Grand Challenges in Software Engineering

Phil Laplante, PE, PhD
Professor of Software Engineering
April 2009
What Are Grand Challenges?

• CRA Grand challenges
  – Systems you can count on
  – A teacher for every learner
  – 911.net (ubiquitous information systems)
  – Augmented cognition
  – Conquering complexity

• “Wicked problems in software engineering”
  – NP complete problems?
  – Custom solutions?
  – Let’s see...

• 10 \rightarrow 5 \rightarrow 3
Wicked Problems

• Wicked problems have ten characteristics: (Rittel and Webber 1974)

  ❄️ There is no definitive formulation of a wicked problem.
  ❄️ Wicked problems have no stopping rule.
  ❄️ Solutions to wicked problems are not true-or-false but good-or-bad.
  ❄️ There is no immediate and no ultimate test of a solution to a wicked problem.
  ❄️ Every solution to a wicked problem is a "one-shot operation"; because there is no opportunity to learn by trial-and-error, every attempt counts significantly
Wicked Problems

- Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that maybe incorporated into the plan.
- Every wicked problem is essentially unique.
- Every wicked problem can be considered a symptom of another problem.
- The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution.
- The planner (designer) has no right to be wrong.

THIS IS SOFTWARE ENGINEERING!
Dealing with Wicked Problems

• Rittel and Webber mean wicked problems to be of an economic, political and societal nature (e.g. hunger, drug abuse, peace in the Middle East...)

• They offer no appropriate solution strategy for Software Engineers or anyone else

• But, it is helpful to view software Grand Challenges as a wicked problem
Fighting Complexity

• Software complexity can be found at three levels
  – Code (e.g. methods – McCabe cyclomatic complexity)
  – Design (e.g. class– cohesion, C&K coupling metrics)
  – Architecture (e.g. package – XS, no real measures)

• Code complexity will increase (at all levels) over time if not managed
High Cohesion, Low Coupling
Low Cohesion, High Coupling
Dependency Graphs

a) high coupling with cycles; b) low coupling and no cycles.
Java Imaging API
Structure 101

• Tool by Headway software for analyzing code structure
• Replace “Review” tool which determined more than 60 different structural measures.
• “fat" refers to the interdependencies in a given package, and
• "tangle" refers to cyclic dependencies between packages.
• Objective measure of complexity – XS is combination of “Fat” and “Tangle”
Complexity (Fat and Tangle)

- $m_1 = \text{code rot rate}$
- $m_2 = \text{design decay rate}$
- $m_3 = \text{architectural degradation}$

- Code refactoring initiative (one or more iterations of these possible)
- Design restructuring initiative (one or more iterations of these possible)
- Architecture reengineering needed

Size (Lines of code and or number of classes)

time
Whether it is a code refactoring initiative or a design restructuring depends on which is reduced (fat for code, tangle for design).
A Real-World Example

• Case involved CAD software for semiconductor industry.
• Client develops software in cooperation with business partner
• Business partner accuses client of stealing “design”.
• Analysis of client and partner code bases showed major differences based on code complexity.
Complexity Summary

• Complexity needs to be constantly measured and managed.
• No universal measures of complexity.
• No universal techniques (automated) for managing it.
• Many software engineers choose to ignore it.
Requirements Engineering in the Age of “Agile”

• Key to cost-efficient, satisfactory software is Requirements Engineering (RE)
• Organizations generally don’t do effective RE
• People don’t know what “agile” is
• People use “agile” as an excuse to avoid proper RE
• There is a structured framework for RE in agile methodologies that works – if applied faithfully
The Complexity of Requirements Engineering

• Difficulty in describing even the simplest of “repeatable” human endeavors
• Consider the task of waking up in the morning
• Consider the task of cutting the lawn
• Consider the task of food shopping
• Now consider a complex information system
Requirements Engineering in Agile Methodologies

- A major difference between traditional and agile methodologies is in the gathering of requirements.
- Requirements engineering approaches for agile methodologies tend to be much more informal.
- Some proponents of agile methodologies use requirements engineering practices as a selling point for agile.
### Requirements Engineering: Agile versus Traditional

<table>
<thead>
<tr>
<th></th>
<th>“Traditional”</th>
<th>Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>When requirements gathered?</strong></td>
<td>Usually at the front end of the process</td>
<td>Always ongoing</td>
</tr>
<tr>
<td><strong>Who gathers the requirements?</strong></td>
<td>One or a few requirements engineers</td>
<td>All developers</td>
</tr>
<tr>
<td><strong>Customer interaction</strong></td>
<td>Limited to requirements engineers</td>
<td>All developers</td>
</tr>
<tr>
<td><strong>Feedback to customers on requirements implementation</strong></td>
<td>Sporadic, mostly at the end during acceptance testing</td>
<td>Ongoing</td>
</tr>
<tr>
<td><strong>Customer involvement</strong></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Vulnerability to changing requirements</strong></td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
Agile Requirements Change Management Process

High Priority

Each iteration implement the highest-priority requirements

Each new requirement is prioritized and added to the stack

Requirements may be reprioritized at any time

Requirements may be removed at any time

Low Priority

Modeled in lesser detail

Modeled in greater detail

Source: Ambler 2007
A Real-World Example

• US Government project (very high profile – in the news right now).
• Version 2.0 of a system that was very difficult to use.
• Contractor chose to use “Agile” development to deal with ongoing stream of requirements changes.
• Project has been a disaster.
Software Security

• Threat management
  – Malware prevention
  – Detection
  – Arms race
  – Is an ongoing problem. Is it a “Grand Challenge”?

• Software security architectures
  – This is a “Grand Challenge”
What is a Software Architecture?

- The structure and organization by which modern system components and subsystems interact to form systems.
- The properties of systems that can best be designed and analyzed at the system level.
- Need to avoid studying security at low-level design and code idiom level.

Source: Kruchten, Obbink, Stafford
Modern Software Architecture

• Seminal work

• Constructed a model of Software Architecture consisting of 3 components
  – Elements: processing, data and connecting elements
  – Form: the choice of architectural elements, their placement and how they interact
  – Rationale: motivation for choice of elements and form

• Boehm later added the notion of “constraints”
Common Architectural Styles

- Shaw and Garlan some common architectural styles
  - Pipes and filters
  - Objects
  - Implicit invocation
  - Layering
  - Repositories
  - Interpreters

Vulnerabilities?
Some Architectural Antipatterns

• Architecture by Implication: lack of architectural specifications for a system under development.
• Design by Committee: everything but the kitchen sink.
• Reinvent the Wheel: avoid.
• Stovepipe Enterprise: a bunch of architectures cobbled together.
• Stovepipe System: legacy qualities.
• Vendor Lock-In: captive systems.

Vulnerabilities?
Research Question

• How do we compare architectural structures that withstand security attacks better than another?
• We may find those more vulnerable if we focus on bug reports of many systems.
• Open Source systems may provide supply of projects to examine.
Approach #1

• Identifying all the architectural patterns claiming that they strengthen security
  – find common characteristics (or salient features) of these structures, which makes them security-relevant.
• Use these characteristics as criteria (or guidelines) to sift through the architectural structures associated with security flaws.
• Repeat analysis to find out the common characteristics of the structures that make them particularly vulnerable to security attacks.
• In essence, we propose to find antipatterns for security-relevant architectural structures.
Approach #2

• What about standards based evaluation?
  – ASSET (Automated Security Self-Evaluation Tool) (NIST)
  – OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) from SEI
  – COBIT (Control Objectives for Information and Related Technology) from Information Systems Audit and Control Association (ISACA)
Approach #3

1. Analyze bug reports; Identify the security attacks and bugs first.
2. Filter obvious non-software architecture misuse-related attacks (we need to deal with other issues)
3. Analyze the architecture [and investigate whether security integrity contributes to vulnerabilities]
4. See if the problem is recurring
An Illustration

Model View Controller Architecture
• Model-view-controller (MVC) architecture
  – Isolates business logic from interface concerns
• Spring is a widely used framework for rapid construction
• Several reports on weak security architecture
• Other reports (see end notes)
• How do we prevent these issues?


Spring problem 1: MVC automatic form field setting using java beans can be exploited to modify the data for another user’s account.

Spring problem 2: User-controllable data is subject to SQL injection style of attack.
MVC Class Structure


Behavior of Passive Model
How Do We Deal With Grand Challenges?
Technology Solutions

• Formal Methods?
• New tools?
• New approaches?

• OR...
Rational Solutions

• Process purity
  – CMMI level 5?
  – Six sigma?
  – ITIL, IVI, CoBIT, ...

• Better communications and expectation setting?

• Managed “sanity”?  
  – Self-mastery
  – Team management...
Current Research (2005-present)

- Focus Areas
  - Requirements engineering
  - Software project management
  - Software testing
  - Security issues

- My “grand challenges” come from these areas
Contact Me

plaplante@psu.edu
www.personal.psu.edu/pal11