Reducing the Technical Debt of Software Systems

Clemente Izurieta, PhD.
Software Engineering Laboratory (SEL)
Department of Computer Science
Montana State University
PRIMER
A series of steps necessary to reach a desired state
OBJECT ORIENTED

• This technique is based on the organization of entities that have structure and behavior:
  
  • Objects are abstractions of real world entities
  • Structure and behavior of an entity is encapsulated
  • Common characteristics are generalized
ABSTRACTION

Smart House
Structural Characteristics:
- Style
- Square Footage
- Number of Rooms

Behavior:
- What is your power consumption this month?
- Set the thermostat

Described by:
**Smart House CLASS**

**Structural Characteristics:**
- Style
- Square Footage
- Number of Rooms

**Behavior:**
- What is your power consumption this month?
- Set the thermostat
ENCAPSULATION

Must send a message to object to request an operation

Smart House
Structural Characteristics:
- Style
- Square Footage
- Number of Rooms
Behavior:
- What is your power consumption this month?
- Set the thermostat
ORGANIZATION

- Associations
  - Simple relationships between classes
- Generalization
  - Abstracting out common features
  - Supports reuse – the sharing of common information
- Aggregation
  - Grouping of components based upon a whole-parts relationships
ASSOCIATION

Smart House
Style

Style
Description
Cost
...
A design pattern is an agreed upon way of solving a problem.

A design pattern captures experience.
A common problem:

- **Environmental Sciences**: Sensor data from field monitoring systems (tsunami warnings)
- **Media**: Newspaper or magazine subscriptions
- **Business**: Stock ticker
OBSERVER PATTERN

Subject

Observers

FLBS
MSU
UM

One Many

1 *
OBSERVER PATTERN

Concrete Subject
- Subject
  - registerObserver()
  - removeObserver()
  - Notify()

Observers
- FLBS
- ConcreteObserver
  - Update()
- UM

One Many

1 *
OBSERVER PATTERN

Concrete Subject

ConcreteObserver

Observer

RegisterObserver()
Notify()
RemoveObserver()

Notify()

Update()
SOFTWARE MAINTENANCE

• Useful systems are modifiable by design
• Such systems need constant modification in order to remain useful
• Most such systems are too expensive to replace, so considerable resources go into their maintenance
• However, maintenance, even more than development, is characterized by tight budgets, staffing, and time constraints

Ref: [1]
TECHNICAL DEBT

• Technical Debt is the gap between:
  • Making a change perfectly
    • Preserving architectural design
    • Employing good programming practices and standards
    • Updating the documentation
    • Testing thoroughly
  • And making the change work
    • As quickly as possible
    • With as few resources as possible
    • With programmers rather than software engineers

Ref: [1]
EVERYDAY INDICATORS OF TECHNICAL DEBT

“Don’t worry about the documentation for now.”

“The only one who can change this code is Rob”

“It’s ok for now, Isaac will refactor it later!”

“ToDo/FixMe: this should be fixed before release”

“Let’s just copy and paste this part.”

“I know if I touch that code everything else breaks!”

“Let’s finish the testing in the next release Ryan!”

“The release is coming up, so just get it done!”

Ref: [1]
COSTS AND BENEFITS OF INCURRING TECHNICAL DEBT

• Benefits
  • Higher software productivity in the current release
  • Lower cost of current release

• Costs
  • “Interest” – increased maintenance costs
  • Risk that the debt gets out of control
TECHNICAL DEBT

- *Technical Debt Principal*

- Refers to the effort required to complete a task that is left undone (Right Now). A task is a representation of a technical debt item that runs a risk of causing future problems if left undone.
TECHNICAL DEBT

• *Technical Debt Interest Amount*

• Refers to an estimate of the amount of extra work that will be needed to maintain the software if a technical debt item is not repaid. Interest incurs a continuing cost to its associated item.
TECHNICAL DEBT

- *Technical Debt Interest Probability*
- Refers to the probability that the technical debt, if not repaid, will make other work more expensive over a given period of time or a release.
MANAGING TECHNICAL DEBT

• It is not possible to remove all technical debt
• It is not practical to remove all technical debt
• It will happen…

• Is there higher ROI in releasing as is vs. letting interest accumulate?
VISUALIZATION OF TECHNICAL DEBT
Sample of well architected software
“Pleasantville”

codecity.inf.usi.ch
NEO V1
NEO V1 DISHARMONIES
NEO V2
NEO V2 DISHARMONIES
Δ compliance = f (instance, description)
DESIGN PATTERN CONFORMANCE CHECKING
TECHNIQUES THAT ALLOW FOR THE MANAGEMENT OF TECHNICAL DEBT

• Design for extensibility
  • Interfaces, not implementation
  • Design Patterns

• Refactoring techniques
  • Automated and manual

• Treating defects or technical debt items with similar priority

• Maintaining
  • current documentation and
  • current test suites