Software Engineering: Issues, Trends and Research Directions

Doris L. Carver
Professor, Department of Computer Science
Associate Vice Chancellor
Office of Research and Graduate Studies
Louisiana State University
dcarver@lsu.edu
January 22, 2004
What is Software Engineering?

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

Software Costs

• Catastrophic
  – Security breaches
  – Invasion of privacy
  – Loss of life

• Financial
  – DOD invests an estimated $38 billion per year in research, development, testing and evaluation of software
  – NASA has many missions in the $200 million range
“People see the mechanical system of this vehicle, parachutes coming out, airbags coming out, and bouncing on the ground. But that is a software-driven system, and a lot of the project risk is in how robust that software is”.

Tom Gavin concerning the Mars Exploration Rover - COMPUTER, January 2004
Software Challenges

• Build reliable systems

• Develop tools and principles that allow construction of large-scale systems that are highly trustworthy

• Computing Research Association –
  – http://www.cra.org/Activities/grand.challenges
Build Reliable Systems

- Fundamental research problems
  - System development tools that reduce the frequency and severity of bugs
  - New approaches to the composition of modular elements
  - System administration tools that reduce the frequency and severity of configuration errors
  - Understandable, deployable, and usable security

http://www.cra.org/Activities/grand.challenges
System development tools that reduce the frequency and severity of bugs

- Increase quality of code
- Improved languages and environments
- Improved bug-fix processes
- Better modularization techniques
- Easier installation techniques

http://www.cra.org/Activities/grand.challenges
New approaches to the composition of modular elements

• Improved ways to compose subsystems
  – Components incomplete
  – Components inconsistent
  – Components not reliable

• Improved ways to determine properties of systems built from components
  – Better property descriptions
  – Better functional descriptions
  – Better methods to deduce information about compositions

http://www.cra.org/Activities/grand.challenges
Growing problem

Managing Complexity
Techniques for Managing Complexity

• Pattern
  – Description of a problem and a proven solution to the problem
• Components
  – Building blocks
• Frameworks
  – Collection of components that provide mechanisms for particular domains
Software Development Approaches

Informal

Formal
Practices and Trends in Software Development

• Agile software development methods
• Unified Modeling Language (UML)
• Patterns
• Model Driven Architecture
• Formal development techniques
Practices and Trends in Software Development

- Agile software development methods
- Unified Modeling Language (UML)
- Patterns
- Model Driven Architecture
- Formal development techniques
**Manifesto for Agile Software Development**

- Individuals and interactions over processes and tools
- Customer collaboration over contract negotiation
- Working software over comprehensive documentation
- Responding to change over following a plan

www.agilemanifesto.org
Agile Methodologies

- XP (Extreme Programming)
- Scrum
- Cockburn’s Crystal Family
- Highsmith’s Adaptive Software Development
- Feature Driven Development
- Dynamic System Development Method
Extreme Programming (XP) Fundamentals (Beck)

- Starting projects with a simple design
  - constantly evolves to add needed flexibility and remove unneeded complexity

- Producing all software in pairs, two programmers at one screen

- Turning minimal system into production quickly
  - growing it in whatever directions prove most valuable
Extreme Programming (XP) Fundamentals (Beck)

- Writing unit tests before programming keep all of the tests running at all times
- Integrating and testing the whole system several times a day
Write user stories
- User-written, scenarios, (use cases)

- Develop release schedule
  - Develop individual iteration schedule
    - Developers are assigned tasks

- Make frequent small releases

- Measure project velocity
  - how much work is getting done on your project.
  - add up the estimates of the user stories that were finished during the iteration
XP Rules - Planning

- Divide schedule into iterations of 1 to 3 weeks
  - Against rules to look ahead and begin implementing anything not scheduled for this iteration

- Move people around
  - Team is more flexible if everyone knows enough about every part of the system to work on it.
  - Pair programming – Collective code ownership
    - Sit side by side in front of monitor
    - Both use keyboard and mouse as needed
    - One person types and thinks tactically about the method being created
    - Other person thinks strategically about how method fits into the class
    - Awkward at first
XP Rules- Planning

• Daily stand up meeting
  – communicate problems, solutions, and promote
team focus
  – everyone stands up in a circle to avoid long
discussions.

• Fix it when it's broken
  – teams can change rules
XP Rules – Designing – (Beck)

• Keep design simple
  – Don’t add functionality before it is scheduled
• Build spike systems (programs to explore solutions to specific problems – throwaway)
• Don’t add functionality that is not scheduled
• Refactor frequently
  – remove redundancy
  – eliminate unused functionality
  – rejuvenate obsolete designs
XP Rules – Coding – (Beck)

- Customer is always available
- Use coding standards
- Code the test case before the code
- All production code is pair programmed
- Only one team tests, integrates, and releases code to the repository at one time
- Integrate often
- Leave optimization to the end
- No overtime
XP Rules – Testing – (Beck)

- All code must have unit tests
- All code must pass all unit tests before it is released
- Create an acceptance test before debugging
- Acceptance tests are run often and the results published
Survey – Cutter Consortium

- Average length of projects using agile methodologies

<table>
<thead>
<tr>
<th>Time Period</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 months</td>
<td>6%</td>
</tr>
<tr>
<td>1-3 months</td>
<td>25%</td>
</tr>
<tr>
<td>3-6 months</td>
<td>39%</td>
</tr>
<tr>
<td>6-9 months</td>
<td>17%</td>
</tr>
<tr>
<td>9-12 months</td>
<td>11%</td>
</tr>
</tbody>
</table>
Survey - Cutter Consortium

- **Size of team**

<table>
<thead>
<tr>
<th>Team Size</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2 people</td>
<td>4%</td>
</tr>
<tr>
<td>3-10 people</td>
<td>51%</td>
</tr>
<tr>
<td>11-20 people</td>
<td>21%</td>
</tr>
<tr>
<td>21-50 people</td>
<td>14%</td>
</tr>
<tr>
<td>51-100 people</td>
<td>5%</td>
</tr>
</tbody>
</table>
Use of Agile Methods

• When to use
  – Systems with uncertain requirements
  – Responsible developers
  – Engaged customers

• When to avoid
  – Large teams
  – Contractual work

(Martin Fowler)
XP Controversies (Beck)

- Doesn’t force team members to specialize
  - every XP programmer participates in all of the critical activities every day
- Doesn’t conduct complete up-front analysis and design
  - starts with quick analysis of system
  - then programmers continue analysis and design throughout implementation
- Infrastructure and frameworks developed as application is developed
- Don’t write and maintain implementation
  - communication occurs face-to-face or through tests and code
Agile Methods

- Adaptive rather than predictive
- People-oriented rather than process-oriented
- Self-adaptive process
  - Regular reviews
- Current state
  - How can we blend agile methods with traditional methods?
  - Focus on scalability, adaptability
More Information

- June 2003, COMPUTER, Special Issue on Agile Methods
  - Laurie Williams, North Carolina State University
  - Alistair Cockburn, Humans and Technology
Practices and Trends in Software Development

- Agile software development methods
- *Unified Modeling Language (UML)*
- Patterns
- Model Driven Architecture
- Formal Methods
What is UML?

- De facto industry standard
- Language for specifying, visualizing, constructing and documenting the artifacts of software
- Provides easy to use visual modeling language to develop models
- Specification independent of any programming language
- Supports high-level development concepts such as components, frameworks, and patterns
UML Diagram Types

- Use case
- Behavioral
  - Interaction
  - Sequence
  - Collaboration
- Statechart
- Activity
- Component
- Deployment
- Static Structure
- Class
- Object
Data Flow between Views

Use Case Model

Interaction Model

State Model

Object Model

Classifiers & Attributes

Interactions

Events

Classes

Operations

States

[Piamsa-nga03]
Practices and Trends in Software Development

- Agile Software Development Methods
- Unified Modeling Language (UML)
- Patterns
- Model Driven Architecture
- Formal Methods
Patterns

• “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice –
  – Christopher Alexander
  – Over 250 patterns for architectural design

• Distinctive format
  – Short name
  – Brief description of the context
  – Lengthy description of the problem
  – Prescription for a solution
Patterns

- Recurring solutions to common problems
- An effective means of reusing, sharing, and building upon existing experience
- Convey insight to novices
- Types of patterns
  - Design (architecture, design, programming idioms)
  - Analysis (recurring and reusable analysis models)
  - Organizational (structure of organizations/projects)
  - Process (software design process)
  - Domain-specific
- www.hillside.net/patterns/
Iterator Pattern

• Access the elements of an aggregate object
  – Linked list, input stream, token stream, …

• Context
  – Object (aggregate) contains other objects (elements)
  – Clients (methods that use the aggregate) need access to elements
  – Aggregate should not show its internal structure
  – Multiple clients may need simultaneous access
Iterator Pattern

• Solution
  – Define an iterator class that retrieves one element at a time
  – Each iterator object keeps track of the location of the next element to be retrieved
  – Implement a common interface type
Iterator Pattern

<<interface>>
Aggregate

createIterator()

Client

<<interface>>
Iterator

Next()
isDone()
currentItem()

returnvalue
nextItem()

List

Concrete Aggregate

Linked List

Concrete Iterator

ListIterator

Software Engineering Laboratory
Practices and Trends in Software Development

- Agile software development methods
- Unified Modeling Language (UML)
- Patterns
- Model Driven Architecture
- Formal Methods
Model Driven Architecture (MDA)

- Uses models to drive
  - Understanding, design, construction, deployment, operation, maintenance and modification
- Uses viewpoints
  - Focus on specific concerns in a given system
- Relies on transformations to convert one model to another
MDA

- Specify a system independently of the platform
  - Specify platform
  - Choose a platform for a system
  - Transform the system specification into one suitable for a particular platform

- Primary goals
  - Portability
  - Interoperability
  - Reusability

- Focus on separation of concerns
MDA

- MDA viewpoints
  - Computation Independent Model (CIM)
    - Domain model
  - Platform Independent Model (PIM)
    - Describes system functionality, not how it will be build
  - Platform Specific Model (PSM)
    - Model of system described in the PIM but including how the model makes use of the specifics of the platform.

http://www.omg.org/docs/omg/03-06-01.pdf
Transformation Types

• Horizontal transformation
  – Improves quality attributes of model
  – Source and target models same level of abstraction
  – Refactoring – type of horizontal transformation
• Vertical transformation
  – Triggers corresponding transformations in other models in order to maintain consistency
  – Source and target models different levels of abstraction
  – Refinement and abstraction – type of vertical transformations
Model Refactoring

- Defines how to restructure semantics and preserve certain properties during the transformation of models
- Redesigning of a model without changing its outward behavior
- Occurs when a model is changed to enhance its quality attributes without modifying its outward behavior
- Aim to deliver a new architecture to the model that will be able to accommodate new requirements
Model Transformation

Source Pattern \rightarrow Transformation Pattern \rightarrow Target Pattern

Source Model \rightarrow model transformation \rightarrow Target

M1' \rightarrow «instance» \rightarrow T1 \rightarrow «instance» \rightarrow M2'
Practices and Trends in Software Development

• Agile software development methods
• Unified Modeling Language (UML)
• Patterns
• Model Driven Architecture
• Formal Methods
Formal Methods

• Integral part of a growing number of mission critical systems

• Integral part of the Mars Science Laboratory mission

• Mars Science Laboratory mission
  – Land a rover as big as a car for a tour lasting more than a year
  – Scheduled for 2009
  – $200 million range
NASA’s Software Development Plans for the Future

“NASA’s Mission Reliable”
Patrick Regan, Scott Hamilton, COMPUTER, January, 2004
Mission Data System (MDS)

- Models
- Patterns
- Components
- Frameworks
- Transformations
- Automation
Mission Data System (MDS)

- Multi-mission framework for building, testing, and reusing software that will fly in spacecraft, land in rovers, and operate here on Earth.
- Artificial intelligence-based autonomy software
- Automated model checking as alternative to conventional methods of software testing
Mission Data System (MDS)

- **Modular (compositional verification)**
  - Verify properties of components
    - Ensure correctness of the reconstituted whole
    - Based on generating assumptions
    - Separating controllable characteristics of environment from the uncontrollable characteristics

- **Conventional methods**
  - Codify best practices
    - use them as they are used in other engineering disciplines
Focus of NASA’s Next Generation Systems

- State-based architecture
- State becomes first class
  - State variables, estimators, controllers, state value histories, state effects models, measurements, time constraints, state constraints
- More formal expression of inputs and outputs
- Traditional - modular, hierarchical decomposition
  - Doesn’t work well in unpredictable environments
NASA’s Next Generation Environment

• Vision
  – To generate 95% of Mars Science Laboratory code automatically

• Automatic code generation

• Transformations

• Model checking for testing and validation
Future Trends

• Increased reliance on models
• Expanding use of model transformations
• Growing use of patterns, components, and frameworks in domain-specific applications
• Continued use of informal techniques for smaller systems
• Codifying and usage of best practices
Software Engineering Research and Practice

• “In theory there is no difference between theory and practice, but in practice there is” – author unknown
Questions?