Carnegie Mellon University

Modularity for the Modern World

Mary Shaw

Institute for Software Research
Carnegie Mellon University

Start with definitions

Modularity (n) from "modular" +"ity“

- "ity" is a suffix used to form abstract nouns expressing state or condition
- so modularity is the state of being modular

Start with definitions

Modularity (n) from "modular" +"ity“

- "ity" is a suffix used to form abstract nouns expressing state or condition
- so modularity is the state of being modular

Modular (adj)

- related to, or based on, a module or modules

Start with definitions

Modularity (n) from "modular" +"ity“

- "ity" is a suffix used to form abstract nouns expressing state or condition
- so modularity is the state of being modular

Modular (adj)

- related to, or based on, a module or modules

Module (n) from Latin “modulus”

- a standardized, often interchangeable, component of a system that is designed for easy assembly or flexible use
- a self-contained component that is used in combination with other components

Examples of modules

Computer Science: A portion of a program that carries out a specific function and may be combined with other modules to form a program

Electronics: A self-contained assembly of electronic components and circuitry that is installed as a unit

Education: A unit of instruction in which a small topic or a section of a broad topic is studied for a given period of time

Manufacturing: A prefabricated self-contained standard unit than can be combined with other modules to assemble a wide range of end products

Astronautics: A self-contained unit that performs a specific task in support of the major function of the craft

Architecture: The size of some one part taken as a unit of measure to regulate the proportions of the composition

… about the definition …

Module is defined in terms of elements such as component, unit, portion, part, assembly …

- … with the property that a module is intended to be combined with other modules to build something
- … with constraints arising from intentions about its use
- The precise nature of the element depends on the purpose and context of module (re)use

Modularity shows the use of *divide-and-conquer*

- The criteria for separating the modules is "*divide*"
- We also need rules for combining them ("*conquer*")

My objective today is to
offer a framework for describing modules
– and hence modularity –
that will open a discussion about
modularity in general and
new opportunities for the AOSD community

Why modularize?

Intellectual control

- Localize representations to separate decisions
- Separate concerns to enhance understandability
- Use standard architecture to reuse design knowledge

Segmentation of work

- Localize decisions to separate responsibilities
- Match work assignments to skills
- Factor tasks to match bodies of knowledge

Evolution and reuse

- Localize decisions to simplify change
- Standardize units to support reuse and substitution
- Support market for reusable parts

Historical examples

Interchangeable parts -- early mass production ~1800

- firearms (Blanc Whitney), sailing blocks (Brunel)
- modular >> facilitate mass production and repair
- not evidently intended to enable new arrangements

Natural language -- enable new compositions

- Minimal unit varies by language (letters to pictographs)
- Grammars provide rules for combining words
- Composition by concatenation or merger (“compose”+”tion”)

Money -- standard modular units of value

- Coins originally had intrinsic value (weight in silver)
- Notes originated in receipts for goods in a warehouse
- Value became abstract when banks or governments guaranteed their value

Understanding modularity

To understand a modularity strategy, identify …

Scope:

- domain, generality, homogeneity, abstraction

Content:

- what's in a module

Criteria:

- how does the designer decide on module boundaries

Organization:

- how are the module definitions organized?

Composition:

- how are modules combined?

Examples of Traditional Modularity

- General strategies
  - Functions/subroutines
  - Data abstraction/objects
  - Concurrency
  - Architectures
- Problem-specific strategies
  - Model-view-controller
  - Scribe text formatter

Set up a basis for comparing modularity strategies

Traditional functions and subroutines

Scope :

- stateless functionality, general purpose, homogenous, abstract definitions match concrete implementation

Content:

- algorithm, corresponding to code

Criteria:

- localize reusable algorithms, package common functions

Organization:

- hierarchical nested definitions

Composition:

- function/procedure call

Traditional functions and subroutines

Scope :

- stateless functionality, general purpose, homogenous, abstract definitions match concrete implementation

Content:

- functionality (algorithm), corresponding to code

Criteria:

Intellectual control

- localize reusable algorithms, package common functions

Evolution/reuse

Segmentation of work

Organization:

- hierarchical nested definitions

Composition:

- function/procedure call

Data abstractions, objects

Scope :

- localize representation, general purpose, homogenous, abstract definitions partly match concrete implementation

Content:

- representation and related operations

Criteria:

- localize data representation and related operations
  - maximize cohesion and coupling

Organization:

- flat, independent definition space

Composition:

- function/procedure call

Data abstractions, objects

Scope :

- localize representation, general purpose, homogenous, abstract partially matching concrete, manage variations

Content:

- representation and related operations; relative definitions

Criteria:

- localize data representation and related operations

Organization:

- hierarchical inheritance

Composition:

- function/procedure call, dynamically bound
- inheritance (abstractly, it's function call under the covers)

Concurrency

Scope :

- asynchronous concurrency, general purpose, homogenous, abstract matching concrete

Content:

- thread algorithm and synchronization

Criteria:

- separate tasks into threads; synchronize to avoid conflict

Organization:

- flat definition space with interactions between definitions

Composition:

- synchronization, data sharing with locks

Architectures

Scope :

- coarse-grained system organization, general purpose, heterogeneous, abstract mostly matching concrete

Content:

- subsystems: databases, processes, data streams, servers

Criteria:

- identify large functional units and their relations

Organization:

- recognize different types of subsystems, flat within types

Composition:

- subsystem interaction (communication & data) protocols

Examples of Traditional Modularity

General strategies

- Functions/subroutines
- Data abstraction/objects
- Concurrency
- Architectures

Problem-specific strategies

- Model-view-controller
- Scribe text formatter

Set up a basis for comparing modularity strategies

Model-View-Controller

Scope :

- user interface to interactive system, special purpose, heterogeneous, abstract matching concrete

Content:

- M: algorithms for system; V: user interface; C: mapping

Criteria:

- separate concerns of underlying model and interaction

Organization:

- coordinated definitions

Composition:

- stylized: assigned roles to components

Text Markup (Scribe, 1981)

Scope :

- document markup data, special purpose, heterogeneous, interpretive

Content:

- document & markup; rendering (style), device properties

Criteria:

- separate document content and markup, formatting style, printing device definition

Organization:

- independent text documents

Composition:

- interpreter applies style & device definitions to document

Issue: Structure clashes

Module definitions are often organized hierarchically

- This is a widespread and very useful approach
- Alas, a static document can have only one hierarchy

A system may have multiple distinct concerns, each with a reasonable definition hierarchy

- Some language devices try to address structure clashes
  - Aspects, multiple inheritance, flavors, etc
- “Cross-cutting concerns” recognize the problem
- Multiple hierarchies can sometimes coexist when concerns are orthogonal

Exercise:

Describe aspects in this framework

Computing in the modern world

Two significant trends …

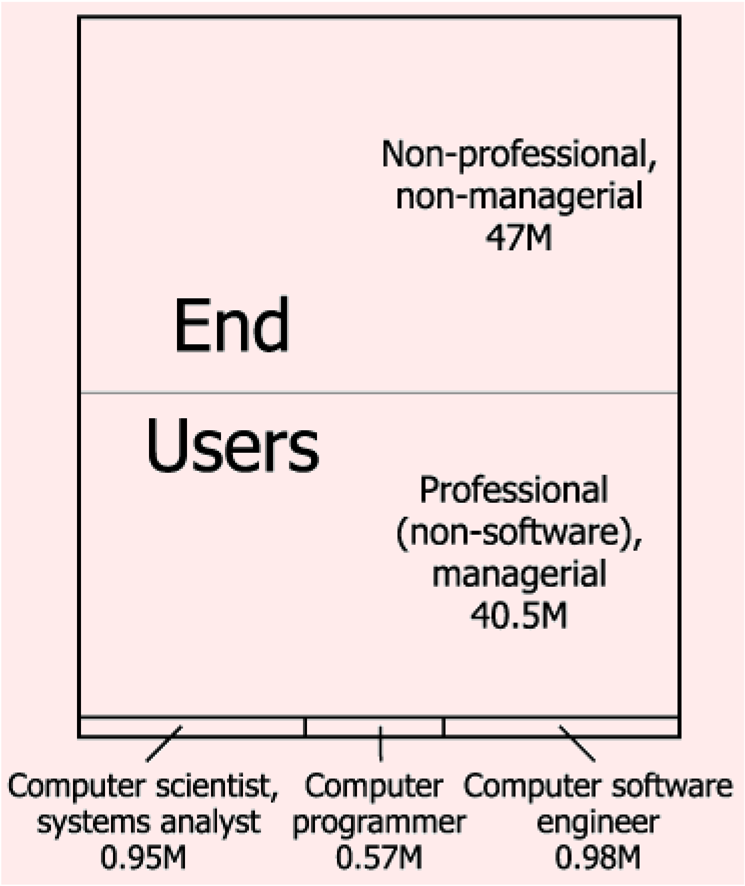
End user development

- Professional software developers are vastly outnumbered by developers whose principal expertise lies elsewhere

Ultra large scale systems

- The traditional model of discrete software projects with managers and clear objectives is becoming obsolete

There are *lots* of end users

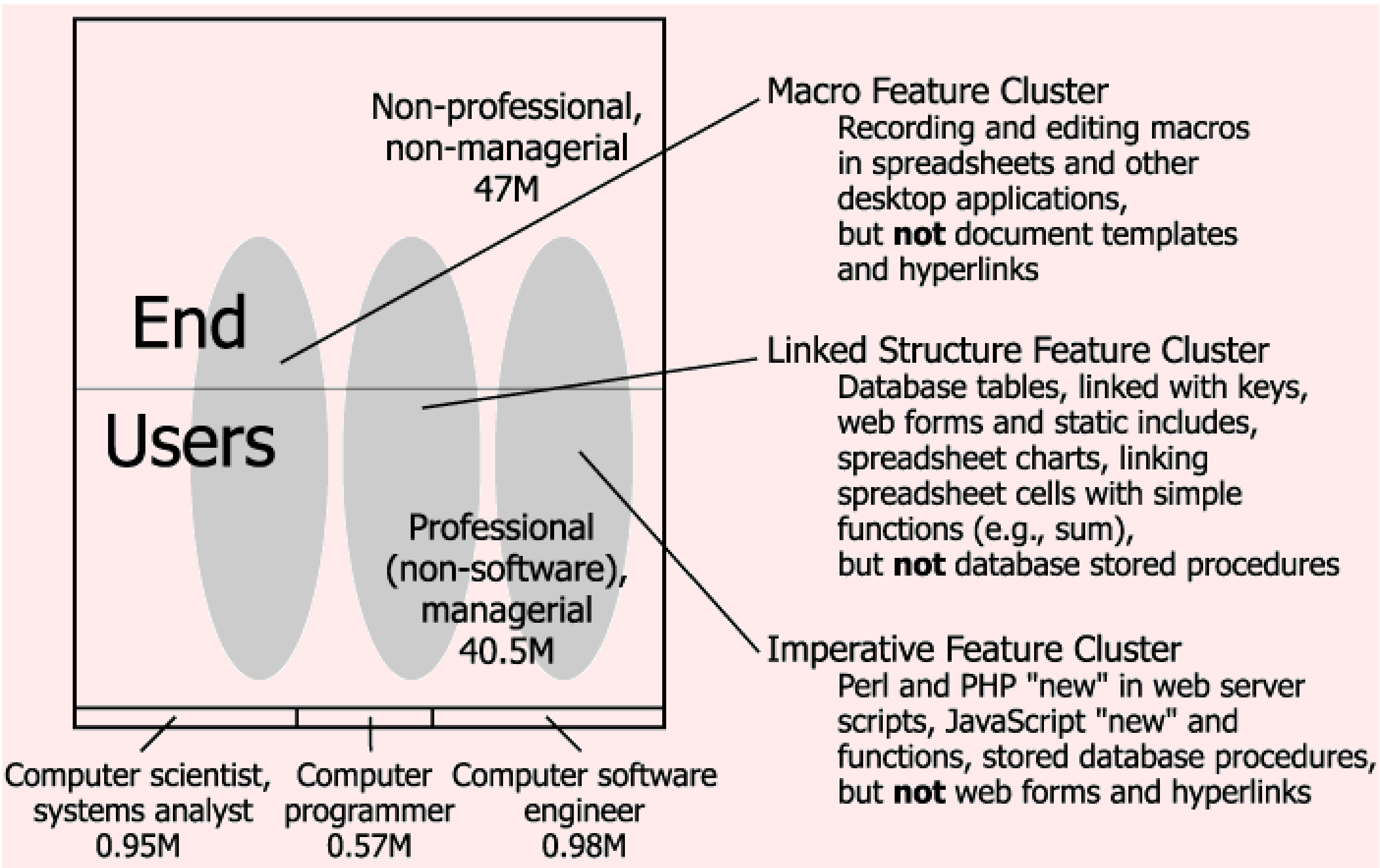


Using data from the Bureau of Labor Statistics, we estimate that over 90M Americans will use computers at work in 2012. Of these, only about 2.5M will be professional programmers; 40.5M will be managers and (non-software) professionals.

This does not include home users or non-US users, so there will be many more than 90M total end users. Most of them will "program" in some way.

C. Scaffidi, M. Shaw, and B. Myers. Estimating the Numbers of End Users and End User Programmers. *VL/HCC'05: Proc 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 207-214, 2005.

They are not all alike



Analysis of web-based survey of Information Week readers

C. Scaffidi, Andrew Ko, B. Myers, and M. Shaw. Dimensions Characterizing Programming Feature Usage by Information Workers. *VL/HCC'06: Proc2006 IEEE Symp on Visual Languages and Human-Centric Computing*, pp. 59-62,

Internet resources

Information: unstructured text, formatted text, databases, live data feeds, images, maps, current status (e.g., inventory, location)

Calculation: reusable software components, applications that can be invoked remotely (e.g., services)

Communication: messages, social networking, streaming media, synchronous communication, agent systems, alert/notification services

Control: coordination for use of resources, access to registration and subscription services

Services: simulation, editorial selection, evaluation, secondary (derived) information, responsive experts, markets

Properties of internet resources

Autonomous

- Independently created and managed
- May change structure or format without notice

Heterogeneous

- Different packagings, output often for viewing only
- Different business objectives, conditions of use

Open affordances

- Independent systems, not dependent components
- Incidental effects may be useful
- Humans integral to some resources

Ultra-Large-Scale Systems

Large size on many dimensions

- Lines of code, amount of data, users, dependencies among and complexity of components, etc

More than "systems of systems"

Characteristics

- Decentralized operation and control
- Conflicting, unknowable, diverse requirements
- Continuous evolution and deployment
- Heterogeneous, inconsistent, changing elements
- Indistinct people/system boundary
- Normal failures
- New forms of acquisition and policy

SEI. Ultra-Large-Scale Systems. 2006

Analogy: Cities and city planning

Cities are complex systems

- Built of individual components chosen by individuals
- Constantly evolve
- Withstand failures and attacks

Cities are not centrally controlled

- Standards for infrastructures
  - Building codes, highway standards
- Policies that allow individual action within constraints
  - Zoning laws
- Regulations that govern individual action
  - Enforcement after the fact, rather than prior constraint

“Wicked problems”

Modern modularity challenges

- General strategies
  - Cloud computing
  - Web modularity
  - Architectural integration
- Problem-specific strategies
  - Yahoo pipes
  - Web page definitions
  - Large-scale fine-grained parallelism

Set up a basis for comparing modularity strategies

Cloud computing

Many providers offer commodity-grade computing services over the internet

- distributed computing power
- storage
- applications
- "software as a service"

"The cloud" can be used in many ways; focus on service-oriented computing

Service oriented computing

Scope :

- commodity services, general-purpose, architecturally homogeneous, currently implementation-oriented

Content:

- coarse-grained interchangeable computation and storage

Criteria:

- independent units in support of business processes, usually with service guarantees

Organization:

- distributed definitions with discovery services

Composition:

- orchestration: discover services, establish contracts, marshal data; interact through defined protocols

Web modularity

Composing information from the web

- Add-ons, plug-ins, extensions for incrementing base system
- Mashups: opportunistic repurposing
  - Currently lacks good modularity and other abstractions
- Smartphone apps
- Participatory web (Web 2.0): user-generated content, interoperability, “network as platform”
  - Social networking lacks good modularity and other abstractions

Annotating and merging data

- Semantic web (Web 3.0): annotated data and data fusion

Semantic web

Scope :

- annotated data, general-purpose, homogeneous, aspires to semantic abstraction

Content:

- data extensively annotated with metadata

Criteria:

- data is not restructured from its natural form; metadata enables identification of matching elements

Organization:

- ontologies and metadata tags on data elements

Composition:

- interchange formats; matching tags used to establish correspondence

Architectural integration

Software architecture has gone beyond standalone systems to distributed compositions of existing components, systems, and services

CONNECT project

- general integration of heterogeneous components

Medical informatics

- new initiatives in interoperability and integration

Modern modularity challenges

General strategies

- Cloud computing
- Web modularity
- Architectural integration

Problem-specific strategies

- Yahoo pipes
- Web page definitions
- Large-scale fine-grained parallelism

Set up a basis for comparing modularity strategies

Yahoo pipes

Scope :

- aggregate data feeds, special-purpose, homogeneous, abstract and concrete

Content:

- RSS data feeds and similar streams

Criteria:

- data is not restructured from its natural form, but only feeds such as RSS feeds are supported

Organization:

- I have found it hard to find useful pipes

Composition:

- visual interface supports filtering, merging, and other remixing of feeds; result is itself a feed

Web page definitions

Scribe criteria remain viable: separate document definition into tagged document, style, & rendering

On the web

- the web page is the annotated document
- the CSS file, template, or content management system is the style
- the browser is responsible for rendering

Opportunity: the web page itself incorporates document markup, algorithm, state management, and structure information

- structure (XML) is modular, but the others are mingled
- that is, the web page is mudular

Large-scale fine-grained parallelism

MapReduce (tightly synchronized)

- automatically parallelize computations over large data sets to run scalably and robustly on large clusters of commodity machines

Grid computing with volunteered resources

- factor very large computations into independent asynchronous units that can be delegated to diverse low-end platforms; must be robust to individual failures
- for example SETI@home
- BOINC has 2.5M users, 6M hosts, >200 countries

Framework for discussing modularity

To understand a modularity strategy, identify …

Scope:

- domain, generality, homogeneity, abstraction

Content:

- what's in a module

Criteria:

- how does the designer decide on module boundaries

Organization:

- how are the module definitions organized?

Composition:

- how are modules combined?

Types of module content

Functionality (algorithm)
Representation
Relative definition (inheritance deltas)
Control (threads)
Subsystems
Document and markup, data plus metadata
Rendering
Device properties
Services
Connectors (protocols)
Invocation vs augmentation (function vs plugin)
Data feeds (e.g., RSS)

Types of composition

Function, procedure, subroutine calls
Dynamically bound calls
Inheritance (for defining variants)
Synchronization
Subsystem interaction (communication & data) protocols
Interpretation
Stylized; assigned roles for components
Services orchestration (discovery, contracts, …)
Interchange language/protocol/representation
Filtering, merging, remixing
MapReduce
Plugin callback (for augmentation)

Types of composition

Function, procedure, subroutine calls
Inheritance (for defining variants)
Synchronization
Interpretation
Stylized; assigned roles for components
Services orchestration (discovery, contracts, …)
Interchange language/protocol/representation
Filtering, merging, remixing
MapReduce
Plugin callback (for augmentation)
Weaving

Note:
These are abstractions. In current technology, most are actually implemented with procedure calls

Status of this framework

Brooks proposed recognizing three kinds of results, with individual criteria for quality:

- findings -- well-established scientific truths -- judged by truthfulness and rigor
- observations -- reports on actual phenomena -- judged by interestingness
- rules-of-thumb -- generalizations, signed by an author (but perhaps not fully supported by data) -- judged by usefulness

with freshness as criterion for all

This framework is certainly not a finding; I present it as a rule-of-thumb and a basis for discussion

Frederick P. Brooks, Jr. Grasping Reality Through Illusion -- Interactive Graphics Serving Science. Proc ACM SIGCHI Human Factors in Computer Systems Conference, May 1988, pp. 1-11.

My objective today is to offer a framework for describing modules – and hence modularity – that will open a discussion about modularity in general and new opportunities for the AOSD community

My objective today is to offer a framework for describing modules – and hence modularity – that will open a discussion about modularity in general and new opportunities for the AOSD community

Particularly interesting opportunities go beyond code to address problems of real users in the real interactive, connected world