<Clemson University>

<Pacemaker>
Software Architecture Document (SAD)

CONTENT OWNER: <Yang Cao>

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
● NO.1	● Version 1.0	● Mar.6, 2017
●	●	●
●	●	●
●	●	●
●	●	●
●	●	●

BACKGROUND

This template is based on the Software Engineering Institute’s “View and Beyond” method for documenting software architectures, as described in Clements, et al., *Documenting Software Architecture: Views and Beyond* (Addison Wesley, 2002). The current version is available for free download from the SEI’s architecture web site.

TIPS FOR USING THIS TEMPLATE

To create an instance of this document:

- Insert relevant information on cover sheet and in placeholders throughout.
- Insert relevant information in page header: Move to a page of the body of the report, select *View* > *Header and Footer* from the main menu, and then replace relevant information in the header box at the top of the page.

To update the contents and page numbers in the Table of Contents, List of Figures, and List of Tables:

- Position the cursor anywhere in the table to be updated.
- Click the *F9* function key.
- Answer “Update entire table”.

To insert a figure or table caption:

- From the main menu, choose *Insert* > *Reference* > *Caption* and then either *Figure* or *Table* as needed.
- Click the OK button.
- Add a colon and a tab stop after the figure number in the caption itself.
- The caption should use the *Caption* style.
- Add a colon and a tab stop after the table/figure number in the caption itself.

TIPS FOR MAKING YOUR DOCUMENT MORE READABLE

- A gray box containing *CONTENTS OF THIS SECTION* is provided at the beginning of most sections and subsections. After determining what specific information will be included in your document, you can remove this gray box or leave it to serve as a quick-reference section overview for your readers. In the case that text has been provided in the template, inspect it for relevance and revised as necessary.
- Consider hyperlinking key words used in the document with their entries in the Glossary or other location in which they are defined. Choose *Insert* > *Hyperlink*.
- Don’t leave blank sections in the document. Mark them “To be determined” (ideally with a promise of a date or release number by which the information will be provided) or “Not applicable.”
- Consider packaging your SAD as a multi-volume set of documentation. It is often helpful to break your documentation into more than one volume so that the document does not become unwieldy. There are many ways that this can be accomplished. The structuring of the document must support the needs of the intended audience and must be determined in the context of the project. Each document that you produce should include the date of issue and status; draft, baseline, version number, name of issuing organization; change history; and a summary. A few decomposition options are:
  - *A 2-Volume approach:* Separate the documentation into two volumes; one that contains the views of the software architecture and one that contains everything else. A common variant[1] of this approach has one volume per view, and one volume for everything else.
  - *A 3-Volume approach:* Document organizational policies, procedures, and the directory in one volume, system specific overview material in a second, and view documentation in a third.
  - *A 4-Volume approach:* Create one volume for each viewtype [Clements 2002] (module, component-and-connector, allocation) that contains the documentation for the relevant views. Include all of the other information in the fourth volume.
  - Software interfaces are often documented in a separate volume.

In *any* case, the information should be arranged so that readers begin with the volume containing the Documentation Roadmap (Section 1 in this template).

[1]

Table of Contents

List of Figures

List of Tables

1 Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section 1.1 (“Document Management and Configuration Control Information”) explains revision history. This tells you if you’re looking at the correct version of the SAD.
- Section 1.2 (“Purpose and Scope of the SAD”) explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you’re seeking is likely to be in this document.
- Section 1.3 (“How the SAD Is Organized”) explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- Section 1.4 (“Stakeholder Representation”) explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.
- Section 1.5 (“Viewpoint Definitions”) explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.5, there is a corresponding view defined in Section 3 (“Views”). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.
- Section 1.6 (“How a View is Documented”) explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 Document Management and Configuration Control Information

CONTENTS OF THIS SECTION: This section identifies the version, release date, and other relevant management and configuration control information associated with the current version of the document. Optional items for this section include: change history and an overview of significant changes from version to version.

- Revision Number: << 1.0 >>
- Revision Release Date: << *03/05/2017* >>
- Purpose of Revision: << Create document >>
- Scope of Revision: <<*list sections or page numbers that have been revised; provide a summary overview of the differences between this release and the previous one.*>>

1.2 Purpose and Scope of the SAD

CONTENTS OF THIS SECTION: This section explains the SAD’s overall purpose and scope, the criteria for deciding which design decisions are architectural (and therefore documented in the SAD), and which design decisions are non-architectural (and therefore documented elsewhere).

This SAD specifies the software architecture for **<insert scope of SAD>.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[2] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible” properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement’s behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system’s structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very

[2] Here, a system may refer to a system of systems.

static structure, in that it focuses on the way the system’s functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system’s function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

CONTENTS OF THIS SECTION: This section provides a narrative description of the major sections of the SAD and the overall contents of each. Readers seeking specific information can use this section to help them locate it more quickly.

This SAD is organized into the following sections:

- **Section 1 (“Documentation Roadmap”) provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.
- **Section 2 (“Architecture Background”) explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 (Views”) and Section 4 (“Relations Among Views”) specify the software architecture**.

Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).

- **Sections 5 (“Referenced Materials”) and 6 (“Directory”) provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.5.

CONTENTS OF THIS SECTION: The list of stakeholders will be unique for each organization that is developing a SAD. ANSI/IEEE 1471-2000 requires that at least the following stakeholders be considered:

- Users
- Acquirers
- Developers
- Maintainers.

You may wish to consider the following additional stakeholders.

• Customer • Application software developers • Infrastructure software developers • End users • Application system engineers • Application hardware engineers	• Project manager • Communications engineers • Chief Engineer/Chief Scientist • Program management • System and software integration and test engineers • Safety engineers and certifiers	• External organizations • Operational system managers • Trainers • Maintainers • Auditors • Security engineers and certifiers

1.5 Viewpoint Definitions

CONTENTS OF THIS SECTION: This section provides a short textual definition of a viewpoint and how the concept is used in this SAD. The section describes viewpoints that may be used in the SAD. The specific viewpoints will be tailored by the organization.

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stakeholder’s concerns**
Users	the system is reliable and available when needed
Customer	the architecture can be implemented on schedule and to budget
Project manager	the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways (in addition to cost and schedule)
Developers	strategies to achieve all of those goals
Security analyst	the system will meet its information assurance requirements
Configuration management specialists	maintaining current and past versions of the elements
Engineers	producing a running version of the system
Maintainers	modifying the software elements
Implementers	implementing the elements
Software architects	those software elements sufficiently large or complex enough to warrant their own software architectures

1.5.1 Module Decomposition Viewpoint Definition

There will be one of these subsections for each viewpoint defined. The subsections are as follows:

- Abstract: A brief overview of the viewpoint
- Stakeholders and their concerns addressed: This section describes the stakeholders and their concerns that this viewpoint is intended to address. Listed are questions that can be answered by consulting views that conform to this viewpoint. Optionally, the section includes significant questions that cannot be answered by consulting views conforming to this viewpoint.
- Elements, relations, properties, and constraints: This section defines the types of elements, the relations among them, the significant properties they exhibit, and the constraints they obey for views conforming to this viewpoint.
- Language(s) to model/represent conforming views: This section lists the language or languages that will be used to model or represent views conforming to this viewpoint, and cite a definition document for each.
- Applicable evaluation/analysis techniques and consistency/completeness criteria: This section describes rules for consistency and completeness that apply to views in this viewpoint, as well as any analysis of evaluation techniques that apply to the view that can be used to predict qualities of the system whose architecture is being specified.
- Viewpoint source: This section provides a citation for the source of this viewpoint definition, if any.

Following is an example of a viewpoint definition.

1.5.1 **Module decomposition viewpoint definition**

1.5.1.1 Abstract. Views conforming to the module decomposition viewpoint partition the system into a unique non-overlapping set of hierarchically decomposable implementation units (*modules*).

1.5.1.2 Stakeholders and Their Concerns Addressed. Stakeholders and their concerns addressed by this viewpoint include

- project managers, who must define work assignments, form teams, and formulate project plans and budgets and schedules;
- COTS specialists, who need to have software elements defined as units of functionality, so they can search the marketplace and perform trade studies to find suitable COTS candidates;
- testers and integrators who use the modules as their unit of work;
- configuration management specialists who are in charge of maintaining current and past versions of the elements;
- system build engineers who use the elements to produce a running version of the system;
- maintainers, who are tasked with modifying the software elements;
- implementers, who are required to implement the elements;
- software architects for those software elements sufficiently large or complex enough to warrant their own software architectures;
- the customer, who is concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements.

1.5.1.3 Elements, Relations, Properties, and Constraints. Elements of the module decomposition viewpoint are modules, which are units of implementation that provide defined functionality. Modules are hierarchically decomposable; hence, the relation is “is-part-of.” Properties of elements include their names, the functionality assigned to them (including a statement of the quality attributes associated with that functionality), and their software-to-software interfaces. The module properties may include requirements allocation, supporting requirements traceability.

1.5.1.4 Language(s) to Model/Represent Conforming Views. Views conforming to the module decomposition viewpoint may be represented by (a) plain text using indentation or outline form [Clements 2002]; (b) UML, using subsystems or classes to represent elements and “is part of” or nesting to represent the decomposition relation.

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria.
Completeness/consistency criteria include (a) no element has more than one parent; (b) major functionality is provided for by exactly one element; (c) the union of all elements’ functionality covers the requirements for the system; (d) every piece of source code can be mapped to an element in the module decomposition view (if not, the view is not complete); (e) the selection of module aligns with current and proposed procurement decisions. Additional consistency/completeness criteria apply to the specifications of the elements’ interfaces. Applicable evaluation/analysis techniques include (a) scenario-based evaluation techniques such as ATAM [Clements

2001] to assure that projected changes are supported economically by the decomposition; (b) disciplined and detailed mapping to requirements to assure coverage and non-overlapping functionality; (c) cost-based techniques that determine the number and composition of modules for efficient procurement.

1.5.1.6 Viewpoint Source. [Clements 2002, Section 2.1] describes the module decomposition style, which corresponds in large measure to this viewpoint.

1.5.1.1 Abstract

Views conforming to the module decomposition viewpoint partition the system into a unique non-overlapping set of hierarchically decomposable implementation units (modules).

1.5.1.2 Stakeholders and Their Concerns Addressed

Stakeholders and their concerns addressed by this viewpoint include:

- users, who is concerned that the system is reliable and available when needed;
- customer, who is concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements;
- project manager, who must define work assignments, form teams, and formulate project plans and budgets and schedules;
- developers, who is worried about strategies to achieve all of those goals;
- security analyst, who is concerned that the system will meet its information assurance requirements;
- testers and integrators who use the modules as their unit of work;
- configuration management specialists who are in charge of maintaining current and past versions of the elements;
- system build engineers who use the elements to produce a running version of the system;
- maintainers, who are tasked with modifying the software elements;
- implementers, who are required to implement the elements;
- software architects for those software elements sufficiently large or complex enough to warrant their own software architectures;

1.5.1.3 Elements, Relations, Properties, and Constraints

Elements of the module decomposition viewpoint are modules, which are units of implementation that provide defined functionality. Modules are hierarchically decomposable; hence, the relation is “is-part-of.” The element's relations determine how they are associated with each other. For example, an element may aggregate or be a specialization of another element. Properties of elements include their names, the functionality assigned to them (including a statement of the quality attributes associated with that functionality), and their software-to-software interfaces. The element's properties determine its impact on the quality attributes. For example, an element may have a large negative impact on the performance of the overall system. The element's interface shows its publicly available services using method signatures. The module properties may include requirements allocation, supporting requirements traceability.

1.5.1.4 Language(s) to Model/Represent Conforming Views

Views conforming to the module decomposition viewpoint may be represented by (a) plain text using indentation or outline form [Clements 2002]; (b) UML, using subsystems or classes to represent elements and “is part of” or nesting to represent the decomposition relation.

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Completeness/consistency criteria include (a) no element has more than one parent; (b) major functionality is provided for by exactly one element; (c) the union of all elements’ functionality covers the requirements for the system; (d) every piece of source code can be mapped to an element in the module decomposition view (if not, the view is not complete); (e) the selection of module aligns with current and proposed procurement decisions. Additional consistency/completeness criteria apply to the specifications of the elements’ interfaces. Applicable evaluation/analysis techniques include (a) scenario-based evaluation techniques such as ATAM [Clements 2001] to assure that projected changes are supported economically by the decomposition; (b) disciplined and detailed mapping to requirements to assure coverage and non-overlapping functionality; (c) cost-based techniques that determine the number and composition of modules for efficient procurement.

1.5.1.6 Viewpoint Source

[Clements 2002, Section 2.1] describes the module decomposition style, which corresponds in large measure to this viewpoint.

1.6 How a View is Documented

CONTENTS OF THIS SECTION: This section describes how the documentation for a view is structured and organized. If you change the *organization* of information in Section 3, then you should also change its description in here. Otherwise, this section is all boilerplate.

If you choose to document all information in a view in a single presentation, then you will not need view packets. In that case, the template is as follows:

- Section 3.i: Name of view
- Section 3.i.1: View description
- Section 3.i.2: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
- Section 3.i.3: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of subsections for (respectively) elements, relations, interfaces, behavior, and constraints.
- Section 3.i.4: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the view packet’s scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.
- Section 3.i.5: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.6: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5. Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

- Section 3.i: Name of view.
- Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.
- Section 3.i.2: View packet overview. This section shows the set of view packets in this view, and provides rationale that explains why the chosen set is complete and non-duplicative. The set of view packets may be listed textually, or shown graphically in terms of how they partition the entire architecture being shown in the view.
- Section 3.i.3: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.
- Section 3.i.4: Variability mechanisms. This section describes any architectural variability mechanisms (e.g., adaptation data, compile-time parameters, variable replication, and so forth) described by this view, including a description of how and when those mechanisms may be exercised and any constraints on their use.
- Section 3.i.5: View packets. This section presents all of the view packets given for this view. Each view packet is described using the following outline, where the letter *j* stands for the number of the view packet being described: 1, 2, etc.
  - Section 3.i.5.j: View packet #j.
  - Section 3.i.5.j.1: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based

representation.

- Section 3.i.5.j.2: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:
  - Section 3.i.5.j.2.1: Elements. This section describes each element shown in the primary presentation, details its responsibilities of each element, and specifies values of the elements’ relevant *properties*, which are defined in the viewpoint to which this view conforms.
  - Section 3.i.5.j.2.2: Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.
  - Section 3.i.5.j.2.3: Interfaces. This section specifies the software interfaces to any elements shown in the primary presentation that must be visible to other elements.
  - Section 3.i.5.j.2.4: Behavior. This section specifies any significant behavior of elements or groups of interacting elements shown in the primary presentation.
  - Section 3.i.5.j.2.5: Constraints: This section lists any constraints on elements or relations not otherwise described.
- Section 3.i.5.j.3: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the view packet’s scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.
- Section 3.i.5.j.4: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.5.j.5: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.
- Section 3.i.5.j.6: Relation to other view packets. This section provides references for related view packets, including the parent, children, and siblings of this view packet. Related view packets may be in the same view or in different views.

1.7 Relationship to Other SADs

CONTENTS OF THIS SECTION: This section describes the relationship between this SAD and other architecture documents, both system and software. For example, a large project may choose to have one SAD that defines the system-of-systems architecture, and other SADs to define the architecture of systems or subsystems. An embedded system may well have a *system* architecture document, in which case this section would explain how the information in here traces to information there.

If none, say “Not applicable.”

Not applicable.

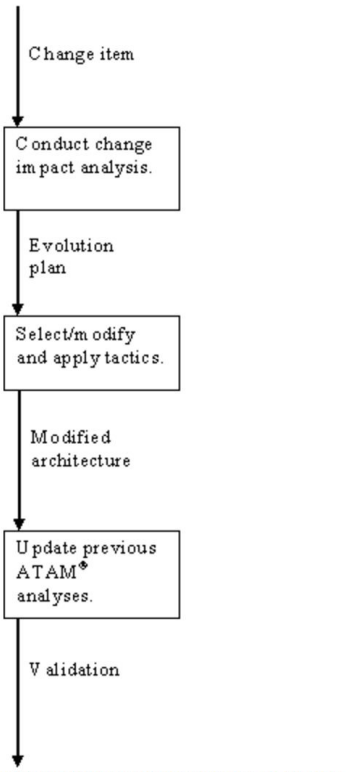
1.8 Process for Updating this SAD

CONTENTS OF THIS SECTION: This section describes the process a reader should follow to report discrepancies, errors, inconsistencies, or omissions from this SAD. The section also includes necessary contact information for submitting the report. If a form is required, either a copy of the blank form that may be photocopied is included, or a

reference to an online version is provided. This section also describes how error reports are handled, and how and when a submitter will be notified of the issue’s disposition.

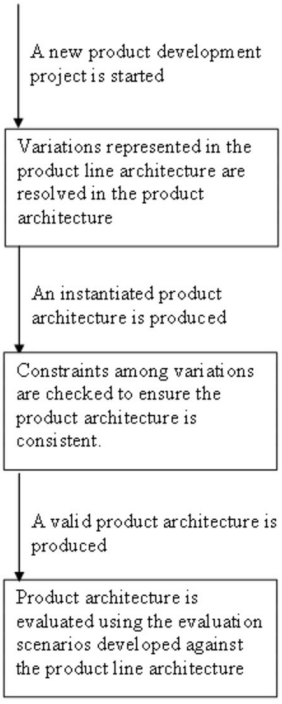
There are several reasons for architecture changes. The following figure describes a process that can be used for any architecture change, but we describe it using a requirements change.

When a requirement is added or modified, the impact of that change to the requirements model is determined. The architecture is reviewed to determine whether the current architecture can satisfy the new requirement. If not, existing items are examined to determine whether they can satisfy the behaviors. If not, new items are designed with responsibility for the new behavior. Any new items are placed in the same operational context as the other items.



Attached Process for Requirements Change

This process becomes a portion of the production plan for a product.



2 Architecture Background

2.1 Problem Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.1 explain the constraints that provided the significant influence over the architecture.

A pacemaker (or artificial pacemaker, so as not to be confused with the heart's natural pacemaker) is a medical device which uses electrical impulses, delivered by electrodes contracting the heart muscles, to regulate the beating of the heart.

The primary purpose of a pacemaker is to maintain an adequate heart rate, either because the heart's natural pacemaker is not fast enough, or because there is a block in the heart's electrical conduction system. Modern pacemakers are externally programmable and allow a cardiologist to select the optimum pacing modes for individual patients. Some combine a pacemaker and defibrillator in a single implantable device. Others have multiple electrodes stimulating differing positions within the heart to improve synchronisation of the lower chambers (ventricles) of the heart.

This PACEMAKER Software Architecture Documentation describes the PACEMAKER-specific programming application and pulse generator (PG). The PACEMAKER system supports the following needs of patients that require bradycardia pacing support: implantation, ambulatory, follow-up and explantation.

2.1.1 System Overview

CONTENTS OF THIS SECTION: This section describes the general function and purpose for the system or subsystem whose architecture is described in this SAD.

The PACEMAKER system provides dual chamber, rate adaptive bradycardia pacing support, provides historical data on device performance, and provides user diagnostics through brady analysis functions.

The bradycardia analysis functions permit the following pacing measurements and tests to be performed: Lead impedance, Pacing threshold, P and R wave measurement, Battery status, Temporary brady pacing and Motion sensor trending.

The PACEMAKER system consists of three major components:

• Device (also called the pulse generator or PG)

• Device Controller-Monitor (DCM) and associated software

• Leads

2.1.2 Goals and Context

CONTENTS OF THIS SECTION: This section describes the goals and major contextual factors for the software architecture. The section includes a description of the role software architecture plays in the life cycle, the relationship to system engineering results and artifacts, and any other relevant factors.

The major goal is to create a pacemaker project that satisfy all of the system requirements.

The PACEMAKER system is indicated for patients exhibiting chronotropic incompetence and who would benefit by increased pacing rates concurrent with physical activity. Generally accepted indications for long-term cardiac pacing include, but are not limited to, sick sinus syndrome; chronic sinus arrhythmias, including sinus bradycardia, sinus arrest, and sinoatrial (S-A) block; second and third-degree AV block; bradycardia-tachycardia syndrome; bundle branch block; and carotid sinus syndrome.

```
stakeholder goals PACEgoals : "Pacemaker stakeholder goals" for system [
    description "This document contains the stakeholder requirements for the Simple Control System (SCS).
PACE provides control for a relatively simple device (SD).
PACE consists of software, hardware, and humans."
    goal g1 : "Reliability" [
        description "The simple controller (SC) shall provide relible control of PACE."
        rationale "PACE is a safety critical device that cannot tolerate unreliable behavior."
        stakeholder pace.jdm
        category Quality.Safety
    uncertainty[
        volatility 2
        impact 3

    ]

    ]

    ]
```

2.1.3 Significant Driving Requirements

CONTENTS OF THIS SECTION: This section describes behavioral and quality attribute requirements (original or derived)

that shaped the software architecture. Included are any scenarios that express driving behavioral and quality attribute goals, such as those crafted during a Quality Attribute Workshop (QAW) [Barbacci 2003] or software architecture evaluation using the Architecture Tradeoff Analysis Method[3]SM (ATAM[SM]) [Bass 2003].

A functional requirement is a statement of something the system must be able to do to be considered “complete.” A module implements a functional requirement. The highest vote getters are the driving requirements. Often these are the only ones we will have time to consider as we make design decisions.

For the Device Controller-Monitor (DCM) only those items that will be supported as part of the application software development for the PACEMAKER will be defined. The PACEMAKER model type shall support single and dual chamber rate adaptive pacing.

Model	Pacemaker Designation	Functionality	Connector
DR1	DR	DDDR full function with accelerometer	Dual-in-line, 3.2 mm (IS-1)

Table 1: Model Type and Lead Port

Customer Scenarios:

Scenario Refinement for Scenario #1		
Scenario(s):		When a Pulse Generator senses an abnormal decrease in the heart rate, it starts generating pulses in less than one millisecond.
Business Goals:		
Relevant Quality Attributes:		Reliability, Functionality (or Performance)
Scenario Components	Stimulus:	The heart rate of a patient decreased abnormally.
	Stimulus Source:	The patient
	Environment:	The Pulse Generator is in the process of working.
	Artifact (If Known):	Leads, Pulse Generator, Device Controller-Monitor
	Response:	The Pulse Generator starts generating pulses.
	Response Measure:	one millisecond
Questions:		What’s the lowest heart rate of a patient before it is detected by the system’s sensor?
Issues:		May need to train installers to avoid parameter setting errors and prevent malfunctions.

[3]SM Quality Attribute Workshop and QAW and Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

Scenario Refinement for Scenario #2		
Scenario(s):		When the battery of a Pulse Generator is about to run out, it switches working modes to extend lifespan in less than one second.
Business Goals:		
Relevant Quality Attributes:		Usability, Cost-saving
Scenario Components	Stimulus:	The battery of a Pulse Generator is below the critical level.
	Stimulus Source:	the Pulse Generator
	Environment:	The Device Controller is in the process of monitoring and the Pulse Generator is working.
	Artifact (If Known):	Leads, Pulse Generator, Device Controller-Monitor
	Response:	The Pulse Generator switches its working mode into a low-power status.
	Response Measure:	One second
Questions:		What’s the lowest battery could a Pulse Generator be before it switches working modes to maintain essential functional requirements?
Issues:		Patients may need to detect the operating states of battery and relevant components in hospital termly.

Developer Scenarios:

Scenario Refinement for Scenario #3		
Scenario(s):		When a developer adds the support of DDDR model to the system, it should be accurate and doesn’t influence other functions.
Business Goals:		
Relevant Quality Attributes:		Maintainability, Modifiability
Scenario Components	Stimulus:	add the support of DDDR model to the system
	Stimulus Source:	developer
	Environment:	design-time, develop time
	Artifact (If Known):	source code
	Response:	New model will be accurate, and don’t influence other functions.
	Response Measure:	finish in one month
Questions:		How developers can familiar with the code quickly?
Issues:		May need perfect and effective mechanism and accurate and complete documents.

Scenario Refinement for Scenario #4		
Scenario(s):		When finishing the code of rate sensing function, The pacemaker experiences a thorough unit test.
Business Goals:		
Relevant Quality Attributes:		
Scenario Components	Stimulus:	When finishing the function of rate sensing
	Stimulus Source:	developer
	Environment:	design-time, develop-time
	Artifact (If Known):	source code
	Response:	Rate sensing passes the unit test
	Response Measure:	every 2 seconds
Questions:		What is the correct value of heart rate?
Issues:		May need a simulator to simulate human’s heartbeat

Engineer Scenario:

Scenario Refinement for Scenario #5		
Scenario(s):		When the sensors sense arrhythmia, the pacemaker will record relevant data and prepared for doctor to check.
Business Goals:		Practical product; feature-rich product
Relevant Quality Attributes:		performance
Scenario Components	Stimulus:	Arrhythmia of the host
	Stimulus Source:	Pulses external to the system, irregular heartbeat
	Environment:	The pacemaker is in the standby mode.
	Artifact (If Known):	System’s impulse sensor, info-logout software component
	Response:	The pacemaker records the arrhythmia information
	Response Measure:	Data in the flash
Questions:		How to distinguish arrhythmia with normal heartbeat rate changing?
Issues:		May need further information to adjust or use machine learning

Scenario Refinement for Scenario #6		
Scenario(s):		When the pacemaker itself meets system error, the pacemaker records and informs doctor about the risk
Business Goals:		Safest system, feature-rich product
Relevant Quality Attributes:		Safety, reliability
Scenario Components	Stimulus:	System errors
	Stimulus Source:	Internal errors such as software error; external errors such as not recorded pulses.
	Environment:	Anytime the pacemaker is at working
	Artifact (If Known):	System’s impulse sensor, system’s component checking software component
	Response:	The pacemaker record and inform the doctor about the risk
	Response Measure:	Data in the flash & data uploaded to the doctor
Questions:		How to determine whether errors come from external factors or internal factors?
Issues:		May need extra sensors for error detection.

Product Manager Scenarios:

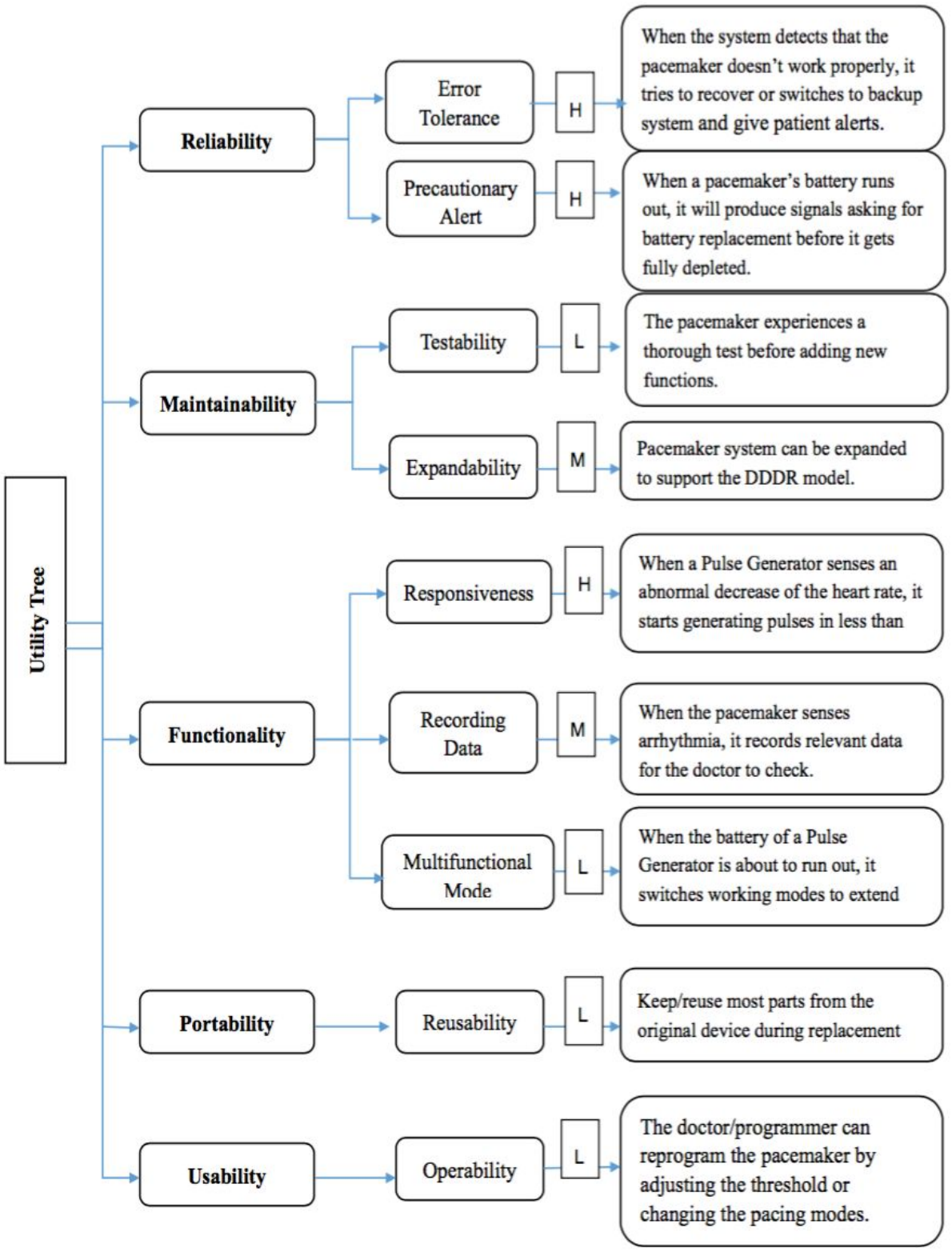
Scenario Refinement for Scenario #7		
Scenario(s):		When the system detects that the pacemaker doesn’t work properly, it will try to recover or switch to backup system and give patient alerts.
Business Goals:		Safest device, feature-rich
Relevant Quality Attributes:		Safety, availability
Scenario Components	Stimulus:	Malfunctions, including low-battery, problems with detecting intra cardiac signals
	Stimulus Source:	Main functional parts of the pacemaker
	Environment:	The error handler is monitoring the status of other parts
	Artifact (If Known):	Error handler
	Response:	Try to recover or switch to backup system and give alerts
	Response Measure:	Less than one second
Questions:		How long at least should the backup system work properly before doctors fix the problem?
Issues:		Need to tell patients avoid potential dangerous situation and teach them how to recognize the meaning of different notifications.

Scenario Refinement for Scenario #8		
Scenario(s):		When you want to add some functions, or replace some units (like battery) of the device, you can do it easily and keep/reuse most parts from the original device.
Business Goals:		Resource economy, maintainability
Relevant Quality Attributes:		Reusability, portability
Scenario Components	Stimulus:	When you want to add some functions, or replace some units.
	Stimulus Source:	New feature requirement or equipment maintenance
	Environment:	Normal operation
	Artifact (If Known):	Technicians, human operator
	Response:	Make modification without side effect, deploy it with minimal effort
	Response Measure:	Extent of effort/cost
Questions:		What’s the specific parameter requirement for different units of the device?
Issues:		The technician must the life cycle of each unit of the device.

Maintainer Scenarios:

Scenario Refinement for Scenario #9		
Scenario(s):		When a patient experiences irregular beating from the pacemaker, the doctor/programmer can reprogram the pacemaker by adjusting the threshold or changing the pacing modes.
Business Goals:		feature-rich product, smart system
Relevant Quality Attributes:		Maintainability
Scenario Components	Stimulus:	Patients experience irregular beating from the pacemaker.
	Stimulus Source:	Problems with sensing including undersensing or oversensing; problems with output including output failure or failure to capture.
	Environment:	Pacemaker is implanted in the patient’s body
	Artifact (If Known):	
	Response:	The pacemaker has a reprogramming functionality allows
	Response Measure:	As needed
Questions:		How is the pacing modes related to the irregular beating?
Issues:		The doctor meets with patients every 6 months for regular check and responses to patients immediately as requested

Scenario Refinement for Scenario #10		
Scenario(s):		When a pacemaker’s battery runs out, it will produce signals asking for battery replacement before it gets fully depleted.
Business Goals:		Safest system, smart system
Relevant Quality Attributes:		Maintainability, Reliability
Scenario Components	Stimulus:	A pacemaker’s battery becomes depleted
	Stimulus Source:	Pacemaker’s battery
	Environment:	The pacemaker is implanted in the patient’s body
	Artifact (If Known):	
	Response:	The pacemaker sends out signals for replacement
	Response Measure:	As soon as the battery life is below the warning threshold
Questions:		How long should the battery keep working after sending out signals?
Issues:		The battery life should be checked during the regular follow-up care.


Utility Tree
Reliability
Error Tolerance
H
When the system detects that the pacemaker doesn't work properly, it tries to recover or switches to backup system and give patient alerts.
Precautionary Alert
H
When a pacemaker's battery runs out, it will produce signals asking for battery replacement before it gets fully depleted.
Maintainability
Testability
L
The pacemaker experiences a thorough test before adding new functions.
Expandability
M
Pacemaker system can be expanded to support the DDDR model.
Functionality
Responsiveness
H
When a Pulse Generator senses an abnormal decrease of the heart rate, it starts generating pulses in less than
Recording Data
M
When the pacemaker senses arrhythmia, it records relevant data for the doctor to check.
Multifunctional Mode
L
When the battery of a Pulse Generator is about to run out, it switches working modes to extend
Portability
Reusability
L
Keep/reuse most parts from the original device during replacement
Usability
Operability
L
The doctor/programmer can reprogram the pacemaker by adjusting the threshold or changing the pacing modes.

2.2 Solution Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.2 provide a description of why the architecture is the way that it is, and a convincing argument that the architecture is the right one to satisfy the behavioral and quality attribute goals levied upon it.

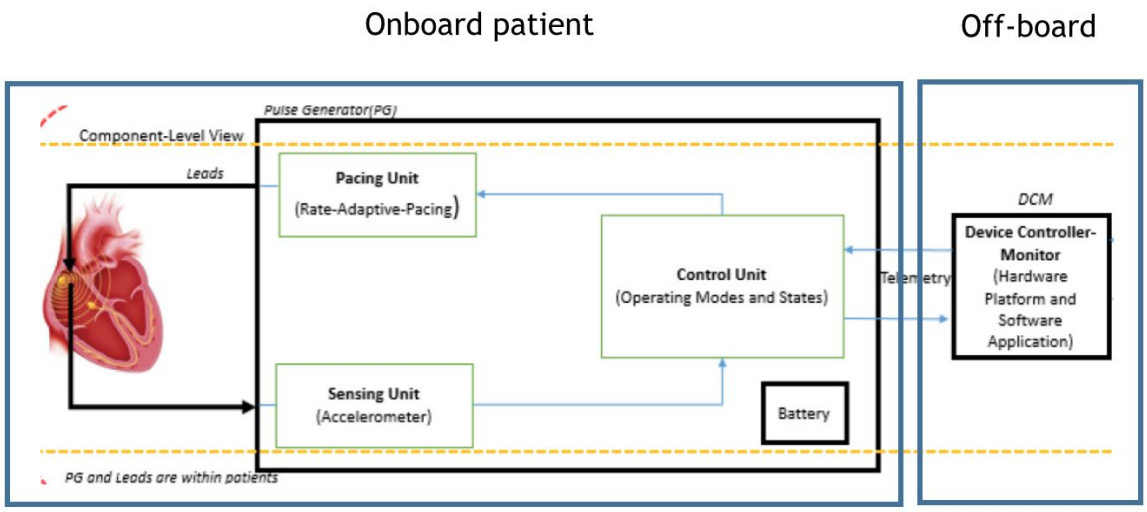
In original design of pacemaker-EASY, the system architecture is not complete. Thus, I used Feedback Control Loop which includes nominal and error models, and three design tactics which include Splitting, Intermediary and Augmenting to refine my pacemaker design. In this way, the performance and reliability of Pacemaker system will be improved efficiently.

2.2.1 Architectural Approaches

CONTENTS OF THIS SECTION: This section provides a rationale for the major design decisions embodied by the software architecture. It describes any design approaches applied to the software architecture, including the use of architectural styles or design patterns, when the scope of those approaches transcends any single architectural view. The section also provides a rationale for the selection of those approaches. It also describes any significant alternatives that were seriously considered and why they were ultimately rejected. The section describes any relevant COTS issues, including any associated trade studies.

Tactic 1: Splitting

The pacemaker system is decomposed into 2 subsystems: Pulse Generator system and Device Controller Monitor system. Distributed architecture techniques require synchronization. This is a closed system so no need to translate between systems.



Besides, the Accelerometer module in previous Pulse Generator system is divided into four modules, including Controller, Accelerometer, Error Handler, and Pacing Unit. Each module is responsible for respective functions after modification. In this way, the error tolerance□ of reliability, software independence of portability and performance of system have been improved dramatically. There is also a feedback control loop forming inside the Pulse Generator.

Tactic 2: Intermediary

The process Pacemaker_TR is added as an intermediary between process Pacemaker_SW and general operating modes for better expansibility and performance. Control Mode is separated from general operating modes, because it determines switching of modes and judging of errors.

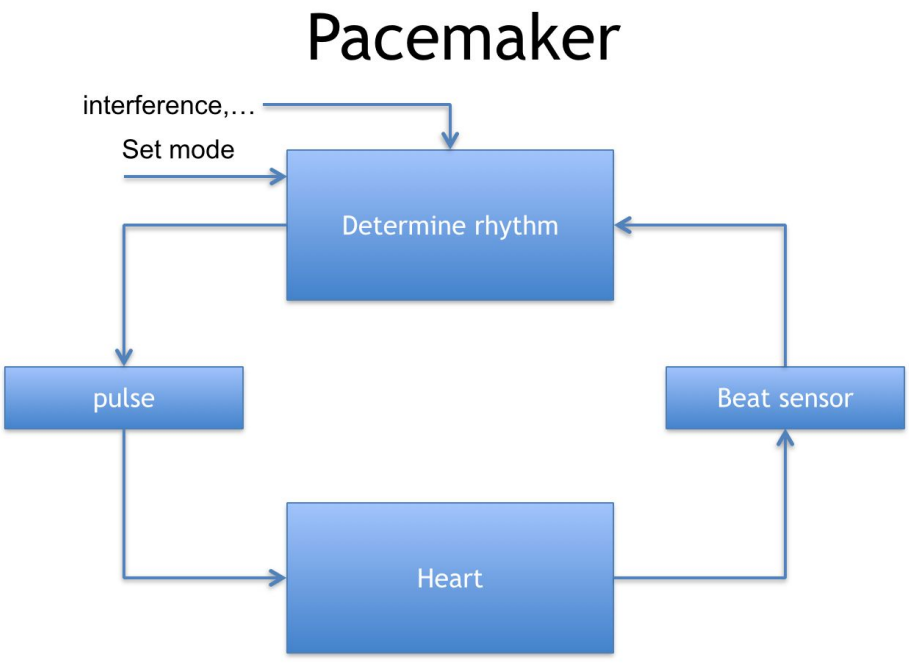
I considered tactic Encapsulation as an alternative of Intermediary, but I didn’t select it. A major reason is that an efficient interface of process Pacemaker_SW is needed in tactic Encapsulation, but I didn’t know how to create an interface between process and threads which include DDD Mode, AAI Mode and any other operating modes. So I chose tactic Intermediary to create another process Pacemaker_TR for transition.

Tactic 3: Augmenting

Two additional threads and three additional processes are added to the system in order to implement a feedback control loop including nominal and error flows. Also, a pacemaker_HR _Platform is added to the Device Controller Monitor system because of the new Pacemaker_TR process.

Feedback Control Loop

Instead of the previous design, a feedback control loop is added into the Pulse Generator system, including nominal and error model and identifying the controller, sensors, actuators, controlled process. Control unit is a finite state machine that controls timing and strength of pulse. Actuators/Pacing Unit uses leads to actuate the heart when needed. In Sensing Unit, the same leads are used to administer shock. In this way, no matter how quick the heart rate changes, we can sense these current values and determine if it is necessary to generate a pulse or change the operating modes of pacemaker. The frequency and magnitude of Pulse are chosen by controller. If sampling misses a beat, may pulse too soon. If sampling reads in between beats, the software may deduce the wrong magnitude.

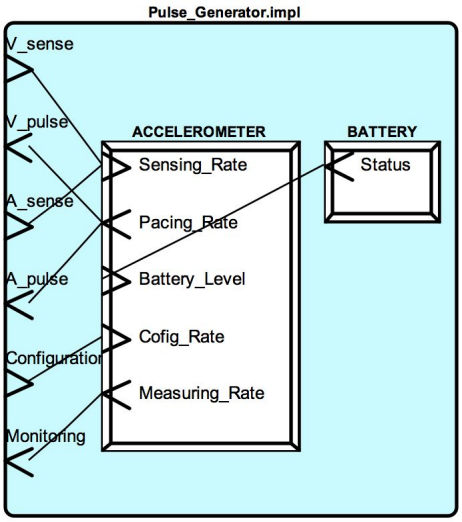


2.2.2 Analysis Results

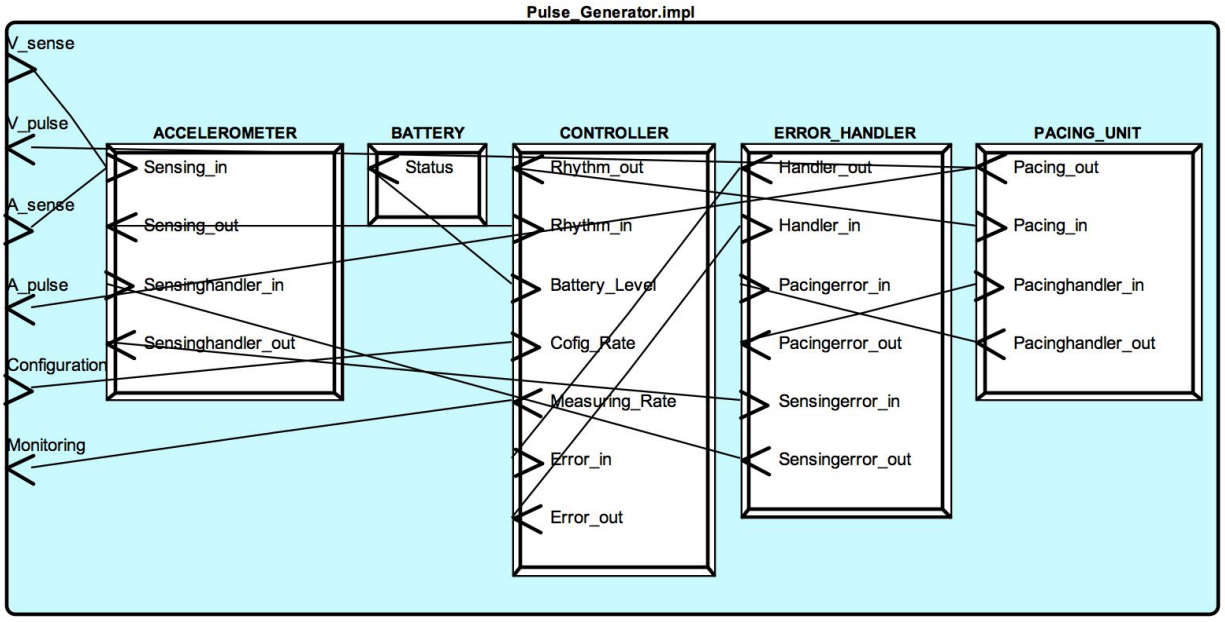
CONTENTS OF THIS SECTION: This section describes the results of any quantitative or qualitative analyses that have been performed that provide evidence that the software architecture is fit for purpose. If an Architecture Tradeoff Analysis Method evaluation has been performed, it is included in the analysis sections of its final report. This section refers to the results of any other relevant trade studies, quantitative modeling, or other analysis results.

The Accelerometer module in Pulse Generator system is divided into four modules, including Accelerometer, Controller, Error Handler, and Pacing Unit. Each module is responsible for respective functions after modification.

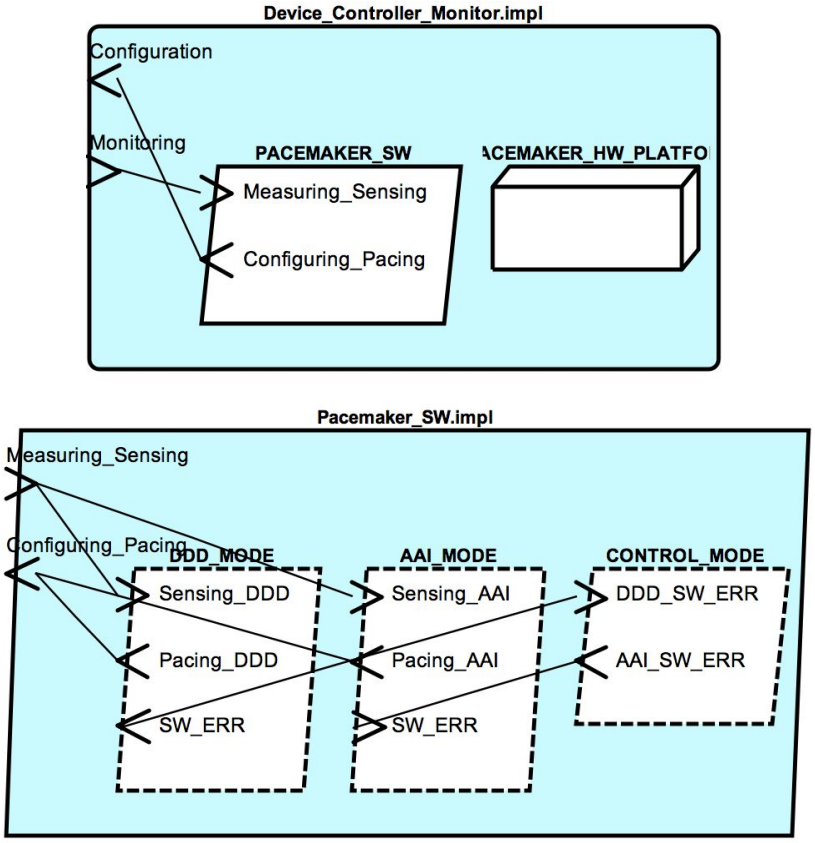
Before modification：



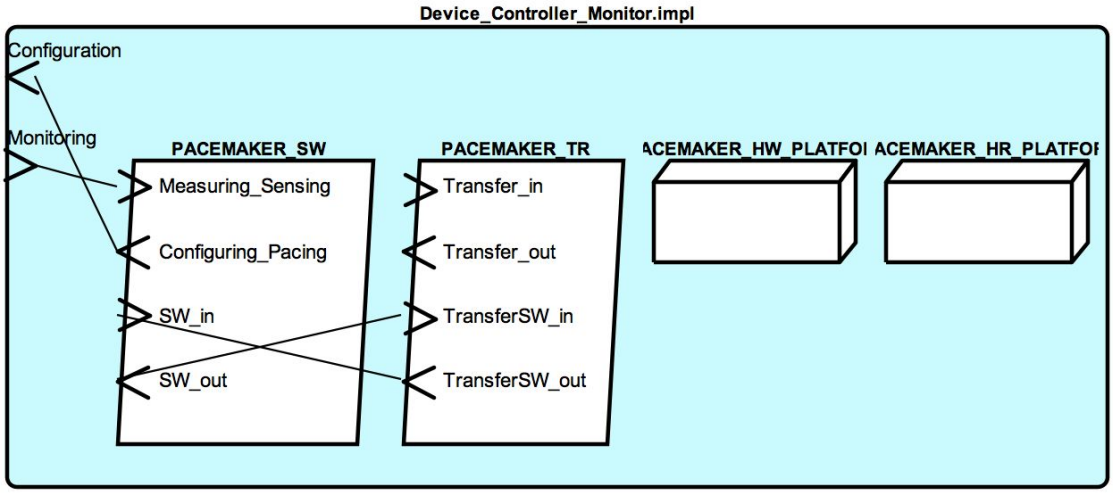
After modification：

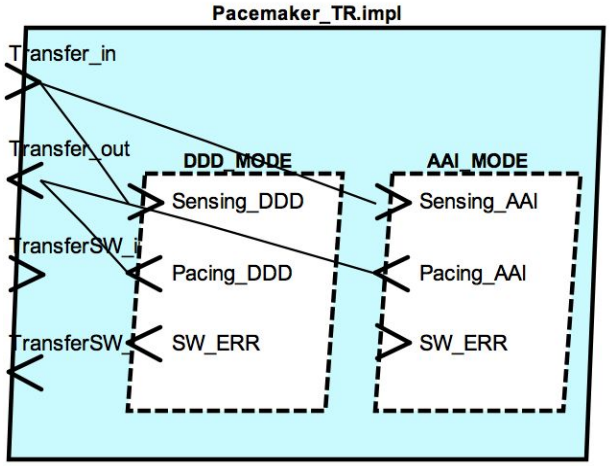


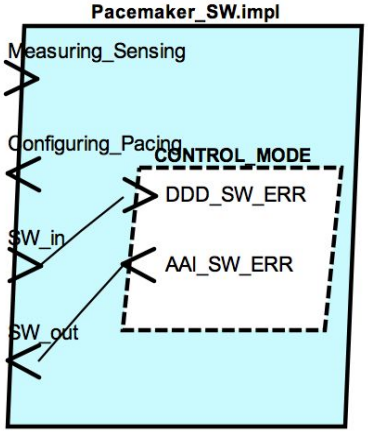
In this way, the testability of maintainability, hardware independence□ of portability and efficiency have been improved in the pacemaker system.
The process Pacemaker_TR is added as an intermediary between process Pacemaker_SW and general operating modes for better expansibility and performance.
Before modification：



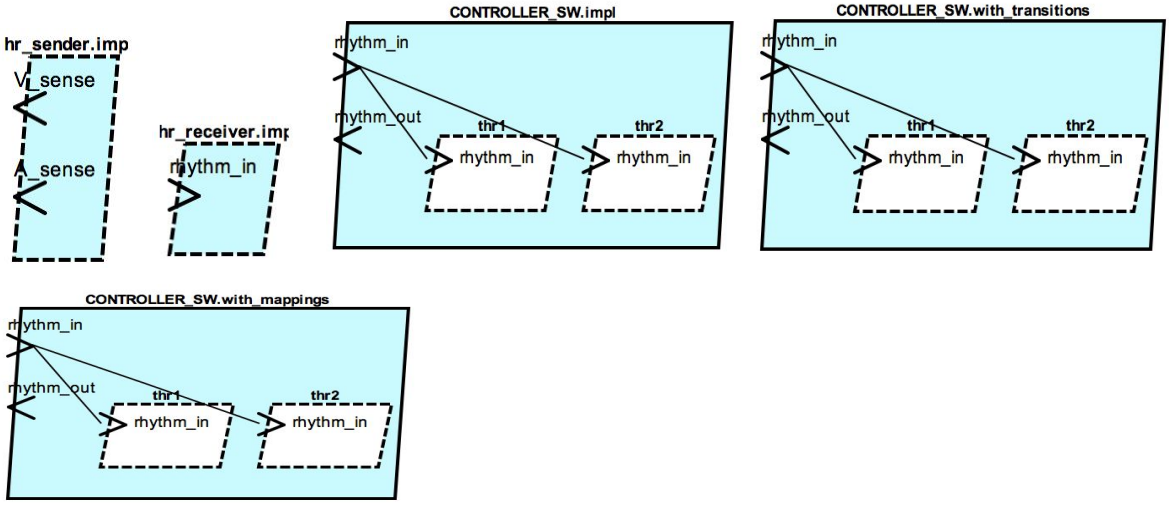
After modification：



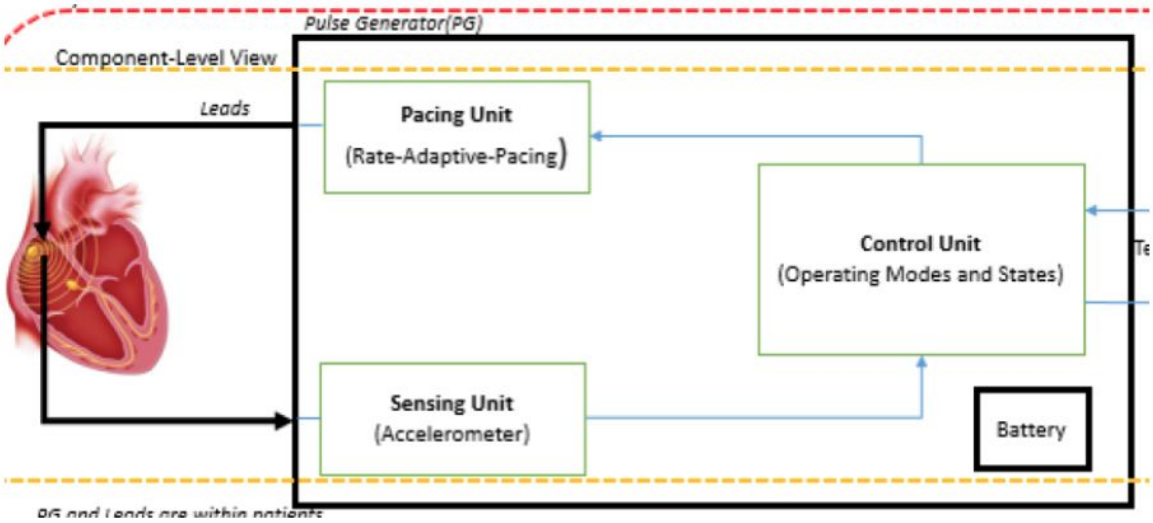




In this way, not only the expandability and testability of maintainability, but also the interoperability of functionality have been improved in the Device Controller Monitor system.
Two additional threads and three additional processes are added to the system in order to implement a feedback control loop including nominal and error flows.



In this way, the completeness, correctness and compatibility of functionality in the entire pacemaker system have been improved.



2.2.3 Requirements Coverage

CONTENTS OF THIS SECTION: This section describes the requirements (original or derived) addressed by the software architecture, with a short statement about where in the architecture each requirement is addressed.

The requirements (original or derived) addressed by the software architecture have been discussed in 2.2.2 Analysis Results. The revised design satisfy most of the Significant Driving Requirements.

2.2.4 Summary of Background Changes Reflected in Current Version

CONTENTS OF THIS SECTION: For versions of the SAD after the original release, this section summarizes the actions, decisions, decision drivers, analysis and trade study results that became decision drivers, requirements changes that became decision drivers, and how these decisions have caused the architecture to evolve or change.

All of these decisions made in revised version have caused the architecture to be evolved. And The revised design satisfy most of the Significant Driving Requirements.

2.3 Product Line Reuse Considerations

CONTENTS OF THIS SECTION: When a software product line is being developed, this section details how the software covered by this SAD is planned or expected to be reused in order to support the product line vision. In particular, this section includes a complete list of the variations that are planned to be produced and supported. "Variation" refers to a variant of the software produced through the use of pre-planned variation mechanisms made available in the software architecture. It may refer to a variant of one of the modules identified in this SAD, or a collection of modules, or the entire system or subsystem covered by this SAD. For each variation, the section identifies the increment(s) of the software build in which (a) the variation will be available; and (b) the variation will be used. Finally, this section describes any additional potential that exists to reuse one or more of the modules or their identified variations, even if this reuse is not currently planned for any increment.

3 Views

CONTENTS OF THIS SECTION: The sub-parts of Section 3 specify the views corresponding to the viewpoints listed in Section 1.5.

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system’s structure change as it executes?
- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)
- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

The views presented in this SAD are the following:

Name of view	Viewtype that defines this view	Types of elements and relations shown	Is this a module view?	Is this a component-and-connector view?	Is this an allocation view?
Allocation Deployment View	Allocation Deployment	dependency	No	No	Yes
Module Decomposition View	Module Decomposition	generalizationspecialization	Yes	No	No
Component-and-Connector View	Component-and-Connector	method invocation	No	Yes	No

3.1 Allocation Deployment View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.1.1 View Description

As shown in the following figure, we intend to deploy all products in the Pacemaker product line on a single processor. Each model in the product line will have the same relationships with the available driver packages that make up the external environment. Therefore, this document focuses only on Pacemaker's software architecture.

3.1.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

<<View packet # 1 : Pacemaker>>

3.1.3 Architecture Background

In original design of pacemaker-EASY, the system architecture is not complete. Thus, I used Feedback Control Loop which includes nominal and error models, and three design tactics which include Splitting, Intermediary and Augmenting to refine my pacemaker design. In this way, the performance and reliability of Pacemaker system will be improved efficiently.

3.1.4 Variability Mechanisms

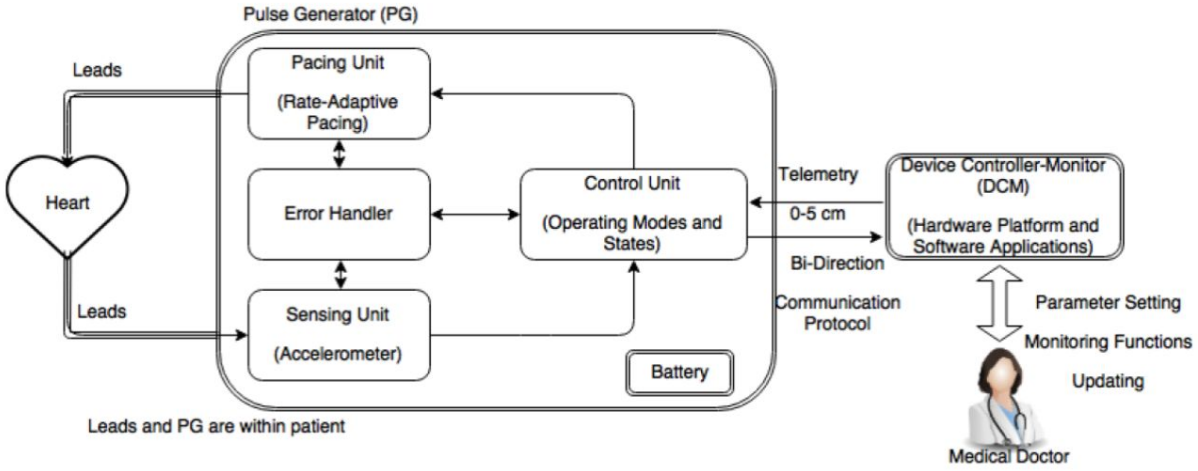
The Variability has been improved through splitting the entire system into several cooperating modules which also take the responsibility of their respective functionality.

3.1.5 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

3.1.5.1 View packet # 1 : Pacemaker

3.1.5.1.1 Primary Presentation



3.1.5.1.2 Element Catalog

3.1.5.1.2.1 Elements

Elements and Responsibilities for Allocation Deployment View Packet 1: Pacemaker	
Element	Responsibilities
Device (also called the pulse generator or PG)	sensing heart rate and generating pulses
Device Controller-Monitor (DCM) and associated software	monitoring functions and PG status
Leads	linking heart and PG within patient

3.1.5.1.2.2 Relations

The relationship in this view is dependency. The Pulse Generator has dependencies on three devices in the entire system that are realized by calls on library methods.

3.1.5.1.2.3 Interfaces

Pacemaker's interface is a GUI that provides the basic start/exit/pause behavior.

3.1.5.1.2.4 Behavior

Pacemaker is the top level of each element. The general behavior is an action loop that runs until the patients take off the Pacemaker. Pulse Generator-specific behavior is provided by specialized Pacemaker class.

3.1.5.1.2.5 Constraints

No other information applies.

3.1.5.1.3 Context Diagram

This is the top level of the product. There is no context other than an executable running on the supported platform.

3.1.5.1.4 Variability Mechanisms

Pacemaker is the generic product.

3.1.5.1.5 Architecture Background

Pacemaker provides the basic behavior for any arcade devices. The specializations add details to provide the specifics of a device.

3.1.5.1.6 Related View Packets

Pacemaker internals are provided in the Module Decomposition View Packet 1: Pulse Generator section.

Specializations of the generalization are provided in the Module Generalization View Packet 1: Pulse Generator section.

3.2 Module Decomposition View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.2.1 View Description

In this section, we describe the basic structure of a Pacemaker. Pacemaker variations are addressed by substitution, parameterization, and specialization. Pacemaker-specific

behaviors are provided by Pacemaker-specific implementations of the interfaces in this document.

3.2.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

<<View packet # 1: Pulse Generator, View packet # 2: Device Controller-Monitor (DCM)>>

3.2.3 Architecture Background

In original design of pacemaker-EASY, the system architecture is not complete. Thus, I used Feedback Control Loop which includes nominal and error models, and three design tactics which include Splitting, Intermediary and Augmenting to refine my pacemaker design. In this way, the performance and reliability of Pacemaker system will be improved efficiently.

3.2.4 Variability Mechanisms

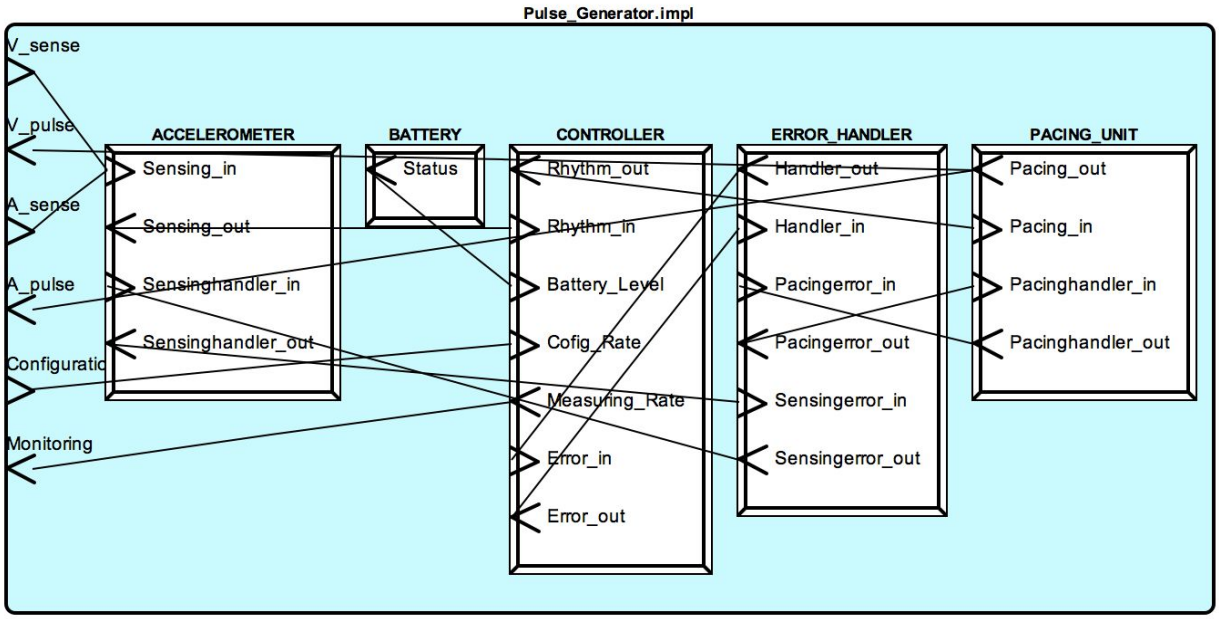
The Variability has been improved through splitting the entire system into several cooperating modules which also take the responsibility of their respective functionality.

3.2.5 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

3.2.5.1 View packet # 1: Pulse Generator

3.2.5.1.1 Primary Presentation



The previous figure shows the Pulse Generator component as the representation for the generic product. The following figure shows that component's interface as the top-level interface of the system and the other major interfaces that are at the first level of decomposition.

3.2.5.1.2 Element Catalog

3.2.5.1.2.1 Elements

Elements and Responsibilities for Module Decomposition View Packet 1: Pulse Generator	
Element	Responsibilities
Accelerometer	sensing heart rate
Controller	controlling PG operating modes and processing received data
Error Handler	handling errors
Pacing Unit	generating pulses

Battery	providing energy

3.2.5.1.2.2 Relations

The primary relation is composition. Pulse Generator is responsible for sensing heart rate, handling errors and generating pulses.

3.2.5.1.2.3 Interfaces

Pulse Generator interface is the program's GUI. It provides the user with sensing heart rate, handling errors, generating pulses and save behaviors. The other interfaces are documented as follows: Accelerometer, Controller, Error Handler, Pacing Unit and Battery.

3.2.5.1.2.4 Behavior

The behaviors of Pulse Generator is sensing heart rate, handling errors and generating pulses.

3.2.5.1.2.5 Constraints

No other information applies.

3.2.5.1.3 Context Diagram

Pulse Generator is the top level of the product and the top-level context.

3.2.5.1.4 Variability Mechanisms

Pulse Generator will be replaced by a Pulse Generator-specific implementation as described in the Module Generalization View section.

3.2.5.1.5 Architecture Background

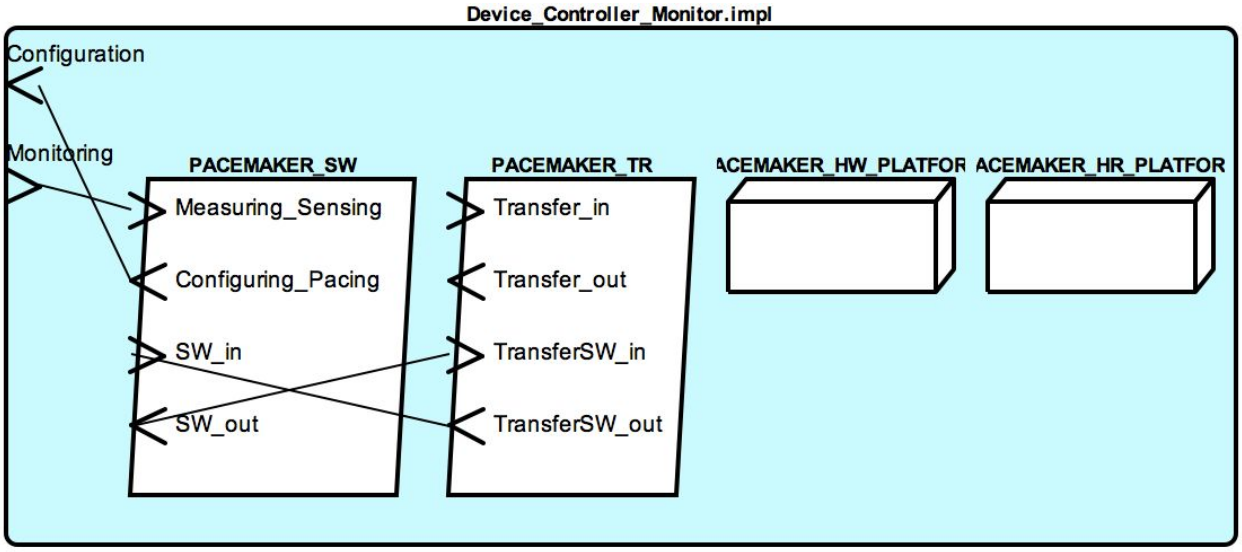
Pulse Generator encapsulates Pulse Generator-specific behavior. It arranges the Accelerometer, Controller, Error Handler, Pacing Unit and Battery based on its rules. The composed elements are mostly generic and can be reused and rearranged to implement other devices.

3.2.5.1.6 Related View Packets

Module Generalization View Packet 1: Pacemaker

3.2.5.2 View packet # 2: Device Controller-Monitor (DCM)

3.2.5.2.1 Primary Presentation



3.2.5.2.2 Element Catalog

3.2.5.2.2.1 Elements

Elements and Responsibilities for Module Decomposition View Packet 2: DCM	
Element	Responsibilities
Process	selecting operating modes of Pacemaker
Platform	providing running processors

3.2.5.2.2.2 Relations

The primary relation is composition. DCM composes the elements that make the selecting operating modes of Pacemaker behavior.

3.2.5.2.2.3 Interfaces

Pacemaker_SW, Pacemaker_TR

3.2.5.2.2.4 Behavior

The behaviors of DCM is selecting operating modes of Pacemaker, parameter setting and monitoring functions.

3.2.5.2.2.5 Constraints

No other information applies.

3.2.5.2.3 Context Diagram

Device Controller-Monitor (DCM) is the top level of the product and the top-level context.

3.2.5.2.4 Variability Mechanisms

DCM is a container. Pacemaker adds the desired operating modes into the DCM.

3.2.5.2.5 Architecture Background

DCM is designed as a component container. It delivers events to the components it holds. For example, it delivers timer ticks to its processes.

3.2.5.2.6 Related View Packets

Allocation Deployment View packet # 1 : Pacemaker

3.3 Component-and-Connector View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.3.1 View Description

In this section, allocation and module views are supplemented with an operational view (see next figure) using a component-and-connector viewtype. This view provides a look at the computation flow through the system.

3.3.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

<<View packet # 1: Feedback Control Loop>>

3.3.3 Architecture Background

In original design of pacemaker-EASY, the system architecture is not complete. Thus, I used Feedback Control Loop which includes nominal and error models, and three design tactics which include Splitting, Intermediary and Augmenting to refine my pacemaker design. In this way, the performance and reliability of Pacemaker system will be improved efficiently.

3.3.4 Variability Mechanisms

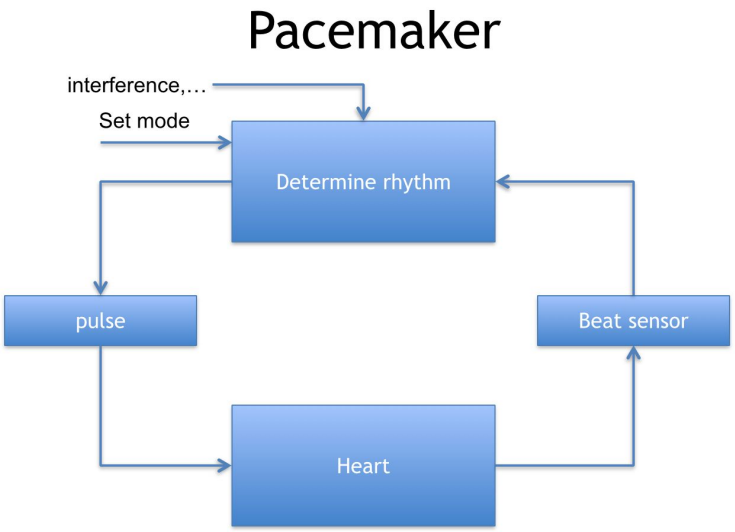
The Variability has been improved through splitting the entire system into several

cooperating modules which also take the responsibility of their respective functionality.

3.3.5 View Packets

3.3.5.1 View packet # 1: Feedback Control Loop

3.3.5.1.1 Primary Presentation



3.3.5.1.2 Element Catalog

3.3.5.1.2.1 Elements

Elements and Responsibilities for Component-and-Connector View Packet 1: Feedback Control Loop	
Element	Responsibilities
Heart	linking to PG with leads
Controller	controlling PG operating modes and processing received data
Sensor	sensing heart rate
Pacing Unit	generating pulses

3.3.5.1.2.2 Relations

The relation shown in the above figure is method invocation. Each arrow represents a synchronous or asynchronous invocation of a method on the called object.

3.3.5.1.2.3 Interfaces

Pulse Generator interface, and DCM interface.

3.3.5.1.2.4 Behavior

No behavior applies.

3.3.5.1.2.5 Constraints

No other information applies.

3.3.5.1.3 Context Diagram

This is the top-level system as shown in the Process Allocation for Pacemaker figure.

3.3.5.1.4 Variability Mechanisms

The variation in this scenario comes when a Pacemaker-specific variant is substituted. This diagram does not change, but the conditions under pacing are sent to change.

3.3.5.1.5 Architecture Background

In this way, no matter how quick the heart rate changes, we can sense these current values and determine if it is necessary to generate a pulse or change the operating modes of pacemaker. The frequency and magnitude of Pulse are chosen by controller. If sampling misses a beat, may pulse too soon. If sampling reads in between beats, the software may deduce the wrong magnitude.

3.3.5.1.6 Related View Packets

Allocation Deployment View packet # 1 : Pacemaker

Module Decomposition View packet # 1: Pulse Generator

Module Decomposition View packet # 2: Device Controller-Monitor (DCM)

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any

known inconsistencies among the views.

4.1 General Relations Among Views

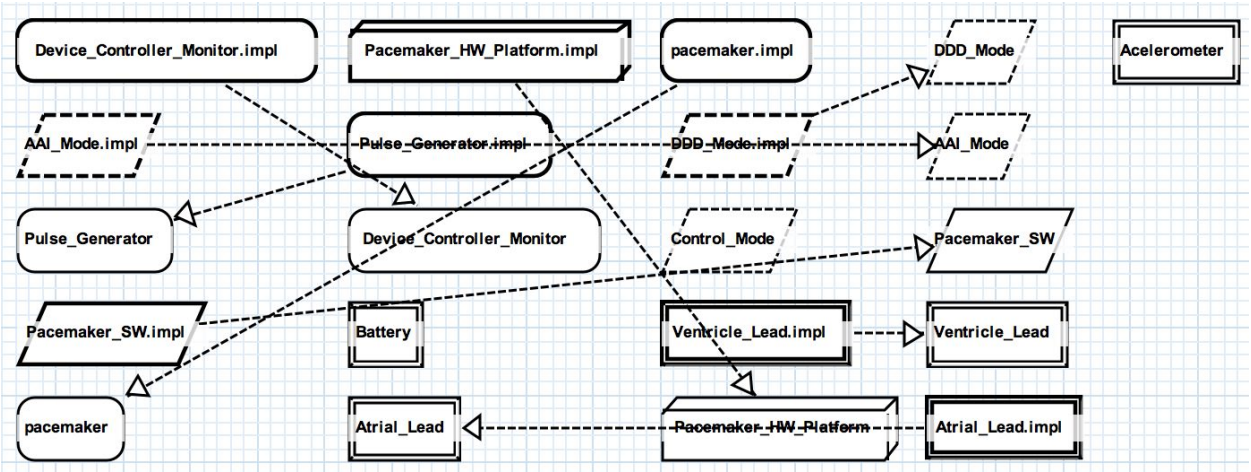
CONTENTS OF THIS SECTION: This section describes the general relationship among the views chosen to represent the architecture. Also in this section, consistency among those views is discussed and any known inconsistencies are identified.

These views are related to each other. Applies a viewpoint to the architecture for a specific stakeholder. Every stakeholder should have at least one view that speaks to their concerns. The view should put the information into context but not confuse the reader with extraneous material.

4.2 View-to-View Relations

CONTENTS OF THIS SECTION: For each set of views related to each other, this section shows how the elements in one view are related to elements in another.

The Pacemaker architecture can show how the elements in one view are related to elements in another.



5 Referenced Materials

CONTENTS OF THIS SECTION: This section provides citations for each reference document. Provide enough information so that a reader of the SAD can be reasonably expected to locate the document.

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*,

	Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice,* second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.

6 Directory

6.1 Index

CONTENTS OF THIS SECTION: This section provides an index of all element names, relation names, and property names. For each entry, the following are identified:

- the location in the SAD where it was defined
- each place it was used

Ideally, each entry will be a hyperlink so a reader can instantly navigate to the indicated location.

Element:

Accelerometer Element Catalog
Controller Element Catalog
Error Handler Element Catalog
Pacing Unit Element Catalog
Battery Element Catalog
Process Element Catalog
Platform Element Catalog
Pulse Generator Element Catalog
DCM Element Catalog
Leads Element Catalog

6.2 Glossary

CONTENTS OF THIS SECTION: This section provides a list of definitions of special terms and acronyms used in the SAD. If terms are used in the SAD that are also used in a parent SAD and the definition is different, this section explains why.

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible” properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or

	template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

6.3 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language

7 Sample Figures & Tables

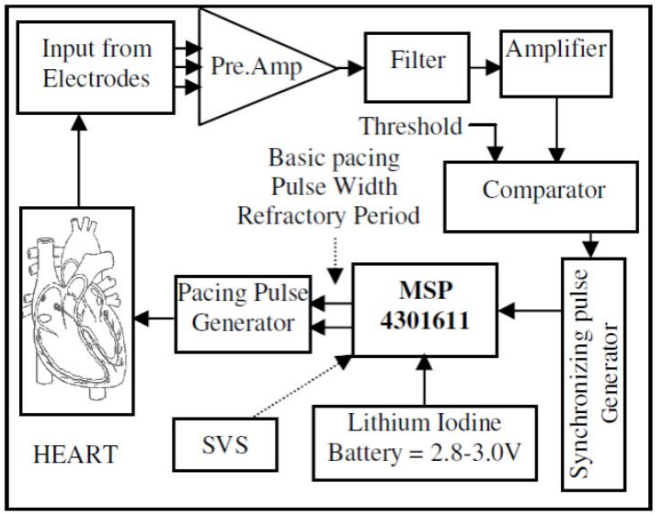


Figure 1: Sample Figure

Table 2: Sample Table

Table Heading	Table Heading	Table Heading	Table Heading
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body

Appendix A

Appendices

CONTENTS OF THIS SECTION: Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data, API specification). As applicable, each appendix is referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. If your SAD has no appendices, delete this page.

A.1 Heading 2 - Appendix

A.2 Heading 2 - Appendix