O

O S E

E

Domain Engineering

Operating-System Engineering

What is Domain Engineering?

Domain Engineering is the activity of collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (i.e., reusable work products), as well as providing an adequate means of reusing these assets (i.e., retrieval, qualification, dissemination, adaptation, assembly, and so on) when building new systems. ([2], p. 20)

What is a Domain?

Domain An area of knowledge ([2], p. 34):

- scoped to maximize the satisfaction of the requirements of its stakeholders
- includes a set of concepts and terminology understood by practitioners in that area
- includes the knowledge of how to build software systems (or parts of software systems) in that area

Defined by the consensus of its *stakeholders*, i.e., people having interest in a given domain: managers, marketing people, developers, vendors, contractors, standardization bodies, investors, customers, and end-users.

Domain Engineering as a Three-Phase Process

Domain Analysis Systematic organization of the existing (i.e., recorded) domain knowledge in a way that enables and encourages extensions to be made in creative ways:

- select and define the domain of focus
- collect all relevant domain information
- integrate the domain information into a coherent *domain model* → p. 4

Domain Design Development of an architecture for the *family of systems* in the domain and to devise a production plan.

Domain Implementation Involves implementing the architecture, the components, and the production plan using appropriate technologies.

Domain Model

A *domain model* is an explicit representation of the *common* and the *variable* properties of the systems in a domain, the semantics of the properties and domain concepts, and the *dependencies* between the variable properties. ([2], p. 23)

Constituents of a Domain Model

domain definition Defines the scope of a domain and characterizes its contents by giving examples of existing systems in the domain, counterexamples, and generic rules (including the rationale) of inclusion or exclusion of a given system or capability.

domain lexicon Defines the domain vocabulary.

concept models Describe the concepts in the domain in some appropriate modeling formalism (e.g., {object, interaction, state-transition, entity-relationship, data-flow} diagrams) and informal text.

feature models Define a set of reusable and configurable requirements for specifying the systems in a domain.

→ p. 6

Feature and Feature Model

Feature A reusable and configurable requirement.

Feature Model Used in domain analysis to capture the commonalities and variabilities of systems in a domain:

- prescribes which feature combinations are meaningful, which of them are preferred under which conditions and why
- represents the configuration aspect of the concept models previously described and, in the end, the configuration aspect of the whole reusable software

Is the basis for developing the means of ordering concrete systems during application engineering.

Activities of Domain Analysis

domain scoping Identifies the domain of interest, the stakeholders and their goals, and defines the scope of the domain

- domain selection and description
- data source identification and inventory preparation

Aims at finding a domain scope that is economically viable and promises business success.

domain modeling Yields the domain model:

- data collection and analysis, taxonomic classification, evaluation

Software Architecture

When developing a software architecture, we have to consider not only functional requirements, but also non-functional requirements, such as performance, robustness, failure tolerance, throughput, adaptability, extendibility, reusability, and so on. Indeed, one of the purposes of software architectures is to be able to quickly tell how a piece of software satisfies its requirements. ([2], p. 27)

Architectural Patterns (1)

layers An arrangement into groups of subtasks in which each group of subtasks is at a particular level of abstraction.

pipes and filters An arrangement that processes a stream of data, where a number of processing steps are encapsulated in filter components. Data is passed through pipes between adjacent filters, and the filters can be recombined to build related systems or system behavior.

blackboard An arrangement where several specialized subsystems assemble their knowledge to build a partial of approximate solution to a problem for which no deterministic solution strategy is known.

Architectural Patterns (2)

broker An arrangement where decoupled components interact by remote service invocations. A broker component is responsible for coordinating communication and for transmitting results and exceptions.

model-view-controller A decomposition of an interactive system into three components: A model containing the core functionality and data, one or more views displaying information to the user, and one or more controllers that handle user input. A change-propagation mechanism ensures consistency between user interface and model.

microkernel An arrangement that seperates a minimal functional core from extended functionality and custom-specific parts. The microkernel als serves as a socket for plugging in these extensions and coordinating their collaboration.

Multi-Architecture Design

- no single architectural pattern is superior to the others:

  Real architectures are usually based on more than one of these and other patterns at the same time. Different patterns may be applied in different parts, views, and at different levels of an architecture. ([2], p. 28)

- which (and how many) of the patterns to apply depends . . .
  - on the specific application ↑
  - on the specific platform ↓
  - on the domain → p. 2

- preferences for specific architectural patterns must never be a dogma

Architectural Design

high-level design Its goal is to come up with a flexible structure that satisfies all important requirements and still leaves a large degree of freedom for the implementation.

> As a rule, we use the most stable parts to form the "skeleton" and keep the rest flexible and easy to evolve. But even the skeleton has to be modified sometimes. ([2], p. 28)

Architectures and their Amount of Flexibility

generic architecture Can be thought of as a fixed frame with a number of sockets where we can plug in some alternative or extension components:

- components and sockets must clearly specify their interfaces
- the resulting system has a fixed topology and fixed interfaces

highly flexible architecture Supports <u>structural variation</u> in its topology:

- it can be configured to yield a particular generic architecture
  - even the skeleton has been componentized
- it allows us to negotiate and configure interfaces

An architecture for a system family has to include an explicit representation of the variablity (i.e., configurability) it covers. $\rightarrow$ *configuration languages*

Production Plan

- describes how to build a system from the common architecture and components that is to say, a production plan describes . . .
  - the interface to the customers ordering concrete systems
  - the process of assembling the components
  - the process of handling change requests and custom development
  - the measuring, tracking, and optimizing of the production process
- makes up the second artifact to be developed during domain design

Automation Levels of the Assembly Process

manual assembly Architecture and components are accompanied by a developer's guide, and applications have to be assembled from components manually.

automated assembly support The assembly of components is supported with various tools including component browsing, search tools, and generators automating selected aspects of application development.

automatic assembly A set of tools supports the process of ordering an application by a customer, and the entire application can be generated based on the order record.[1] → *generative programming*

[1] Excluded therefrom are parts requiring customary development.

Application Engineering

- the process of building systems based on the results of domain engineering

  During the requirements analysis for a new concrete application, we take advantage of the existing domain model and describe customer needs using the features (i.e., reusable requirements) from the domain model. ([2], p. 30)

- new customer requirements should be fed back to domain engineering
  - to be able to refine and extend the reusable assets

- finally, the application is assembled manually or generated automatically

Family Orientation

Conventional software engineering concentrates on satisfying the requirements for a *single system*, whereas Domain Engineering concentrates on providing *reusable* solutions for a *family of systems*. ([2], p. 21)

$\underline{\textit{domain}}\ \{\textit{analysis, design, implementation}\}$

Bibliography

[1] G. Arango. Domain Analysis Methods. In W. Schäfer, R. Prieto-Diaz, and M. Matsumoto, editors, *Software Reusability*, pages 17–49. Ellis Horwood, New York, 1994.

[2] K. Czarnecki and U. W. Eisenecker. *Generative Programming—Methods, Tools, and Applications*. Addison-Wesley, 2000. ISBN 0-201-30977-7.

[3] S. Wartik and R. Prieto-Diaz. Criteria for Comparing Domain Analysis Approaches. *Interational Journal of Software Engineering and Knowledge Engineering*, 2(3):403–431, Sept. 1992.