Journal of Software Engineering and Applications, **2012, 5, 1-6**
http://dx.doi.org/10.4236/jsea.2012.51001 Published Online January 2012 (http://www.SciRP.org/journal/jsea)

Guidelines Based Software Engineering for Developing Software Components

Muthu Ramachandran

Faculty of Arts, Environment and Technology, School of Computing and Creative Technologies, Leeds Metropolitan University, Leeds, UK.
Email: m.ramachandran@leedsmet.ac.uk

Received October 20th, 2011; revised November 25th, 2011; accepted December 10th, 2011

ABSTRACT

Software guidelines have been with us in many forms within Software Engineering community such as knowledge, experiences, domain expertise, laws, software design principles, rules, design heuristics, hypothesis, experimental results, programming rules, best practices, observations, skills, algorithms have played major role in software development. This paper presents a new discipline known as Guidelines Based Software Engineering where the aim is to learn from well-known best practices and documenting newly developed and successful best practices as a knowledge based (could be part of the overall KM strategies) when developing software systems across the life cycle. Thereby it allows reuse of knowledge and experiences.

Keywords: Software Reuse; Software Guidelines; Software Design Knowledge; CBSE; GSE

1. Introduction

The term Software Engineering was coined by F. L. Bauer the chairman of 1968 NATO Software Engineering conference held in Garmisch, Germany to promote a disciplined approach to developing software. The term Software is meant a list of machine instructions where as the Engineering is meant the use of disciplined approaches and laws when building software systems. This paper would argue that the term Software should include best practices which are the laws due to the nature and the age of software as a science compared with Science and Engineering where the laws have been proved and established. In the world of software our principles are out current practices and are continue to emerge as we speak. Later, the term algorithm has emerged to provide a structured step by step programmable instructions/solution to a software problem. Best practices provide a step by step instructions/solution to software problem across the life cycle and are based on the successful use in real world.

Guidelines provide a precise set of steps based on underlying software design principles which help us to follow any course of disciplined set of activities. The term guidelines are defined in the dictionary as follow:

- A recommended approach, parameter, etc. for conducting an activity or task, utilizing a product, etc.;
- A statement of desired, good or best practice;
- Advice about how to design an interface;
- A document used to communicate the recommended procedures, processes, or usage of a particular business practice;
- A recommendation that leads or directs a course of action to achieve a certain goal;
- A written statement or outline of a policy, practice or conduct. Guidelines may propose options to enable a user to satisfy provisions of a code, standard, regulation or recommendation.

Software Engineering is a set of disciplined activities that are based on well defines standards and procedures. In Software Design we use guidelines that help us to identify a suitable design criterion when faced with design decisions. Therefore software guidelines summarises expert knowledge as a collection of design judgements, rationales, and principles. This can be used by students/ engineers when learning about new principles with examples and experts alike.

2. Guidelines Based Software Engineering

The very definition of Software Engineering deals with best practices, disciplined & systematic approaches to software development and management. These best practices have been found throughout software development life cycle. Starts from good program design by Parnas [1], Algorithms design by Dijkstra [2], concurrent programs by Hoare and they all have provided good design guidelines which are applicable until now. The term best prac-

tices should support knowledge and wisdom that has emerged from many years of successful use across several projects, products, programs, and portfolios. Software as a profession, we must also include a list of recommended conduct and ethical activities when developing software product or research. Once we accept the term *Software Guidelines* as a new discipline that provide well established principles and rules that are successful in practice and thus also provide knowledge and wisdom. This way we can also tell the world proudly, we are Engineers since we follow principles strictly and ethically. Where do we start?

In practice we are not sure of the process by which to apply those principles. Therefore, our work on software guidelines have started on specifically on software components [3-5], extended to concurrency, software process improvement, agile methods, and software product line based development (aimed on good practice requirements guidelines). Therefore, we prefer to call *Guidelines Based Software Engineering* (GSE) which aimed to collect best practices and experiences as guidelines from many years of wealth of knowledge and wisdom in Software Engineering and apply them wherever possible across all artefacts of software development. Guidelines provide rationale for making a solution that has worked well and successfully in previous applications, environment, and in people. **Figure 1** shows the process view of guidelines based software engineering.

The process states start with gathering *domain knowledge*, classify domain, classify best practice design, identify artefacts (components, patterns, frameworks), identify and classify best practice design guidelines on various aspect of their design (for example how well requirements have been represented as use cases and how well use case have been used effectively and their features, how well OCL specifications have been used to document and describe the model). Building the domain knowledge is crucial for success of using software guidelines or GSE. We can define domain analysis is an activity for identifying a key set of software artefacts that can be ready-made for reuse. There are numerous approaches to this end which can conclude by summarising a common set of domain analysis process as follow:

1) Setting Domain principles: Select a domain, definitions, business analysis, scope and boundaries and planning.

2) Data collection—learn more about the domain, discover success and failures, and collect guidelines, discover abstractions, review literature extensively, interview and discuss with domain experts, and develop scenarios.

3) Data analysis—the aim is to identify entities, objects, models, sub-domains, related classes and models, events, operations, relationships amongst all of them, tacit knowledge, analyse similarities and variabilities, analyse combinations and trade-offs, cost-benefit analysis, modular decompositions and design decisions.

4) Classification—the aim during this phase is to describe domain classes, models, and components, conduct cluster analysis and HIPO chart, describe artefacts, classify models and components, generalize artefacts descriptions, conduct domain vocabulary.

5) Evaluation of domain models—the aim in the last phase is to evaluate the findings systematically—use expert meeting, reviews, discussions, and review interviews.

In case if the artefacts are represented in any programming language, then identify and classify best design constructs that can be used for expressing various design factors such as reuse, flexibility, security, and so on. Guidelines fall into several categories such as good practice guidelines on requirements engineering (Sommerville and Sawyer [6]), RE methods-specific guidelines such on UML, Use Case driven modelling, design (OO, generic design principles), quality and SQA procedures and best practices, software development (good program design and language-specific guidelines), and good test process guidelines, and guidelines on software process improvement. The first step in building guidelines based SE is to devise a classification system/mechanism for collating guidelines which the useful for finding an appropriate guideline. A number of guidelines, best practices, projects, and knowledge engineering support for software development life cycles are presented by Ramachandran [7].

Best practice guidelines on components based software engineering (CBSE) fall into a number categories such as definitions, process, methods, techniques, models, design, implementation, domain engineering, and development for component reuse, component security, component testing, validation, certification, and QSA. Identifying software components from your application models is a human intensive activity. This comes from domain expertise. However, Pressman [8] has identified a few self-assessment questions to identify components from your design abstracts as given below:

- Is component functionality required on future implementations?
- How common is the component's function within the domain?
- Is there duplication of the component's function within the domain?
- Is the component hardware-dependent?
- Does the hardware remain unchanged between implementations?
- Can the hardware specifics be removed to another component?
- Is the design optimized enough for the next implementation?

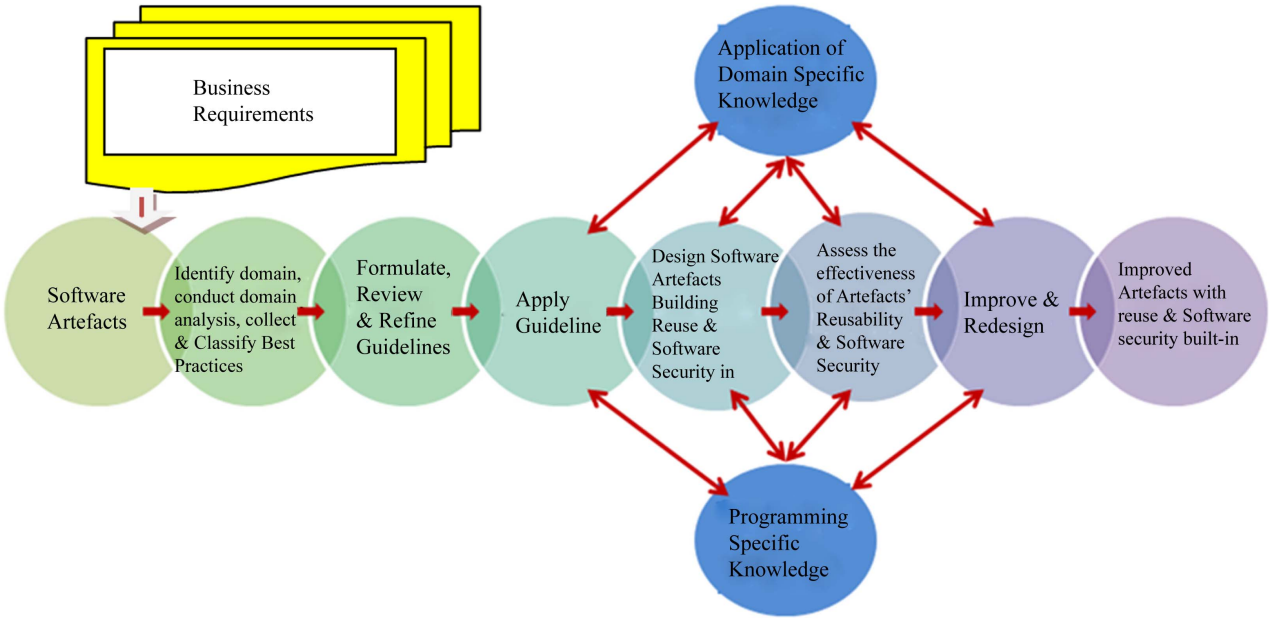

Figure 1. The process of guidelines based software engineering.

- Can we parameterize a non-reusable component so that it becomes reusable?
- Is the component reusable in many implementations with only minor changes?
- Is reuse through modification feasible?
- Can a non-reusable component be decomposed to yield reusable components?
- How valid is component decomposition for reuse?

Example of a Process Guideline for Component Identification: One rule of thumb can be use here is to identify a group of related object classes to make up a self-independent component. UML view of component identification process is depicted in the following diagram (**Figure 2**). UML process starts with identifying use cases, class modeling, dynamic modeling (state transition and message sequence models), collaboration models (grouping related classes), packaging, components, and deployment/implementation models (processors and network architectures) where components and packages will be placed in the expected processors.

Implementation effort and Return on Investment (RoI): This is an initial step in CBSE and it is therefore vital to identify a component which will have a longer life in your application domain and hence high returns on investment. Therefore it is absolutely essential to have a business view to each identified components with domain experts.

Process guidelines have also helped us to identify common processes and patterns across CBSE and reuse. Knowledge about commonly occurring patterns in a process helps to save cost. Therefore, for each guideline, it is important to present a description, illustration, return on investment (RoI), and possible implementation effort required along with cost-benefit analysis.

3. Guidelines, Observations, Empirical Studies to Laws and Theories

Guidelines form principles from observations, laws, and theories. Observations, in software terms, mean to visually able to see changes or results of an experiment/software tools used by people, etc. However, these observations may not be a repeatable event. A law can be defined as repeatable observations according to Endres and Rombach [9]. For example, a rainy season, symptoms of a widespread disease, etc. Theories can help to explain and order our guidelines, observations, and laws. Theories can also help it predict new facts from existing guidelines, observations and laws. The diagram shown in **Figure 3** illustrates the relationships amongst guidelines, observations, law, and theory. Guidelines also add human perspective to observations, laws, and theories as it adds knowledge and experiences.

We have used similar approach to domain-specific modelling to generate reusable software components automatically for several application domains. An example of a CBSE guidelines classification system has been shown in **Figure 4** and their relevant guidelines have been adopted when designing software components [5]. Best practice guidelines on components based software engineering (CBSE) fall into a number categories such as definitions, process, methods, techniques, models, design, implementation, domain engineering, and development for component reuse, component security, component testing,

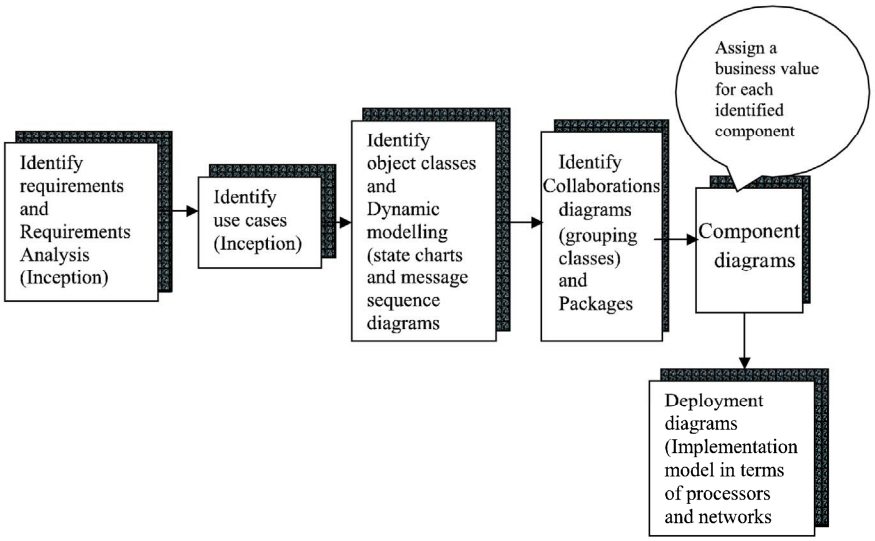

Figure 2. UML view of component identification.

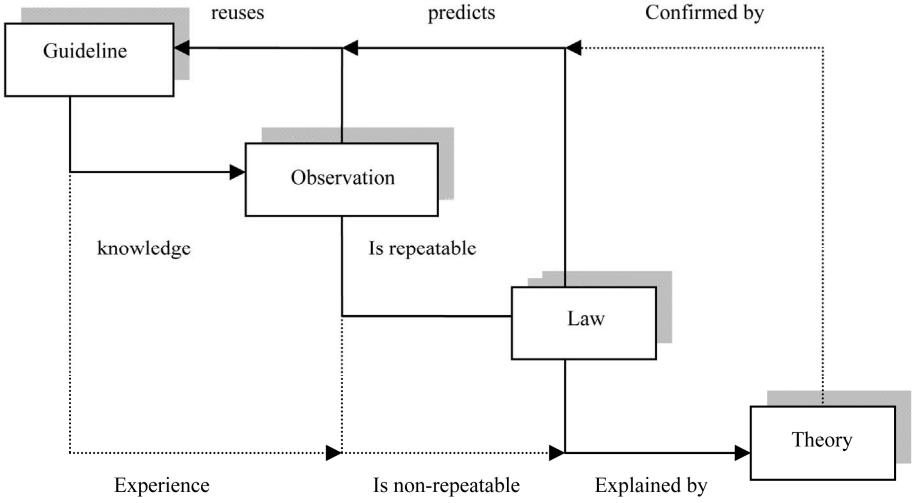

Figure 3. Guidelines, observations, laws, and theories.

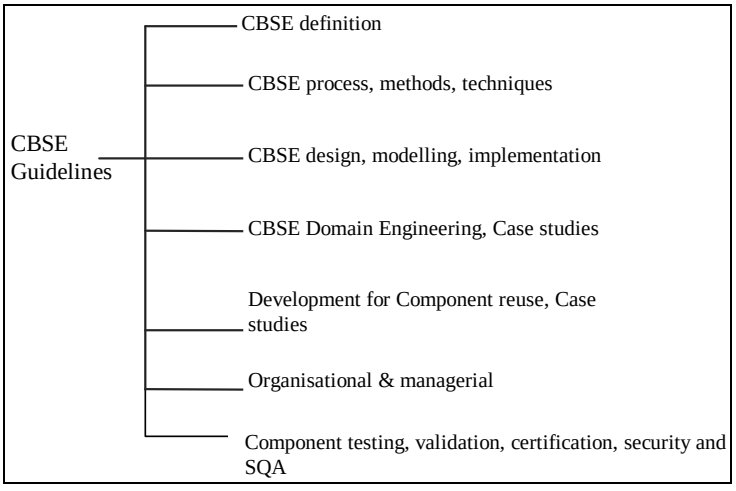

Figure 4. Classification of best practice CBSE guidelines.

validation, certification, and QSA.

Each of these guidelines has been followed against various models for Helpdesk management systems. There were 15 software component identified and their relevant interfaces. Each of these guidelines can also be used to conduct a systematic inspection against use case models, class diagrams, and component diagrams. Therefore, it allows us to achieve fine tuned models that can be further checked against guidelines during implementation as there are plenty of guidelines developed for JavaBeans and C# components. Similar best practices have been presented by many authors [10-21], all of them can be encoded as a knowledge base.

Our earlier results have shown components designed with guidelines seem to have improved reuse and easy to re-design (more than 70% reusability gain has been achieved) for a simple help desk management system. The **Table 1** shows an example of a list of components and their reusability gain in percent.

Reuse gain represents the percent of reusability which is measured against percent of guidelines met. The GUI component 1 consists of a large component for Helpdesk system for the front-end consisting of more than 100 interfaces that can be served to other components. This component has met 50% of the best practice guidelines therefore reusability gain is 50%. Guidelines become highly useful for building software security. This is a new are for research and hence formulating best practice guidelines can help to achieve software security early in the life cycle. According the above data we can see the percent of security-specific design guidelines that have been met. The security design guidelines are further classified into a set of language-specific features (when not to use some features found in most programming practices) and design principles that help to design components for software security built in rather than as add ons.

Our future work includes designing automated tool to predict developing high quality software components that are designed for reuse and quality. This can be achieved by encoding guidelines as knowledge to assess, review and improve components development right from analysis. This will improve component based development with less effort and cost and can be manufactured as a mass production that has been seen in other industry. Due to current improvement in knowledge based technologies, this is will be possible to encode domain knowledge thereby best practice guidelines can be implemented efficiently.

Table 1. Component reusability gain & security guidelines met.

Helpdesk System Components	Reuse Gain	Security Guidelines
GUI component 1	50%	92%
GUI component 2	40%	99%
User records management	70%	95%
Job status	55%	90%
Test components	65%	99.99%

4. Conclusion

Guidelines based SE can create best practices as guidelines to be followed when developing software artefacts. Guidelines provide knowledge and wisdom that has emerged from several years of best practice and experiences in previous projects successfully. This can save time, cost, and effort with quality that we all seek. Our work has shown increase in reuse gains to the maximum of 70%. The security factor can be achieved up to 99%. Thus, we believe, attributes such as reuse and security factors can be improved significantly which results in achieving high quality of the software systems and reducing software development costs.

REFERENCES

[1] L. Parnas, “Good Program Design,” Prentice-Hall, Upper Saddle River, 1979.

[2] E. W. Dijkstra, “Selected Writings on Computing: A Personal Perspective,” ACM Classic Books Series, 1982.

[3] T. Hoare, “Concurrent Programs,” Prentice-Hall, Upper Saddle River, 1979.

[4] M. Ramachandran and Sommerville, “Software Reuse Assessment,” *First International Workshop on Software Reuse*, Germany, 1992.

[5] M. Ramachandran, “Software Components: Guidelines and Applications,” Nova Publishers, New York, 2008. https://www.novapublishers.com/catalog/product_info.php?products_id=7577

[6] I. Sommerville and P. Sawyer, “Requirements Engineering: Good Practice Guide,” Addison Wesley, Boston, 1999.

[7] M. Ramachandran, “Knowledge Engineering for Software Development Life Cycle,” IGI Global, Hershey, 2011. doi:10.4018/978-1-60960-509-4

[8] Pressman, “Software Engineering,” 6th Edition, McGraw Hill, New York, 2005.

[9] A. Endres and D. Rombach, “A Handbook of Software and Systems Engineering,” Addison Wesley, Boston, 2003.

[10] W. A. Brown and K. C. Wallnau, “The Current State of CBSE,” *The Current State of CBSE, IEEE Software*, Vol. 15, No. 5, 1998.

[11] M. Broy, *et al.*, “What Characterizes a Software Component?” *Software—Concepts and Tools*, Vol. 19, No. 1, 1998, pp. 49-56. doi:10.1007/s003780050007

[12] J. Cheesman and J. Daniels, “UML Components,” Addison Wesley, Boston, 2000.

[13] D'Souza and Wills, "Objects, Components and Frameworks with UML," Addison Wesley, Boston, 1999.

[14] G. Eddon and H. Eddon, "Inside Distributed COM," Microsoft Press, Washington, 1998.

[15] G. T. Heineman and W. T. Councill, "Component-Based Software Engineering," Addison Wesley, Boston, 2001.

[16] IEEE SW, "Special Issue on Software Components," *IEEE Software*, Vol. 15, No. 5, 1998.

[17] I. Jacobson, *et al*., "Software Reuse: Architecture, Process and Organisation for Business Success," Addison Wesley, Boston, 1997.

[18] K.-K. Lau and Z. Wang, "A Taxonomy of Software Component Models," *Proceedings of the* 31*st EUROMICRO Conference on Software Engineering and Advanced Applications*, 2005.

[19] O. Rob Van, *et al*., "The Koala Component Model for Consumer Electronics Software," *IEEE Computer*, 2000.

[20] R. Sessions, "COM and DCOM," Wiley, New York, 1998.

[21] C. Szyperski, "Component Software," Addison Wesley, Boston, 1998.

