriter, lthread, log, string, categori, pstmt, pmr</td>
</tr>
</tbody>
</table>

## Tactic-Grained Classification

![Graphs showing f-measure for Scheduling, Resource Pooling, and Heartbeat with different classification thresholds for both description-trained and code-trained scenarios.]

**Legend**
- - - - 0.001 term threshold
- - - - 0.005 term threshold
- - - - 0.01 term threshold
- - - - 0.05 term threshold
- - - - 0.1 term threshold
Tactic-trained Classification / Code Trained

Audit Trail

Authentication

Legend

0.001 term threshold
0.005 term threshold
0.01 term threshold
0.05 term threshold
0.1 term threshold

HADOOP Case Study

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Class Count</th>
<th>Explanation</th>
<th>Package name or Subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heartbeat</td>
<td>12</td>
<td>HDFS uses a master/Slave architecture with replication. All slaves send a</td>
<td>MapReduce Subsystem</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>heartbeat message to the master server indicating their health</td>
<td>HDFS Subsystem</td>
</tr>
<tr>
<td>Resource</td>
<td>36</td>
<td>MapReduce uses Thread pooling to improve performance of many tasks e.g. to</td>
<td>Mapreduce Package</td>
</tr>
<tr>
<td>Pooling</td>
<td>7</td>
<td>run the mapper function.</td>
<td>Compress Package</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td>Block pooling is used to improve performance of the distributed file system.</td>
<td>HDFS subsystem</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Combines scheduling and job pooling. Organizes jobs into pools, and shares</td>
<td>MapReduce Subsystem</td>
</tr>
<tr>
<td></td>
<td>88</td>
<td>resources between pools.</td>
<td>Common and MapReduce</td>
</tr>
<tr>
<td>Scheduling</td>
<td>4</td>
<td>Audit log capture</td>
<td></td>
</tr>
<tr>
<td>Audit Trail</td>
<td>35</td>
<td>Users Kerberos subsystems.</td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td>The MapReduce</td>
<td></td>
</tr>
</tbody>
</table>
More Challenging: Identifying Roles

Reliability goal helps
Availability goal helps

Rationale justifies Heartbeat tactic is realized by Requirement

Emitter sends pulse maps
Receiver is monitored by Fault Monitor

<<Component> Fault Monitor

Finding Roles is Hard

We integrated light-weight structural approaches – but only evaluated them in a single case study.
Using Generated Links to mitigate Architectural Decay

- Are automatically reconstructed traceability links good enough for use?
- Evaluated the usefulness of the generated fine-grained traceability links for supporting software maintenance.
- Utilized Hadoop change logs for the past four releases, and simulated the scenario in which generated links were used to control the generation of notification messages.
Managing quality concerns (aka NFRs) is a complete life-cycle activity.

- Elicit them early
- Design to satisfy them
- Preserve them
- If necessary, rediscover them
Tackle cutting edge problems in software traceability.

Build a supportive community of researchers.

PRESERVING, GENERATING, AND VISUALIZING KNOWLEDGE OF ARCHITECTURALLY SIGNIFICANT REQUIREMENTS IN SOURCE CODE

Institute for Software Research
Distinguished Speaker Series
University of California, Irvine
April 19th, 2013
Dr. Jane Cleland-Huang
DePaul University

Research funded by the US National Science Foundation under Major Research Instrumentation Grants CCF-0959924 and CCF-1265178.