Software Engineering Institute
Carnegie Mellon University

THE ARCADE GAME MAKER PEDAGOGICAL PRODUCT LINE

Software Engineering Institute

September 2009

Introduction

The Arcade Game Maker (AGM) product line is an example product line created to support learning about and experimenting with software product lines. The product line encompasses three simple arcade games. The primary goal has been to be comprehensive rather than focus on high-quality graphics. The material available follows the basic product line approach described in [Clements 02].

This example was incrementally created over 2 years with a great deal of evolution. As the product line was used to illustrate topics in various courses, that material was added to the paper. This release captures the current state of the product line artifacts.

The Arcade Game Maker product line is a simple, but comprehensive, example. Arcade Game Maker is a fictitious company. We provide meta-information about the organization, its personnel, and the general setting for the example. This information is important to understanding the decisions made at various points in building the product line.

The example has two distinct parts:

- the actual product line assets and products
  - business case
  - scope
  - concept of operations
  - requirements
  - architecture
  - production plans
  - test plans
  - code assets for three products
- a pedagogical section that includes
  - class-tested pedagogical elements of the product line
  - suggested exercises using the assets of the product line

Support for this effort was provided by

- The Product Line Systems Program of the Software Engineering Institute
- Luminary Software

– Clemson University Department of Computer Science

Before looking at any of the assets, you need some background on AGM. The introduction presents information about the company and its employees. This information sets the context for the example. This is not part of the usual product line artifacts; rather it is provided for pedagogical purposes.

The arrows in Figure 1 indicate dependencies among the materials. Generally you will first want to study the contents of the materials on which other materials depend. To understand the example, we suggest that you begin with the Business Case section. Managers may wish to then go to the Concept of Operations section while technical people may prefer to read the Scope next. The arrows and your own needs will dictate how you explore the remainder of the sections.

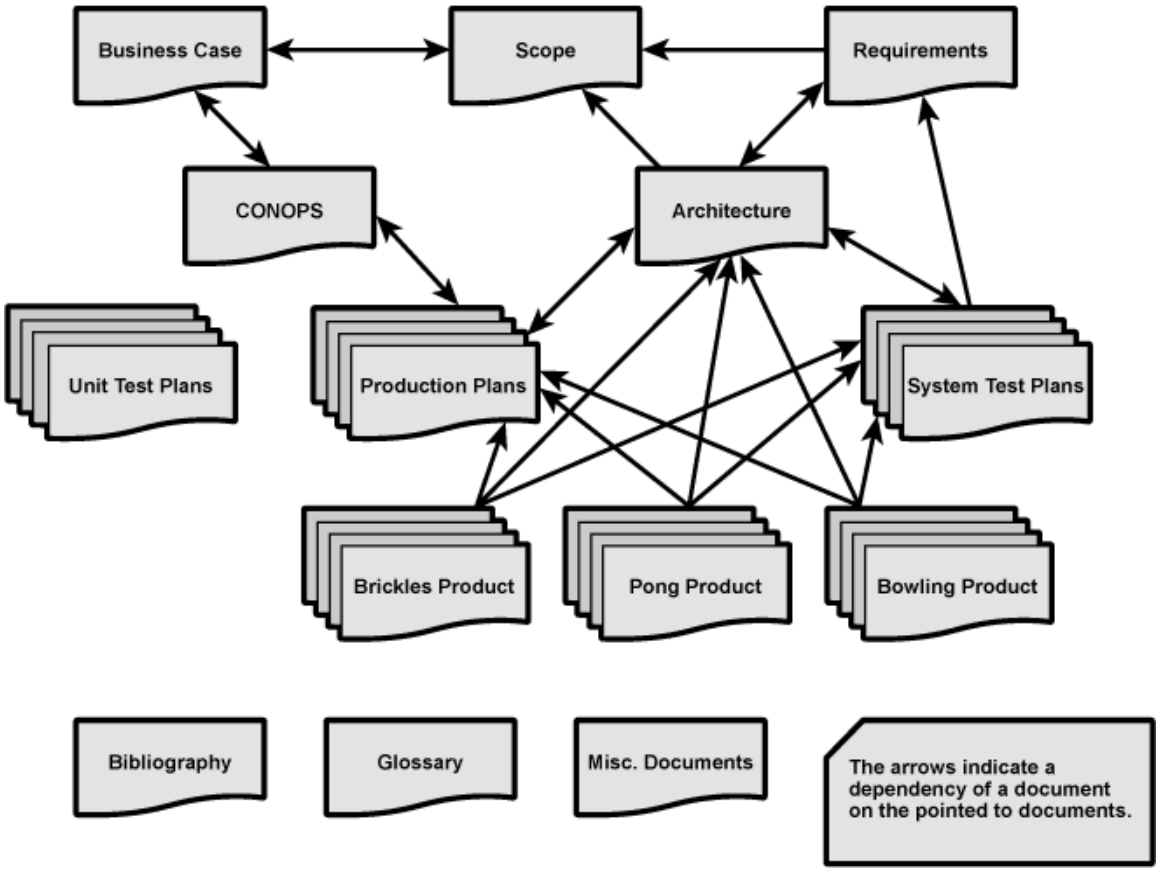

Figure 1: Document Map, Showing Information Dependencies

Current status – As you enter the AGM product line, the team is in the first increment of products. These products are all freeware that will be downloadable from the AGM website. The product line planning documents such as the business case, scope, and the architecture are complete. The team has an implementation for all three games but the Bowling product has distinct problems, which will be addressed by that product team as soon as possible. Neither the Pong nor Bowling products have passed the system test phase. Every item on the map Figure 1 has some valuable information, and most of the items will be evolving over the next few months as the team moves forward.

This section is the gateway into the Arcade Game Maker (AGM) product line example and includes supporting information that is not in standard product line documents. For example, included in this paper is an overview of the structure of the fictional company that produces the product line. As such, this section is not part of the product line but supports various pedagogical devices such as problem scenarios.

This case study presents an example software product line. The example is intended for several types of use including training and self-study. The example follows the approach to software product lines described by Clements and Northrop [Clements 02].

This case study, which examines a company that has adopted the product line approach, covers a three-increment process that the company went through to reach a stable product production process. The product line experiment will build three game products, each in three variations. The case study provides a number of assets and tracks the evolution of the product line as well as the decisions that led the product line to its current state. The AGM example is meant to be an actual product line with assets sufficient enough for readers to extend the product line to include additional products with very little effort and contribute additional assets.

The Company

Type

AGM, a subsidiary of a multinational corporation, produces a series of software-intensive products delivered directly to retailers that, in turn, sell them directly to individual consumers. The company is one of several subsidiaries that share portions of the product roadmap established by the parent corporation. They all make similar products but for somewhat different markets.

The remainder of this paper will refer to the subsidiary as “the company” or “AGM.”

Structure

AGM has a chief executive officer (CEO) who reports to the corporation’s chairman of the board. Although the chairman has line authority over the AGM CEO, the CEO has a great deal of latitude in achieving the objectives identified by the corporation.

The CEO has several direct reports (see Figure 2), including a vice president for product development (VPPD), a vice president for product planning (VPPP), and a vice president for business affairs (VPBA). The local director of human resources also reports to the CEO.

There are a number of horizontal interactions among these positions as illustrated in Figure 2. The director of human resources (DHR) controls the training budget for AGM. The VPPD must coordinate with the DHR to train managers and engineers in new techniques. The VPPP has primary responsibility for the product roadmap, while the VPPD is attempting to develop an efficient means of producing those products.

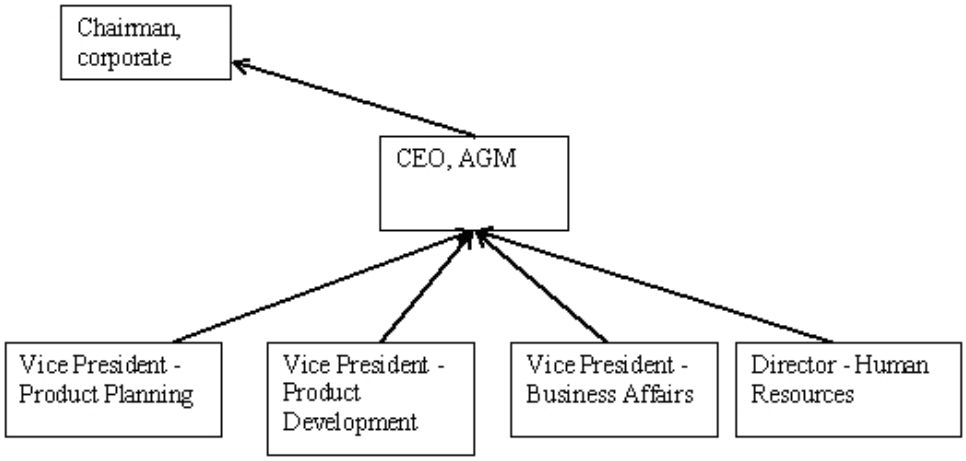

Figure 2: CEO Interactions

The engineering personnel all report to the VPPD as shown in Figure 3. Originally, there was one product team that encompassed all the engineers. Within that team, the engineers were divided into functional specialties. The hardware teams built the game boxes, and the software teams developed the game code. As the product line was initiated, AGM decided to maintain the specialty approach by maintaining the functional teams while also having a project-based approach in which each product, as well as each major asset, is viewed as a project. Personnel are "matrixed" into a project (see figure). Initially, a core asset team and the first product team were formed as projects. Now, there are three active product teams and one core asset team.

Each product team is responsible for one game, including all three variations of that game. The core asset team is responsible for delivering core assets to each product team and for making the infrastructure for each different environment as transparent as possible.

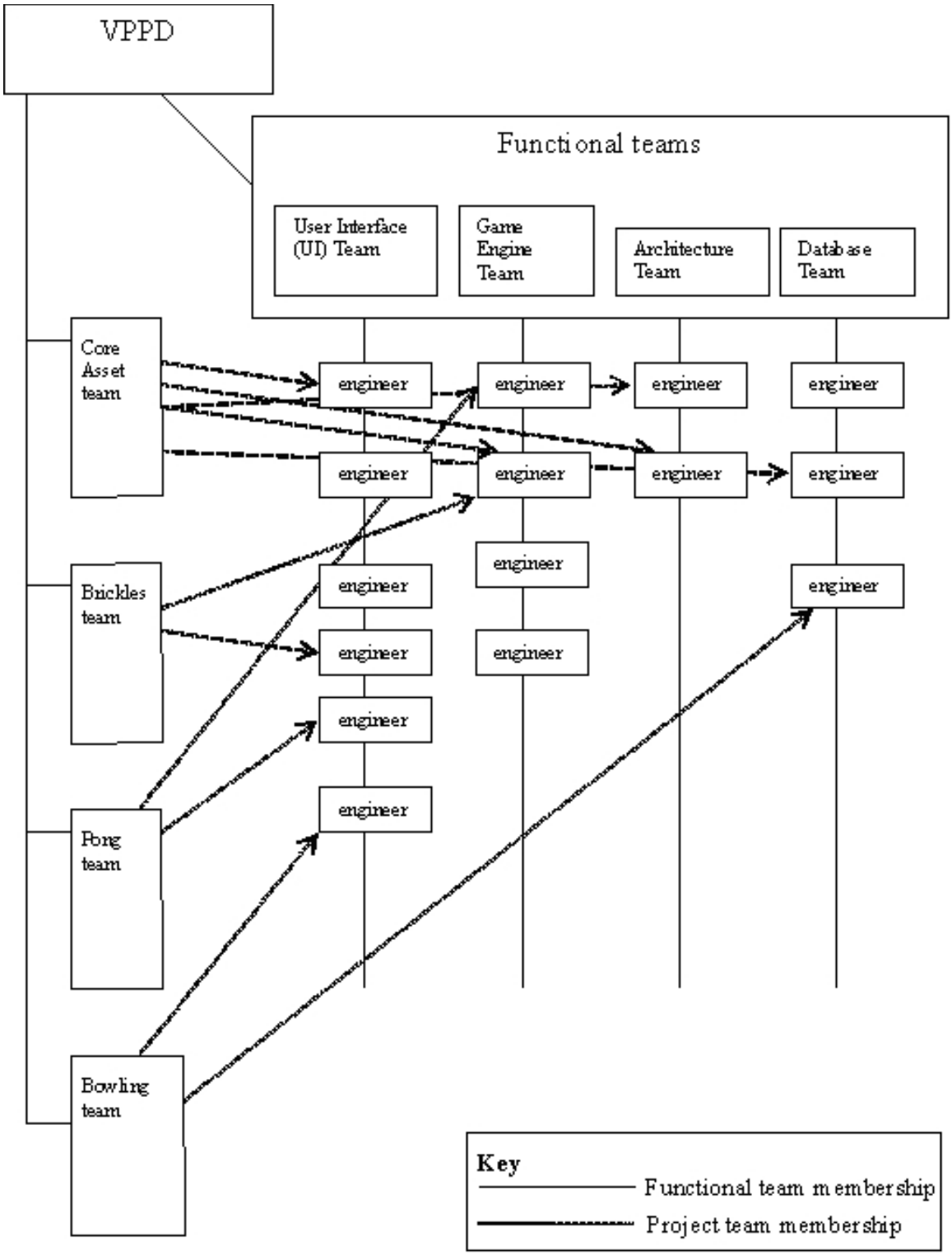

Figure 3: VPPD's Matrix Development Organization

Core Asset Development Project Charter

The Core Asset Development Team is responsible for providing the Product Development teams with high-quality assets that meet their needs in a timely manner. The team will work within the constraints of the specified production method to produce assets that are compatible with the manner in which products will be produced by the product development teams. The team will negotiate schedules with the product development teams to ensure timely delivery of the assets. The Core Asset Development Project will be an on-going project with no pre-determined completion date. The project will only terminate when the product line is terminated. However, the initial level of activity will be much higher than eventually when refined versions of most of the assets have been delivered. During this lowering of activity, core asset development team personnel will gradually be transferred to product development teams wherever possible. The team lead will report to the product line manager.

Product Development Project Charter

The product development teams are responsible for delivering products that meet the functional and quality product requirements. The team is responsible for developing any product-specific functionality not accommodated by variation mechanisms in the core assets. The team will work within the constraints of the product-specific production plan for the product to be built.

Each Product Development Project will have a definitive termination point defined in its production plan. The schedule in that plan will identify headcounts for various points in the project and the anticipated termination date. The team lead reports to the product line manager.

Strategic Objectives

The company has identified four strategic objectives. The AGM product line is expected to contribute directly to the attainment of those objectives.

Market Position

AGM will be a market leader. Currently two other companies have a larger market share. This market is sensitive to how rapidly new technologies are introduced into products and the scope of the feature set. The company has been a “late adopter” of new technologies such as C++ and Java. To achieve this strategic objective, the company decided it must become at least an early adopter.

Time to Market

The company will be able to produce products at an increasingly rapid rate. Ideas for games come from a number of sources. Many ideas come from the popular media where an idea has a very short life span. AGM must be quick to develop and deploy any games based on the popularity of a media or sports figure or one inspired by an actual event.

Productivity

AGM will increase productivity so that the effort per product decreases. Software makes up roughly 90% of the content of current products. To remain competitive, AGM must reduce the cost of building the games.

Mass Customization

The company will be able to serve more specialty markets. The current product development process requires too many resources to make products with projected sales of under a million units profitable. For example, the parent company's marketing division sees an opportunity in the area of convention giveaway products. They would like to be able to add a company's logo and other advertising marks to a game and sell it to that company as a marketing handout at conventions.

Funding for the Organization

The initial AGM product line is funded from the CEO's discretionary account as a pilot project at the request of the VPPD. The funding model is structured with a checkpoint at the end of each of three increments. The CEO has set expectations for each of these checkpoints and reserves the right to cancel the pilot at any of the checkpoints. At the end of the first increment, the freeware increment, no profit will have been made but the organization should show a trend toward a substantial increase in productivity. By the end of the second increment, there should be a positive trend in profit. At the end of the third increment, evidence of the improvements resulting from the software product line approach should be sufficiently positive to propagate the product line idea to the rest of the corporation.

Future funding for product line efforts will be evaluated using the SIMPLE and the data from the pilot. As sets of products are identified, the return on investment for using the product line approach will be computed as the basis for decision making. The product line strategy will be used when it is economically beneficial to do so.

Current Development Environment

In this section, we describe the software development environment prior to the use of the product line approach.

Language History

The company began manufacturing games in C in the late 1970s and migrated to C++ in the late 1980s. In the mid-1990s, the company switched to Java as the primary development language but kept some core functionality in C++. The company originally used simple applets as the basis for the game interface but has now begun to use a variety of approaches including active server pages for Internet-based games.

Design Paradigm

The company used a structured method with a functional paradigm until the mid-1990s when it began to take advantage of some of the object-oriented (OO) features of C++. Later, when the company switched to Java, it went to a full OO design paradigm with an iterative, incremental development process. Recently, this OO approach has been modified slightly to become a component-based approach.

Development Personnel

The development staff is heavily loaded with people who have been with the company for many years. A few employees have computer science degrees, but more have electrical engineering degrees due to

the early emphasis on specialized game boxes. Only recently has the Human Resources department taken an active role in building a skills profile and actively recruiting software engineers who specialize in the various practices needed by the product line organization..

The staff can best be characterized as “developers.” A developer performs the full life cycle of tasks: analysis, design, and coding tasks. Two developers have been trained on and dedicated to architecture on a part-time basis. The newer developers are paired with more experienced personnel for six months to a year for on-the-job training. Developers perform unit testing, and a separate team handles integration and system testing.

Chronology

This section provides a comprehensive timeline for AGM’s product line activities. This timeline supports the company’s decisions and the evolution of its thinking and artifacts.

1998: May - A team investigates OO analysis and design techniques by creating an implementation of the Brickles game; this internal release involved a domain analysis and a design.

2000: March - AGM considers building freeware versions of games prior to releasing the actual game, as advertising for the “real” thing.

June - An implementation of Brickles is released to the public; the team experimented with domain analysis and OO modeling.

October - AGM’s executive team reviews its strategic objectives and finds the need for a quicker time to market and wider variety of products.

2002: March - The VPPD and lead engineers identify software product lines as a possible solution technology.

April - AGM assigns a scout team to investigate product lines and decides that its first product line will be a set of freeware games.

May - AGM identifies a tentative product roadmap for the freeware product line.

July - AGM reorganizes its engineering staff into two teams: the core asset team and the initial product team.

August - AGM’s scout team attends the Carnegie Mellon Software Engineering Institute’s (SEI’s) Second Software Product Line Conference (SPLC2).

September - Based on their experiences at SPLC2, AGM’s scout team recommends an emphasis on architecture and generating domain-based assets. A production strategy is created.

November - The first version of AGM’s arcade game domain analysis is released, along with a prerelease version of product line architecture.

December - The final version of AGM’s arcade game product line domain analysis is released. The production method is defined and the production plan is created.

2003: February - The first release of AGM’s product line architecture is made available.

April - AGM’s first product reaches final release.

May - Artifacts from AGM’s first product are reengineered to be core assets.

August - AGM’s three freeware games have been constructed using the core assets.

Note: All activities beyond this point are tentative.

October - AGM's first client product will be released.

2004: January - The first of three AGM products to run on wireless devices will be released.

2005: January - A new product line will address new challenges.

Executive Profiles

CEO

The CEO was hired into the corporation just as the AGM was being formed. He has several years of experience managing the development of products using game boxes but less so with the software portion of the product. He has been CEO for the life of the company.

He has a degree in electrical engineering. Due to this background, he views product building as the assembly of preconstructed standard parts. Until very recently, he did not understand why it took the software developers longer to produce their portion of the product than it took the hardware staff. When it became obvious that software development was a bottleneck for product development, he took an in-depth look at software development both in general and in his company.

VP for Product Development (VPPD)

The VPPD was hired several years after the subsidiary was formed. She has experience in building software-intensive products but not games. She has a degree in computer science and understands the problems accompanying software development.

The CEO and VPPD complement each other when they take time to fully explain their perspectives on a problem. Without this sharing of information, they often clash because of their different viewpoints. If either takes action without consulting the other, the decision is often modified later after they discuss the situation.

VP for Product Planning (VPPP)

The VPPP has a background in consumer electronics such as game boxes, but he is less familiar with new environments such as wireless devices. He does not understand the implications of a specific feature being on the product delivery schedule or the implications it has on the product's memory consumption or performance.

VP for Business Affairs (VPBA)

The VPBA has a master's degree in business administration but little experience with software-intensive products. For the entire five years he's been with the company, he's been the VPBA. When the product line effort began, he became aware of the problems associated with purchasing software from outside vendors. He still has difficulty understanding the differences between contracts for software and contracts for hardware.

Director of Human Resources (DHR)

The DHR has been the subsidiary company's DHR since its inception. He was in the Human Resources department of the parent corporation before joining the subsidiary. He tends to focus on the myriad of laws affecting the employer/employee relationship. He has not developed any type of training tracking system, nor does he elicit training ideas from the other executives.

The DHR participates in product-oriented discussions but usually contributes only when the issues involve personnel. He often reminds the other executives of the conflicting demands on the development staff. He is often a conduit for expressing developers' concerns to the other executives.

Business Case

This section presents the business case analysis for the Arcade Game Maker (AGM) product line. The product line encompasses a set of arcade games that operate in a variety of environments. The product line approach has not been used in this company before. This business case will examine options for creating the products including using a product line approach. The initial games in the product line will be made available at no cost to promote the company.

Overview

It is proposed that the company initiate a software product line approach to developing sets of related software-intensive systems. This section analyzes the feasibility and viability of producing products using the product line approach.

Our off-site strategic planning session came up with a goal of significantly reducing our costs. During the discussions in this session, we realized that software development now contributes the majority of content to each product. Any serious effort at reducing costs by improving the company's productivity will require a dramatic improvement in software development productivity. A survey of existing techniques has shown that most reuse programs improve the productivity of an organization by 10-15%. While this is a positive result, the magnitude is not sufficient to achieve the gains required by the company.

A review of the software product line literature shows that companies using the product line approach have achieved gains as large as 400%. Results vary depending on the degree to which the products are similar, the maturity of the software development process within the company, and other factors. One action item from the off-site planning meeting was to commission a project to develop a business case for adopting the product line approach in our environment. This section is the report of that project.

The business case development project was chartered jointly by AGM's Vice President for Product Development (VPPD) and Vice President for Product Planning (VPPP). The project team includes a

marketing representative, a requirements representative, one of the software architects, two development managers (one of whom will be the product line manager if the business case is approved) who have project planning and estimating expertise.

Product Line Context

The AGM product line will be implemented in an organization that has historically placed its lowest priority on software engineering. The VPPD decided that, along with adopting the product line strategy, AGM will use an architecture-centric, component-based development method. Due to the problems experienced with previous products and the radical change in development method, no attempt will be made to mine existing products. The expertise of the development staff will be the main assets with which the new product line organization begins. The plan of action for this product line will need to establish basic software engineering practices as well as product line practices.

Relation to Corporate Strategic Objectives

The product line must support the strategic objectives.[1] Here, we show how the product line approach relates to each one.

Become a market leader

The product line approach would make AGM a market leader by improving the company's quality of products, allowing it to pursue specialty markets and reducing the time to market. Hewlett-Packard reported a 25-fold decrease in the number of bugs over four products in a product line [Toft 00]. Others have used the increased flexibility of their products to pursue niches in their domain.

In particular, the marketing group has an idea about customizing games to include sponsors' logos. A company could order giveaway products that feature its message integrated into the game.

Reduce time to market

The product line will allow us to achieve a faster time to market through an order of magnitude increase in software component reuse. Other industrial product companies have achieved three-fold improvements in the time needed to produce a product [Toft 00].

Increase productivity

The product line approach has successfully increased productivity. Hewlett-Packard reported that the head counts on product line projects are four times smaller than similar non-product-line projects [Toft 00]. AGM must achieve higher levels of productivity to remain competitive, but that is not our top priority.

---

1 The product line context section provides a more detailed discussion of the strategic objectives of AGM.

Enable mass customization of products

Companies such as Nokia have improved their ability to address niche markets through an architecture that supports the customization of numerous attributes [Heie 02]. AGM intends to open a new market by providing low-cost games as convention giveaways. We will customize each game to fit certain desires of the client. At this point, the color scheme and logos appear to be major customization points, but that may change after analysis.

Strategies

There are several options for how AGM initiates the basic product line approach. In this section, we consider some standard approaches [Clements 02].

Totally Proactive

In this approach, the assets are built before the products. This approach is the easiest to manage since the focus is solely on building the assets. This approach is difficult to fund because no new products are generating revenue while the assets are being created.

Totally Reactive

In this approach, the assets are built as the products are built. This approach is much more difficult to manage than the proactive one. Asset development must be scheduled just in time for use in a product that has a specific delivery date. There is more rework of assets in this approach be-cause the design of an asset does not fully take into account the needs of later products. Revenue from earlier products supports the costs of developing later products.

Incremental

We have “rolled our own” interpretation of the incremental approach. In our interpretation of this approach, the total product line is divided into sets of products. In our case the sets are (1) the freeware set; (2) the wireless set; and (3) the customizable set. Initially, the architecture and asset designs are intended to specifically support the first set of products and to in general support all of the products. A set’s assets are built before any of its products are.

The product line process is divided into increments. Each increment is intended to construct one set of products. Sets of products are built using the assets developed for the previous increment’s products plus any new assets that need to be constructed.

This approach is an attractive option for the AGM product line. Our vision is that the first increment, which would begin the advertising campaign early, would be the three freeware games that will be given away. The next increment will be a release of those same game products that now operate on wireless devices. The company will now have sufficient assets and experience to enter the new market. The third and final increment will be the set of games to be sold as marketing giveaways. That is, our customer will provide image files and other company related information and we will produce a version of the selected game with their images in prominent places. The customer will then give these away on a disk or memory stick from their booth at a trade show.

The initial analyses will give decreasing emphasis to the features of the products in each of the succeeding increments. The architecture and other initial assets will be developed with evolution in mind. The later increments will plan out the time and resources needed to rework the early as-sets so they support the needs of the entire product line.

This approach also fits well with the software engineering maturity of our organization. The incremental approach will give us time to develop needed practices. The first increment of products will be developed by a team of the best developers and analysts in the company. Then, that set of products will be used to train others in the company.

Environmental Scan

In this section we present the results of our analysis of the strengths, weaknesses, opportunities and threats that characterize the environment in which the AGM product line will be sited. The exact strategy that we propose in the upcoming Feasibility section must be appropriate for this environment.

Strengths

The content of the product line is a set of simple arcade games. Such a set has the strength of being easy for the developers to understand and will reduce the initial costs of the product line.

From the perspective of providing advertisement for the company, a number of extensions to the basic products are possible that could be the basis for additional commercial games if the initial distribution is successful. This approach has the strength of making the games easy to modify.

The development technique we are considering using has been proven effective in actual industrial projects. It has the strength of helping to achieve the strategic objectives of the company.

Weaknesses

The weakness of using simple arcade games as the content of the product line's initial increment is that some developers and managers in the company will dismiss the example as too simplistic. We will counter this by using standard architectures and patterns, even in the initial increment, that would be used in an industrial product line.

The initial product line has the weakness that even a simple product line has a greater complexity than a single program. We will counter that by using innovative techniques to present unique opportunities for interacting with the product line.

The managers of AGM are not rigorous in their decision-making process or careful about lines of authority. The complexity of a product line organization will stress the management capabilities of the company.

The weakness of presenting the product line through a paper is that a product line is an evolving, dynamic entity. Attempting to represent that in a static form makes understanding the dynamics of a product line more difficult. We will attempt to counter that by having multiple versions of the paper to show the evolution.

Opportunities

The company has the opportunity to produce a very complete product line of freeware games at very little cost. As a result, the staff can study the product line approach before tackling for-profit products.

The company has the opportunity to become visible to new customers in existing markets by giving away the products from the product line. The low cost of the products makes this possible. In addition, the assets from the initial, free products will be used to more quickly and cheaply produce the commercial products.

The company has the opportunity to enter a new market by creating a highly customizable architecture. The ability to customize the outward behavior of the game will allow variation with little or no structural impact.

Threats

The biggest threat is that the two current market leaders will adopt the product line approach and take advantage of their leader position to create an even greater advantage. The company needs to move as quickly as possible to begin realizing the benefits of the product line approach.

A second threat is that taking time to build this initial product line may take time away from producing products for which the company will realize actual income. Training is always a long-term investment but a short-term threat.

Analysis of Strategic Factors

Market Viability

The market for the company's products is rapidly expanding as shown in Figure 4. However, consumers expect additional features to be added on a regular basis.

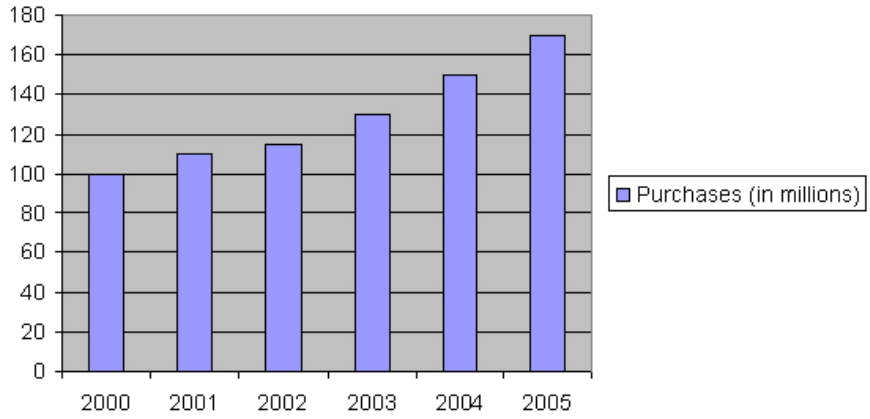

Figure 4: Purchases of Products

The company intends to use the arcade game product line to move into new markets. First, the games of the product line will be customized to run in the reduced environment of a wireless device. There is

a very rapidly growing market for such games, as shown in Figure 5. Then, the games will be renovated to add a mass-customization capability so they can serve as giveaways for companies at conferences and other marketing opportunities.

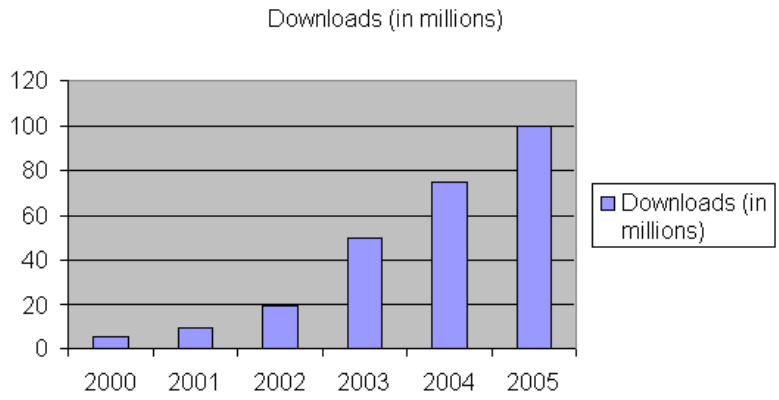

Figure 5: Downloads of Games

Market Segments

The company has developed products for the traditional game markets for some time. Their games were originally targeted to special game processors. The market shifted to the personal computer markets for Windows-based and Apple-compatible systems. This market continues to be viable, as shown in Figure 5. Versions of the game are also developed for several newer game machines, although this is a smaller market.

The company is now addressing additional market segments: those for wireless devices and convention giveaways. The architecture for the product line will include a variation point in terms of the operating environment constraints such as memory size, display size, and operating environment. The architecture will also provide for a number of variations that a company can use to advertise its message.

Feasibility

In this section, we present a cost/benefit analysis for using the product line approach in this environment.

Analysis Technique

The analysis technique we will use is based on the "ABCs" of a business case as described by Cohen [Cohen 03]. The ABCs correspond to the following:

- Applications
- Benefits
- Costs

Applications

In the foreseeable future, the company will produce nine products that fall into three categories: freeware games, wireless games, and giveaway games. The same three games-Brickles, Pong, and Bowling-will be in each grouping. The applications are identified more specifically in the Requirements section.

Benefits

The benefits of using a product line approach include variation management, increased productivity, and a shorter time to market.

- Variation Management

  Variation management is a major benefit of the product line approach. The company produces a series of games that are very similar. The overlap between the games is rather large. The major variations in the products are the environment and the markets for which they are intended.
- Increased Productivity

  The increases in productivity realized by product line organizations are much larger than the increases experienced by organizations that use approaches such as reuse repositories and other reuse schemes alone. Product lines would make a positive contribution to the achievement of the strategic goal to produce more products without additional resources. We expect to reduce the number of staff needed to produce a product.
- Shorter Time to Market

  The reduction in time to market experienced by companies using the product line approach is much larger than for those using only techniques based on commercial off-the-shelf (COTS) components. We expect to be able to achieve a large reduction but gradually over the first few products.

Costs

Using a product line approach involves both direct and indirect costs as explained below.

- Direct Costs

  The direct costs for the product line are in three categories:

  1. personnel - The product line organization will be formed with 20 people initially. This will include 1 executive, 2 managers, 2 technical leads, 13 developers, and 2 testers. This trial organization will produce the three games even if the product line approach is judged unsatisfactory.
  2. tools - We are taking this opportunity to upgrade our development process so that it will be sufficiently robust to support the product line approach. The costs here will involve buying these new tools:
     - a configuration management system
     - an Improved Integrated Development Environment (IDE)
     - a Unified Modeling Language (UML) modeling tool

These costs will be held as low as possible by purchasing only a couple of seats for each tool until we have a better idea of their usefulness.

3. advertising - Even though we are giving the products away for advertisement, people still need to know they are available. We will minimize the costs by limiting our advertisement to Web pages of affinity groups. By agreeing to carry their ads on our customer pages, we can eliminate any real costs.

- Indirect Costs
  - The resources being applied to the product line experiment will not be available to apply to other opportunities. Thus, there is an opportunity cost.
  - The morale of personnel will be damaged if the product line initiation effort is not successful.

Modeling the Approaches

In this section we present models that quantify the costs and benefits that we have described in previous sections. We use SIMPLE to analyze the three strategies that we discussed in the Strategies section. There are a large number of variables that come into play in these models. We could run each model many times varying certain parameters each time, but we will only present a single example of each in this paper. We first consider three approaches and then complete a table tied to the SIMPLE model to compare the three models. (In a real product line business case there would be exact numbers, here we will use relative numbers.)

Two variables that will influence our models are the degree of reuse (DOR) and the cost of reuse (COR). The degree of reuse reflects how much of the product comes from the core asset base. We expect this to be high in a product line but the final scoping of the product line will determine this value. The cost of reuse relates to the technologies that we are using to build the assets. If components are parameterized over their complete range of operation, the COR will be less than if we rely on a strategy of modifying each asset statically.

For these models, time is mostly important with respect to the individual increments. So t = 1 means the first increment. Therefore, t ranges from 0 at initiation to 3 for the release of the third iteration. Later, these time values can be mapped to calendar time so that net present value and other calendar time measures can be computed.

Proactive Approach

In the proactive approach, all the core assets needed for all the products are built before any products. A thorough analysis is conducted, and the complete set of components is constructed. The DOR value is 100% for every product, meaning that each product is produced entirely from the asset base. The full cost of asset maintenance is applied starting with the first product since the entire asset base is available at that point.

The SIMPLE expression is

$$C_{org}(t) + C_{cab}(t) + \sum_{i=1}^{n} (C_{unique}(product_i, t) + C_{reuse}(product_i, t)) + \sum_{j=1}^{nbrBen} B_{ben_j}(t)$$

For the proactive approach, the C_{org} and C_{cab} costs are all accumulated at the start of the product line. Even though the core assets are all built first, each product still has a unique part (that part not shared by any other product).

Incremental Approach

In the incremental approach, the core asset base is built over time. In our case, there is a natural breakpoint between each grouping of products. At these breakpoints, the needs for the next set of products are analyzed and then the needed assets are developed and added to the asset base. In this analysis, the first three products have a DOR of 30%, the second three a DOR of 60%, and the third a DOR of 100%.The cost of asset maintenance is prorated as the asset base grows at the start of each set of new products. The asset base is updated at the start of each increment. The maintenance costs are charged against the first product in each increment since that effort begins as soon as the assets are created.

The SIMPLE expression is

$$C_{org}(t) + F_{cab}(t) * C_{cab}(t) + \sum_{i=1}^{n} (C_{unique}(product_i, t) + C_{reuse}(product_i, t)) + \sum_{j=1}^{nbrBen} B_{ben_j}(t)$$

For the incremental approach C_{org} is mostly expensed at the initiation of the product line. The C_{cab} is spread across the three increments but not evenly. $F_{cab}(t)$ is the percentage of total C_{cab} that is accrued at each increment. In the table below include costs and benefits for each product across the increments.

Reactive Approach

In the reactive approach, the first few products are built without a core asset base. That base is created later by mining the first products for useful assets. This approach is similar to the incremental approach except that the asset base lags behind products rather than preceding the need. One difference is that the increment in the DOR will vary from product to product rather than from set of products to set of products, as in the incremental approach. The cost of asset maintenance is prorated on a product-by-product basis as the asset base grows.

In the reactive approach, products are mined to create the assets. Thus, the first product is created from scratch. The second product is built from assets mined from the first product. The rate at which the asset base grows is determined by the similarity among the products. The DOR for two very different products is lower than that for two more similar products.

The SIMPLE expression is

$$C_{org}(t) + F_{cab}(t) * C_{cab}(t) + \sum_{i=1}^{n} (C_{unique}(product_i, t) + C_{reuse}(product_i, t)) + \sum_{j=1}^{nbrBen} B_{ben_j}(t)$$

For the reactive approach C_{org} is mostly expensed at the initiation of the product line. The C_{cab} is spread across the three increments. $F_{cab}(t)$ is the percentage of total C_{cab} that is accrued at each increment. Each increment uses the work of the preceding increments so the effort decreases across the increments but costs must also reflect a rise in wages.

Conclusions from the Analysis

In Table 1, the costs and benefits for each approach across the increments are rated.

Table 1: Costs and Benefits for Each Approach

	C_{org}	C_{cab}	C_{unique}	C_{reuse}	Benefits
Pro-active	medium	medium	low	high	high
Incremental	medium	low	medium	medium	medium
Re-active	medium	high	high	low	low

Risks

The risks of using the product line approach include increased organizational complexity, a changing product roadmap, and an insufficient reduction in the COR.

Increased Organizational Complexity

Increased organizational complexity can lead to conflicts about how an engineer prioritizes his many tasks. This risk is being mitigated in this product line by starting with a core asset team and a single product team. As managers become more used to a matrixed approach to resource management, additional product teams will be added.

Changing Product Roadmap

The rapidly changing game environment can make it difficult to sustain the roadmap long enough for the product line to be profitable. The benefits realized from a product line are based on similarities between products and the ability to build a large percentage of each product from previously developed components. This risk is mitigated in this product line by having a domain analysis that is created in layers of abstraction. The more general the layer, the longer its information will be viable. Even if some products are cancelled, the more general information will apply to the remaining products and new products as well.

Insufficient Reduction in the COR

The development organization might not be able to modify its approach enough to efficiently reuse assets. The analysis shows that unless the cost of reusing an asset can be held to about 25% of its original cost, the product line approach will not be viable. Often, a developer will change features, not because they need to be changed, but rather because the developer wants to change his or her approach to them. This risk will be mitigated by a strong architecture team who will supervise the design and development process.

Plan of Action

We recommend implementing the incremental approach described earlier. We also recommend that, during the first increment, the existing inventory of games be mined for assets.

The plan of action is detailed in Table 2.

Table 2: Plan of Action

Increment	Responsibility	Task
Freeware Games	Core asset team	Perform analysis
		Develop software architecture
		Mine assets from existing games
		Develop assets for games
	Product teams	Develop Brickles
		Develop Pong
		Develop Bowling
Wireless Device Games	Core asset team	Analysis to identify new assets needed
		Upgrade existing assets to meet needs
		Develop necessary assets
	Product teams	Modify games for Symbian/Series 60 platforms
Giveaway Games	Core asset team	Analysis to identify new assets needed
		Upgrade existing assets to meet needs
		Develop necessary assets
	Product teams	Develop customized games to order

Scope

Identification

The Arcade Game Maker (AGM) product line will produce a series of arcade games. Each game is a one-player game in which the player controls, to some degree, the moving objects. The objective is to score points by hitting stationary obstacles. The games range from low obstacle count to high and will be available on a variety of platforms. This section defines the boundaries of the product line. Design and implementation decisions are made to address the full scope of the product line but with no concern for any characteristics outside the product line.

Concepts

For definitions of basic concepts, see Appendix A: Acronym List and Glossary.

Reusable Components

This section establishes the high-level context for work in the product line. In a product line, components are designed to be reusable within the context of the product line. That is, no attempt is made to make a component "as general as possible." Each design decision is made with regard to the extent of the products in the product line. The Architecture section further refines the context defined in this section.

Readership

This section is intended to provide some level of information to all the stakeholders in the AGM software product line. Managers will find information that supports product planning, architects will find information that supports commonality and variability analysis, and product developers will find the rationale for each product's membership in the product line.

Use in Product Production

The scope document is used in the earliest stages of product production. It is used when the product production process seeks to determine whether it is feasible to build the proposed product as part of the product line. The lists of common features and variations are used to determine the fit between the proposed product and the product line. Later, assets such as the architecture can be used to provide more detailed information about product fit with the product line.

Product Line Context

In-Scope Products

The context of the AGM product line is illustrated in Figure 6. The four games shown inside the circle constitute the current definition of the product line. Those shown outside the circle are out of scope and will not be constructed from the assets built for the product line.

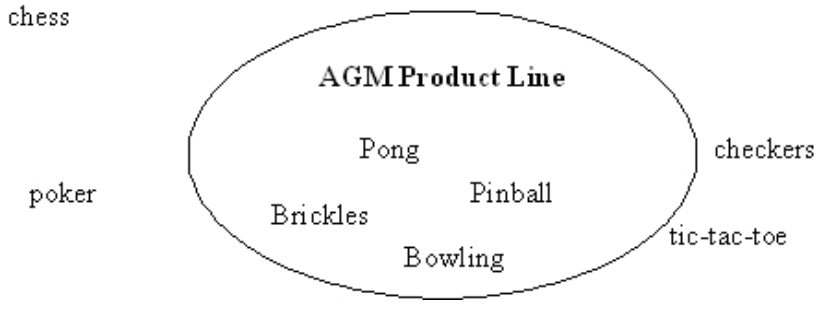

Figure 6: Context Diagram

Each product in the product line

- is a game with specified game elements
- is for a single player
- offers a graphical view of a game or games
- has animation-driven games
- uses moving and stationary objects
- follows certain rules about the interaction of game pieces

The games may have multiple players, but only one is active for a particular match. For example, having a computerized player is useful for testing purposes.

Product Line Variation

The principle variations in the AGM product line are the

- rules of the game
- types and numbers of pieces involved
- behavior of those pieces
- physical environment in which the game operates

These variations are elaborated on in the Arcade Game Maker Architecture Documentation Beyond Views section and the Arcade Game Maker Software Architecture Views section.

Out-of-Scope Products

The AGM product line does not include traditional "board" games. A framework[2] for this type of game was provided by McGregor and Sykes [McGregor 92] but is out of scope for this product line. These games contain an element of strategy not included in the current product line. Board games usually allow multiple players. Strategy games are also out of scope. Dungeons and Dragons and other more recent games will not fit in the AGM product line.

In-Scope Features

Each product will provide the user (called a player) with the ability to interact with the game and to control some portion of the action. The product will operate according to the usual rules of each game and provide a graphical representation of the state of the game.

Out-of-Scope Features

Each product will control hardware through an existing operating environment so variations in hardware will be handled by the operating system

---

[2] A product framework provides a tightly coupled architecture and set of components that can be used to quickly complete any application within the scope of the architecture.

Analysis

The previous sections provided a first level of data capture for scoping the product line. Figure 6 provides a functional context. The attribute/product matrix shown in Table 3 gives a market view of the scope. In this table, the products are divided into three high-level categories, which correspond to three different markets. As the product line expands into commercial products, we should do a more in-depth analysis using a technique such as PuLSE-ECO [DeBaud 99].

Table 3: Example Attribute/Increment Matrix

Feature	Freeware Arcade Games	Commercial PC Games	Commercial Wireless Games
display	standard bitmaps	vector graphics	wireless access protocol WAP bitmap
interaction	mouse/buttons	pointing devices/buttons	buttons
game pieces	simple icons	dynamic icons	dynamic icons
scoring	high scores stored locally	local storage/scores shared with subscription	scores stored on network/subscription required

Feature Model

In this section, we present a high-level feature model of the arcade game domain. This model captures the product features that are necessary to compete in the market and satisfy our existing customers. It includes all features in all products.

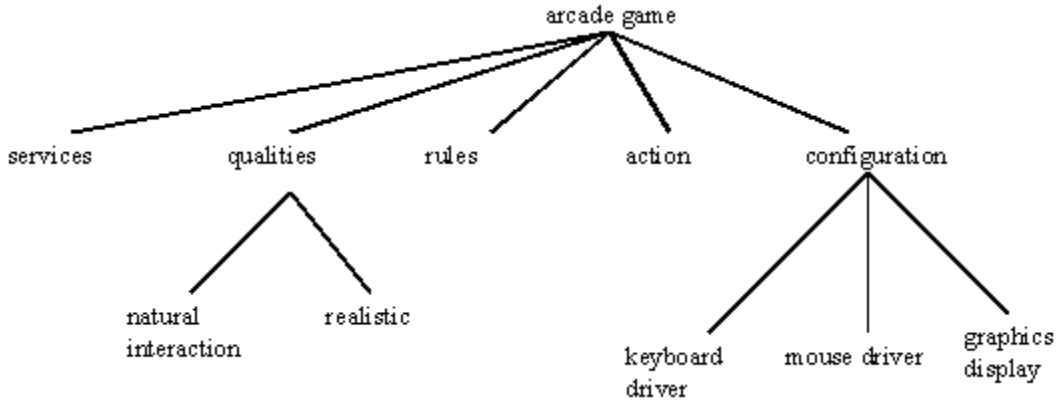

Figure 7: Top-Level Feature Model

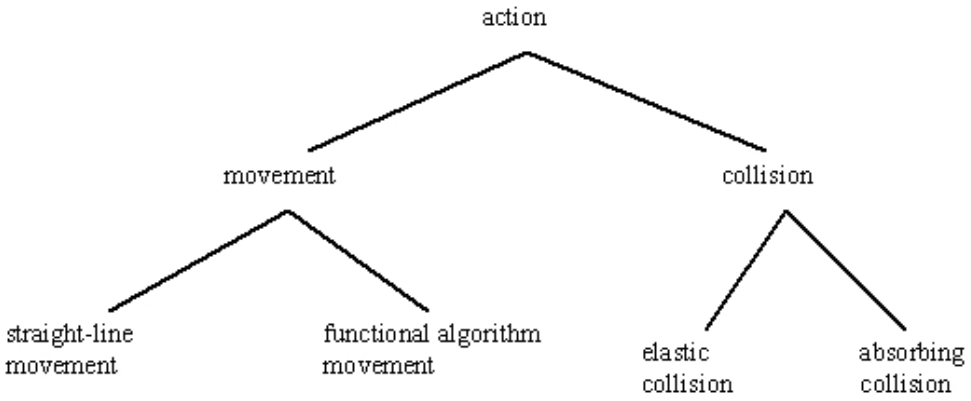

Figure 8: Action Subfeatures

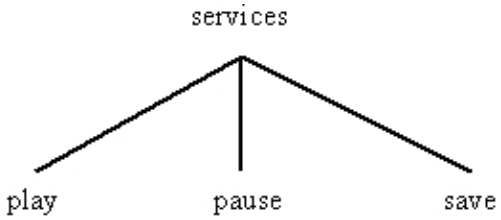

Figure 9: Services Subfeatures

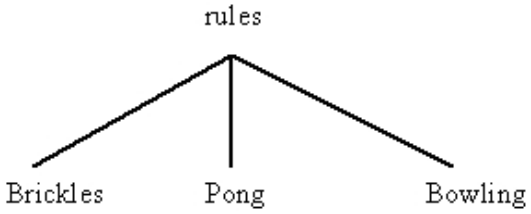

Figure 10: Rules Subfeatures

Determining Product Fit[3]

For a new product to be created using the assets of the product line, the product must be judged to be within the scope of the product line. The process for determining whether a product is in scope will depend on how formally the scope is defined.

For the AGM product line, the scope is partially defined by the attribute/product matrix shown in Table 3. As the process diagrammed in Figure 11 shows, the new product is compared to each column in the matrix. If the new product fits the first column, the new product may belong in the product line. Next, the proposed product would next be evaluated against the architecture description for a more technical match. If the proposed product fits one of the other columns or does not match any of the

---

3 This section is the "attached process" described by Clements and Northrop [Clements 02]. For the product line scope, this process focuses mainly on determining whether a particular product could be produced profitably using the assets of the product line.

columns, additional market analysis should be performed to determine whether the product line scope should be expanded to include the proposed product.

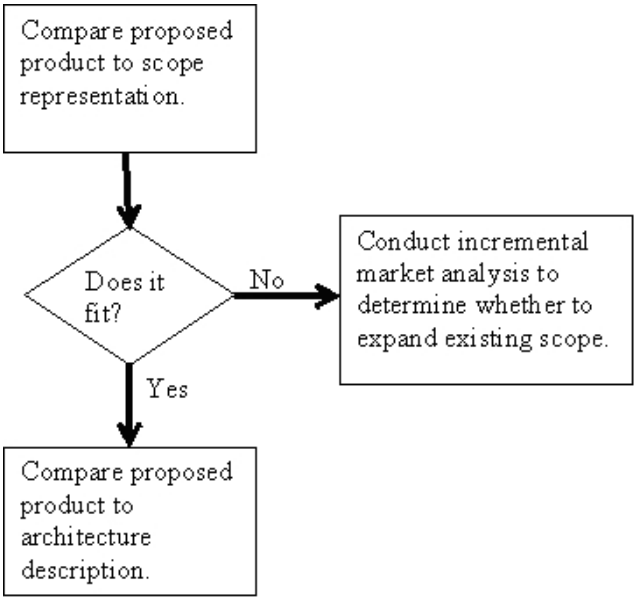

Figure 11: Process Flow for Adding a Product to a Product Line

Requirements

This section is the requirements document for the Arcade Game Maker (AGM) product line. Its purpose is to provide the specifications for the products that will be built as part of the product line.

Concepts

For definitions of basic concepts, see Appendix A: Acronym List and Glossary.

Reusable Components

The requirements document establishes the high-level context for work in the product line. In a product line, components are designed to be reusable within the product line's context. That is, no attempt is made to make a component "as general as possible." Each design decision is made with regard to the extent of the products in the product line. The Architecture section further refines the context defined in this section.

Readership

The requirements document is intended to provide some level of information to all stakeholders in the AGM framework. Managers will find information to support product planning. Product line analysts

will find information to support commonality and variability analysis. Product developers will find the rationale for each product's membership in the product line.

Use Case Model

Figure 12 provides an overview of the use cases for the AGM product line.

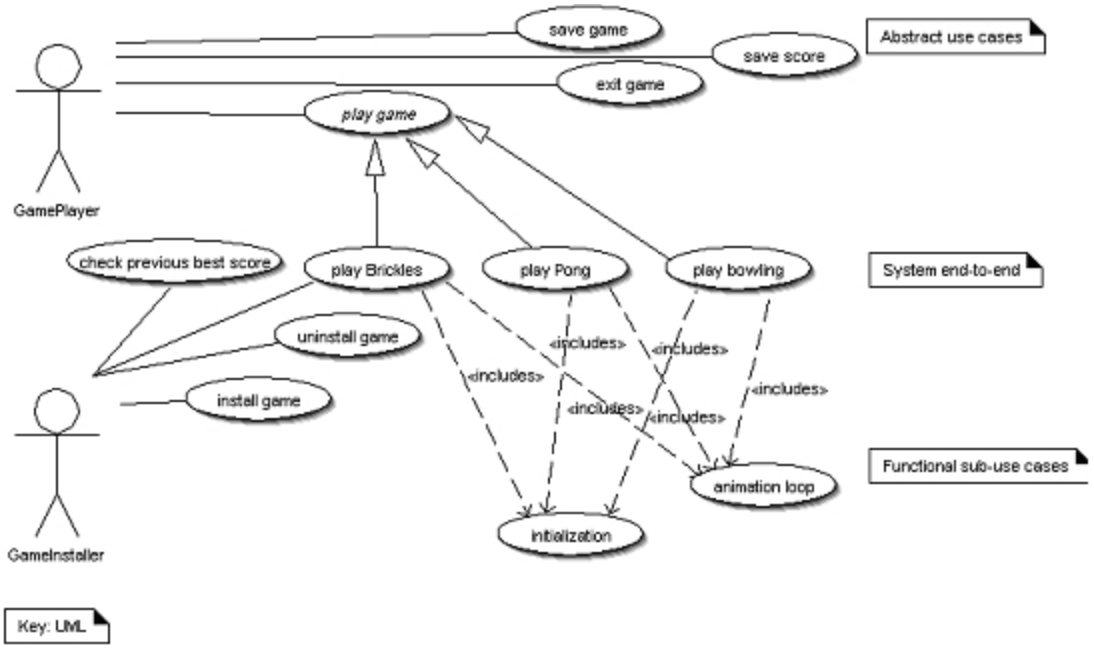

Figure 12: Use Case Diagram

Use Cases
AGM001 - Play selected game
AGM002 - Exit game
AGM003 - Save game (change case)
AGM004 - Save score (change case)
AGM005 - Check previous best score (change case)
AGM006 - Play Brickles
AGM007 - Play Pong
AGM008 - Play Bowling
AGM009 - Initialization
AGM010 - Animation loop
AGM011 - Install game

AGM012 - Uninstall game

Additional use case details are located at the end of this section.

Domain Model

Before specific requirements were identified, a domain analysis (see Figure 13) was conducted to identify essential concepts. These concepts form the basic vocabulary that describes the use cases and the commonality and variability analysis.

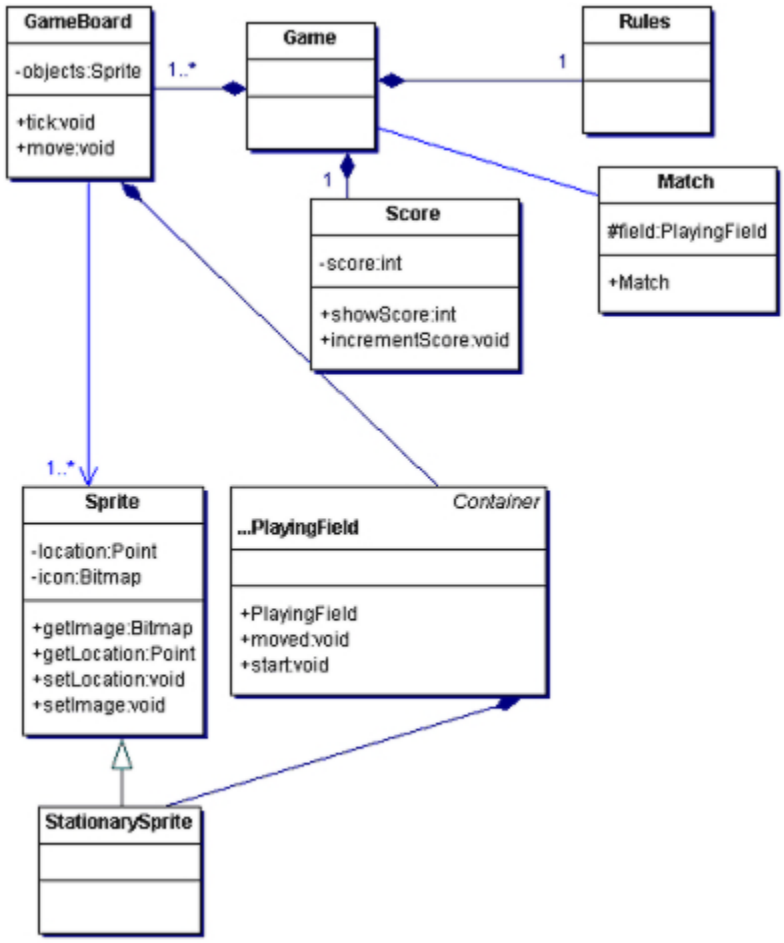

Figure 13: Domain Model in UML

Commonality and Variability Analysis

In a product line, the commonality and variability analysis covers a set of products rather than a single one. In this section, we document what is common among products in the AGM product line.

Definitions

Sprite. From some of the earliest days of computer-based games, the elements that players see and interact with on screen have been referred to as Sprites.

Rule. Game play and operation is governed by rules. For example, a game may have a rule that a moving sprite that strikes a stationary sprite obeys the laws of classical mechanics. Games also have rules about scoring.

Commonalities

- Every game will have a set of Sprites.
- Every game has a set of rules.
- All the games involve motion.

Variations

Rules. This is the biggest variation amongst games. Some of the rules relate to basic physical laws (e.g., gravity or elastic collisions) and may be applicable to multiple games. Other rules relate to the specifics of a game and can be used in all implementations of that game, but they don't apply to other games.

Motion initiation. In some games, motion is inherent in the operation of the game. It happens periodically and is driven by time. In other games, the player selects and initiates motion. It is driven by the player's actions.

Parameters of Variation

Parameters are defined at several levels to provide the maximum variation among products. The set of Sprites that make up a game can be varied to enhance or change the game. In Brickles, for example, different types of bricks can be defined to exhibit different types of behavior during a collision.

Issues

- What is the scope within which a variation is constant?
- Should parameterized units be saved as assets, or should they always be built up for each product?

Scenarios

- The developer is assigned to build a new product that incorporates a previously implemented game. Many assets will be available.
- The developer is assigned to build a new product that incorporates a game that has not been implemented before. Some assets are available because some of the physical rules are the same as in other previously implemented games. Scoring rules will have to be constructed from scratch.

Feature Model

The figures in this section present the results of the feature-oriented domain analysis (FODA) for the arcade game domain.

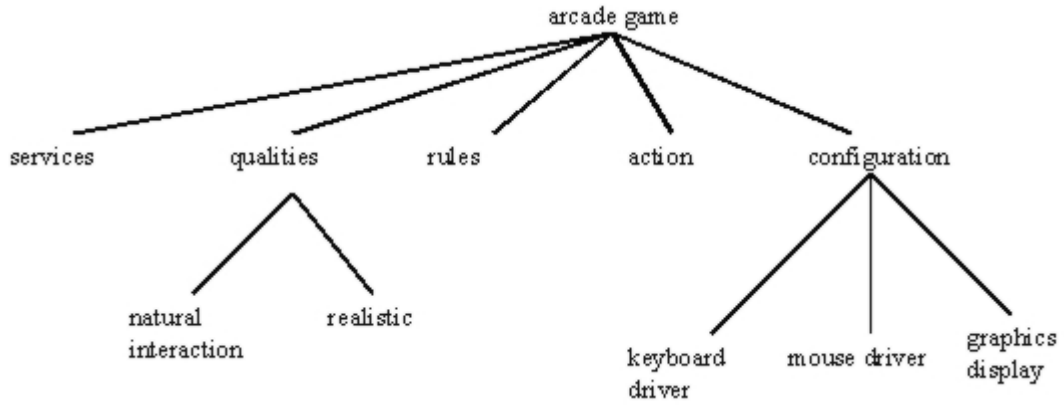

Figure 14: Top-Level of Feature Analysis

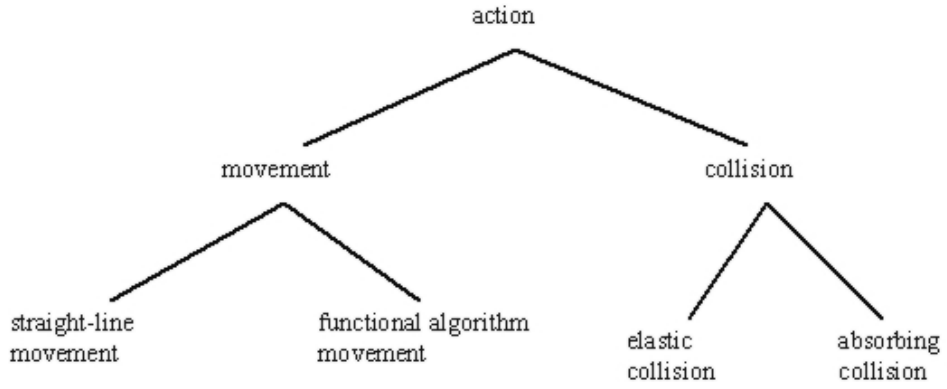

Figure 15: Continuation of Feature Analysis

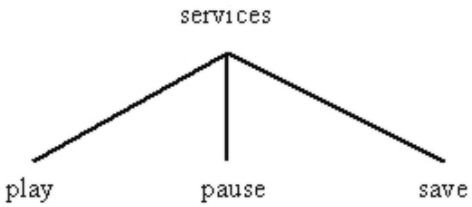

Figure 16: Feature Analysis of the Services Feature

Nonfunctional Requirements

This section describes requirements that do not lend themselves to use cases.

System Operational Requirements

Performance

The action of the game must be fast enough to seem continuous to the user. There should be no blurring of the graphics. Research in human computer interface has shown that such fast action requires a refresh every tenth of a second.

Display Quality

The colors chosen for the games must be such that one element's color does not impair the user's view of another element. No provision will be made for color blindness since the games seldom have similarly shaped game pieces that differ only by color.

System Development Requirements

Evolvability

The assets used in the first increment will be the basis for the products in the next two increments. For the second increment, one programmer must be able to modify first increment assets in less than a week and the architecture in less than two weeks.

Maintainability

Some of the core assets will age as new environments are being deployed. Assets must be maintained with the components with which they interact. For a new release of the environment, one experienced systems programmer must be able to integrate the assets in three days.

Attached Processes[4]

Building the Requirements Model

AGM will use the requirements modeling technique defined by Chastek and colleagues [Chastek 01] and shown in Figure 17. Since this technique is for a product line, each activity refers to all AGM products.

---

4 This section is the "attached process" described by Clements and Northrop [Clements 02]. For the product line requirements model, this process defines how the requirements model is initially built. The process focuses mainly on modifying the requirements model of the product line to represent one particular product.

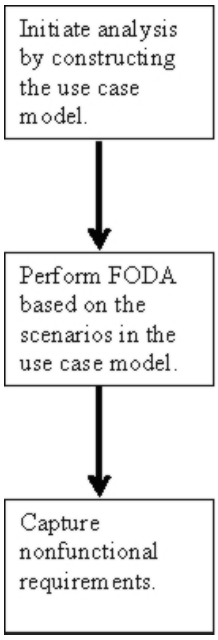

Figure 17: Building a Requirements Model

Modifying the Requirements Model

The requirements model will evolve over time as requirements are added, deleted, modified from a change case to a current requirement, or modified to change the scope of the requirement. Each time a requirement is added or deleted, the person making the change must check the model for consistency, correctness, and completeness as shown in Figure 18.

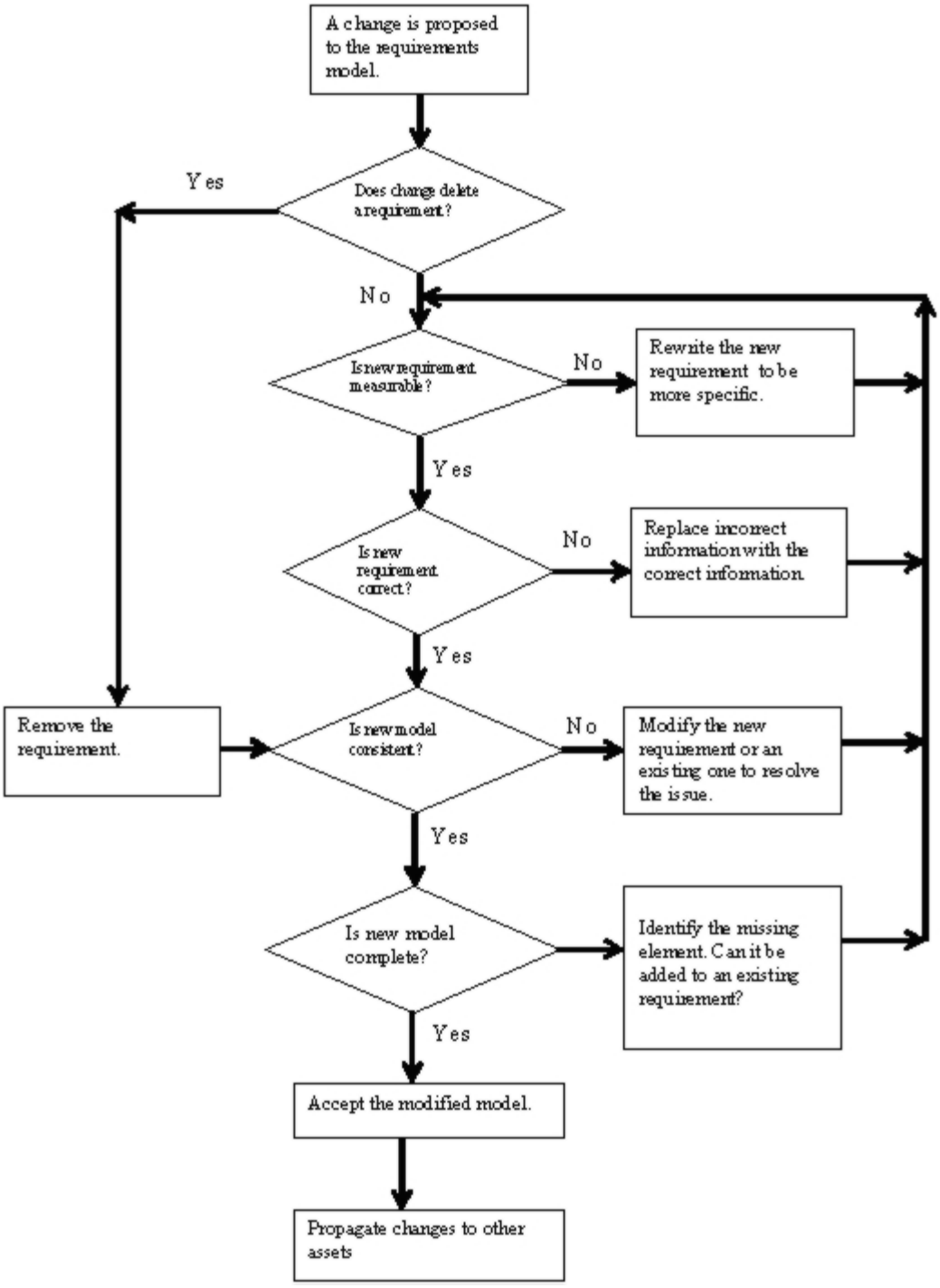

Figure 18: Process Flow for Adding or Deleting a Requirement

Use Cases

Play Selected Game

Use Case ID: AGM001

Use Case Level: Abstract

Scenario

Actor: GamePlayer or GameInstaller

Preconditions: AGM011: Install game has completed successfully.

Detailed Description

Trigger: Actor selects game executable and initiates execution.

Actor	System Response
Selects PLAY from the menu	Initializes the game and displays the gameboard
Left-clicks to begin play	Starts game action
Left-clicks or uses keyboard to enter commands	Responds to the command in the expected manner
Responds to Won/Lost/Tied dialog box with left-click	Returns the gameboard to its initialized, ready-to-play state

Postconditions: Actor has won/lost/tied and the game is ready to play again.

Alternative Courses of Action

No. 1:

Actor	System Response
At any time, may select EXIT from the menu	See AGM002: Exit game

Extensions

List:

Actor	System Response
See AGM006: Play Brickles	
See AGM007: Play Pong	
See AGM008: Play Bowling	

Exceptions

List:

Actor	System Response
N/A	N/A

Concurrent Uses: N/A

Related Use Cases: AGM002: Exit Game

External Supporting Information

Requirement Originator: Domain analyst

Rationale for Requirement: Main purpose of the program

Additional Relevant Requirements: N/A

Decision Support

Frequency: On demand

Criticality: High

Risk: Low

Exit Game

Use Case ID: AGM002

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer or GameInstaller

Preconditions: Game system is running.

Detailed Description

Trigger:

Actor	System Response
Selects EXIT from system menu	Prompts actor to save or exit the game
Saves game	Saves the game and exits the program

Postconditions: Game system is terminated.

Alternative Courses of Action

No. 1:

Actor	System Response
Cancels the EXIT action	Returns to suspended action

No. 2:

Actor	System Response
May initiate EXIT by left-clicking the upper right-hand corner of the game window	Prompts actor to save or exit the game
Saves game	Saves the game and exits the program

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
N/A	N/A

Concurrent Uses: N/A

Related Use Cases: AGM001: Play selected game

External Supporting Information

Requirement Originator: Product planning

Rationale for Requirement: Usual action for an interactive program

Additional Relevant Requirements: N/A

Decision Support

Frequency: Low - only once per game start-up

Criticality: Low

Risk: Low

Save Game (Change Case)

Use Case ID: AGM003

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer or GameInstaller

Preconditions: Game executable is running, and a game has been started.

Detailed Description

Trigger:

Actor	System Response
Selects the SAVE option in the system menu	Allows the actor to specify a filename
	Writes game data to the file
	Returns to the game's pre-SAVE status

Postconditions: Current state of the game has been written to the specified file OR action has been cancelled.

Alternative Courses of Action

No. 1:

Actor	System Response
Selects the EXIT menu option	See AGM002: Exit Game

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
Selects the SAVE option in the system menu	Prompts the actor to specify a filename
	Raises exception because the disk is full

Selects a different disk	Attempts to save again
	Identifies an existing file with the same name as specified in the Save dialog
	Raises ExistingFileException
Chooses a different name OR Agrees to overwrite the existing file	Writes the file

Concurrent Uses: N/A

Related Use Cases: AGM002: Exit Game

External Supporting Information

Requirement Originator: Product planning

Rationale for Requirement: Convenience feature for users

Additional Relevant Requirements: N/A

Decision Support

Frequency: On demand

Criticality: Medium

Risk: Medium - other files might be corrupted.

Save Score (Change Case)

Use Case ID: AGM004

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer

Preconditions: Game system is running and a game is in progress

Detailed Description

Trigger:

Actor	**System Response**
Selects SAVE SCORE from system menu	Prompts actor to specify a filename
	If the file does not exist, creates new file
	Writes score to the file

	Returns to the game's pre-SAVE status

Postconditions: File has been written, or action has been cancelled

Alternative Courses of Action

No. 1:

Actor	System Response
Selects SAVE SCORE from system menu	Prompts actor to specify a filename
	If the file exists, overwrites existing score Else creates new file and writes score

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
Selects the SAVE SCORE option in the system menu	Prompts actor to specify a filename
	Raises exception because the disk is full
Selects a different disk, if available	Attempts to save again

Concurrent Uses: N/A

Related Use Cases: AGM005: Check Previous Best Score (Change Case)

External Supporting Information

Requirement Originator: John D. McGregor

Rationale for Requirement: To motivate the user

Additional Relevant Requirements: N/A

Decision Support

Frequency: On demand

Criticality: Medium

Risk: Low

Check Previous Best Score (Change Case)

Use Case ID: AGM005

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer

Preconditions: Game system is running.

Detailed Description

Trigger:

Actor	System Response
Selects Check Previous Best Score	Prompts actor to specify a filename
	Reads the file and returns score in a dialog box
Selects OK on dialog box to continue	Returns to state before select

Postconditions: Stored score has been shown to actor.

Alternative Courses of Action

No. 1:

Actor	System Response
N/A	N/A

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
Selects Check Previous Best Score	Prompts actor to specify a filename
	Finds that file does not exist
Selects OK on dialog box to continue	Returns to state before select

Concurrent Uses: N/A

Related Use Cases: AGM004: Save Score (Change Case)

External Supporting Information

Requirement Originator: Product planning

Rationale for Requirement: Motivating feature

Additional Relevant Requirements: N/A

Decision Support

Frequency: On demand

Criticality: Medium

Risk: Low

Play Brickles

Use Case ID: AGM006

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer or GameInstaller

Preconditions: AGM011: Install game has completed successfully.

Detailed Description

Trigger:

Actor	System Response
Selects PLAY from the menu	Initializes the game and displays the gameboard
Left-clicks to begin play	Starts game action
Left-clicks or uses the key-board to enter commands	Moves the paddle horizontally to follow the mouse track

	After each movement of the puck, system checks for a collision with another object. If puck collides with the ceiling, it is reflected back into the playing area. If the puck collides with a wall, it is reflected back into the playing area. If the puck collides with the floor, it ceases to exist. If the maximum number of pucks has not been reached, requests and provides a new puck. If the maximum has been reached, the Lost dialog is presented. If the puck collides with a brick, defines action by the type of brick. When the puck collides with the last brick, the Won dialog is presented.
Responds to Won/Lost dialog box with left-click	Returns the gameboard to its initialized, ready-to-play state

Postconditions: Game has been played

Alternative Courses of Action

No. 1:

Actor	System Response
N/A	N/A

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
N/A	N/A

Concurrent Uses: N/A

Related Use Cases: N/A

External Supporting Information

Requirement Originator: Product planning

Rationale for Requirement: Main action for one of the products

Additional Relevant Requirements: N/A

Decision Support

Frequency: On demand

Criticality: High

Risk: Low

Play Pong

Use Case ID: AGM007

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer or GameInstaller

Preconditions: AGM011: Install game has completed successfully.

Detailed Description

Trigger:

Actor	System Response
Selects PLAY from the menu	Initializes the game and displays the gameboard
Left-clicks to begin play	Starts game action

Postconditions: Game has been played.

Alternative Courses of Action

No. 1:

Actor	System Response
N/A	N/A

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
N/A	N/A

Concurrent Uses: N/A

Related Use Cases: N/A

External Supporting Information

Requirement Originator: Product planner

Rationale for Requirement: Main action of one of the products

Additional Relevant Requirements: N/A

Decision Support

Frequency: On demand

Criticality: High

Risk: Low

Play Bowling

Use Case ID: AGM008

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer or GameInstaller

Preconditions: AGM011: Install game has completed successfully.

Detailed Description

Trigger:

Actor	System Response
Selects PLAY from the menu	Initializes the game and displays the gameboard
Left-click to begin play	Starts game action
Repeat the following for 10 frames plus a bonus throw	
Positions the mouse and left-clicks to send ball down alley	Moves the ball down the alley using a randomly selected algorithm. If collisions result when the ball reaches the pins, moves pins as determined by the physics of the collision.
	Counts number of pins knocked down

Positions the mouse and left-clicks to send ball down alley	Moves the ball down the alley using a randomly selected algorithm. If collisions result when the ball reaches the pins, moves pins as determined by the physics of the collision
	Computes score

Postconditions: Game has been played.

Alternative Courses of Action

No. 1:

Actor	System Response
N/A	N/A

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
N/A	N/A

Concurrent Uses: N/A

Related Use Cases: N/A

External Supporting Information

Requirement Originator: Product planning

Rationale for Requirement: Main action of one of the products

Additional Relevant Requirements: N/A

Decision Support

Frequency: On demand

Criticality: High

Risk: Low

Initialization

Use Case ID: AGM009

Use Case Level: Function sub-use case

Scenario

Actor: N/A

Preconditions: AGM006: Play Brickles, AGM007: Play Pong, or AGM008: Play Bowling has begun operation.

Detailed Description

Trigger:

Actor	System Response
	Creates the standard instances of the required classes
	Enters the READY state

Postconditions: Game is ready to operate.

Alternative Courses of Action

No. 1:

Actor	System Response
N/A	N/A

Extensions

List:

Actor	System Response
Selects Load Game from the menu	Presents a file chooser box
	Opens the file that is indicated
	Reads and constructs the game objects

Exceptions

List:

Actor	System Response
	Runs out of memory while creating objects
	Displays the Error dialog box
	Destroys objects already created

Concurrent Uses: N/A

Related Use Cases: N/A

External Supporting Information

Requirement Originator: System engineer

Rationale for Requirement: Natural module within the program

Additional Relevant Requirements: N/A

Decision Support

Frequency: Once per game start-up

Criticality: High

Risk: Medium

Animation Loop

Use Case ID: AGM010

Use Case Level: Functional sub-use case

Scenario

Actor: N/A

Preconditions: AGM009: Initialization has operated successfully and the user has left-clicked.

Detailed Description

Trigger:

Actor	System Response
	Generates periodic signals and sends them to the game
	Moves all objects one step according to their movement algorithm
	Checks for collisions and executes the collision algorithms of the objects

Postconditions: Game is completed.

Alternative Courses of Action

No. 1:

Actor	System Response
Holds down the left mouse button	Pauses the movement of the game

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
N/A	N/A

Concurrent Uses: N/A

Related Use Cases: N/A

External Supporting Information

Requirement Originator: System engineer

Rationale for Requirement: Main action sequence in a game (standard in all products)

Additional Relevant Requirements: N/A

Decision Support

Frequency: Once for every playing of the game

Criticality: High

Risk: Medium

Install Game

Use Case ID: AGM011

Use Case Level: System end-to-end

Scenario

Actor: GameInstaller

Preconditions: N/A

Detailed Description

Trigger:

Actor	System Response
Selects the installer executable to execute	Presents a file chooser to allow selection of a directory in which to place the game files
Selects a directory	Places game files in the directory

Postconditions: Game is installed.

Alternative Courses of Action

No. 1:

Actor	System Response
N/A	N/A

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
	Finds insufficient space to which to write files
	Displays the Out of Space dialog box
Left-clicks on OK button	Exits the program

Concurrent Uses: N/A

Related Use Cases: AGM012: Uninstall Game

External Supporting Information

Requirement Originator: Product planning

Rationale for Requirement: Necessary to get game operational

Additional Relevant Requirements: N/A

Decision Support

Frequency: Very seldom

Criticality: High

Risk: Low

Uninstall Game

Use Case ID: AGM012

Use Case Level: System end-to-end

Scenario

Actor: GameInstaller

Preconditions: AGM011: Install game completed successfully.

Detailed Description

Trigger:

Actor	System Response
Selects UNINSTALL from the system menu	Presents a file chooser to the actor
Selects directory where game is stored	Erases files in the directory
	Presents the Uninstall Completed dialog box
Selects the OK button in the dialog box	Closes dialog box

Postconditions: All disk space taken up by the game is reclaimed.

Alternative Courses of Action

No. 1:

Actor	System Response
N/A	N/A

Extensions

List:

Actor	System Response
N/A	N/A

Exceptions

List:

Actor	System Response
N/A	N/A

Concurrent Uses: N/A

Related Use Cases: AGM011: Install game

External Supporting Information

Requirement Originator: Product planning

Rationale for Requirement: Feature of the product

Additional Relevant Requirements: N/A

Decision Support

Frequency: Very seldom

Criticality: Low - can be done manually

Risk: Medium - might erase the wrong files

Actor Profiles

GamePlayer

Abstract: No

Description: Usual, frequent user of game

Skill Level: Medium

Actor's Perspective on Use Cases			
Use Case	**Importance**[5]	**Personality**[6]	**Relative Frequency**[7]
AGM001: Play selected game	Primary	Initiator	High
AGM002: Exit game	Secondary	Initiator	Low
AGM003: Save game (change case)	Secondary	Initiator	Medium
AGM004: Save score (change case)	Secondary	Initiator	Low
AGM005: Check previous best score (change case)	Secondary	Initiator	Low
AGM006: Play Brickles	Secondary	Initiator	High
AGM007: Play Pong	Secondary	Initiator	High
AGM008: Play Bowling	Secondary	Initiator	High

GameInstaller

Abstract: No

Description: Installer of game, infrequent user

Skill Level: High

Actor's Perspective on Use Cases			
Use Case	**Importance**	**Personality**	**Relative Frequency**
AGM002: Exit game	Secondary	Initiator	Low
AGM003: Save game (change case)	Secondary	Initiator	Low
AGM004: Save score (change case)	Secondary	Initiator	Low
AGM005: Check previous best score (change case)	Secondary	Initiator	Low
AGM006: Play Brickles	Secondary	Initiator	Low
AGM007: Play Pong	Secondary	Initiator	Low
AGM008: Play Bowling	Secondary	Initiator	Low
AGM011: Install game	Primary	Initiator	Low

---

5 Importance is primary or secondary.

6 Personality is initiator, server, receiver, or facilitator.

7 Relative Frequency is high, medium, or low.

AGM012: Uninstall game	Primary	Initiator	Low
Rank is primary or secondary. *Personality is initiator, server, receiver, or facilitator.* *Relative Frequency is high, medium, or low.*			

Concept of Operations

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about AGM products, see the Scope section.

The concept of operations (CONOPS) document describes how the product line will operate to accomplish its work. Product line organizations use this document to capture how the organization makes decisions and manages production.

Readership

The CONOPS document provides a road map of the decision-making process for the AGM product line organization. Managers use the road map to determine who is affected by a decision and who needs to be informed about specific decisions. Technical members use the CONOPS to determine who to contact for decisions. All product line personnel use the CONOPS as the guide to how to carry out any product line action.

Product Line Guiding Principles

The software product line strategy is intended to improve an organization's product production by planning and building a set of products as a group rather than treating each product separately. There are several fundamentals that affect how the product line organization should operate.

Plan a set of products as a unit. The software product line strategy is intended to improve an organization's product production by planning and building a set of products as a group rather than treating each product separately. The strategy is to analyze the similarities and differences between the products. The similarities allow developers to build assets that are used in all the products achieving very high levels of software reuse among the products. The variabilities allow developers to build assets that can be selected to support any of the identified variations among products.

Use development techniques that maximize reuse of all assets. AGM intends to achieve the high level of reuse through commonality analysis, explicit architectural support, and an organization that separates the functions of building pieces to be reused and using those pieces to build products.

Organize around roles. Figure 19 shows the structure the VPPD has designed for the product line organization. This CONOPS discusses how this new structure works.

Determine and enforce boundaries on which products are a part of the product line. When assets are created, they are created to apply to as many products in the product line as possible. The strategy is to analyze the commonalities and variabilities between the products. The commonalities allow developers to build assets that are used in all the products achieving very high levels of software reuse among the products. The variabilities allow developers to build assets that are sufficiently flexible to support the range of identified variations among products. The AGM product line scope document captures this information.

Develop a structure that achieves the qualities required in the products AGM produces. The software architecture for the product line is perhaps the most significant core asset. The architecture captures and exploits those commonalities and variabilities. By spanning the range of commonalities and variabilities, the architecture guides the core asset builders in structuring the assets to meet the needs of the product line.

Define a process for producing products that contributes to the achievement of the business goals of the product line. The production plan for a software product line provides an explicit guide to core asset builders about how assets should be structured to be of use to product builders. The production plan also provides a guide to the product builders of how to build products given the characteristics of the core assets. This plan makes clear the need to span all of the features required in any product that is within the scope of the product line as core asset development occurs.

Vision for This Product Line

AGM expects that using the software product line approach will allow us to gain market share by offering custom products quicker and at lower cost than the competition. In particular, we expect to be able to have entry level, lower cost, employees build products from the core assets. This will reduce the cost of each product. Eventually we expect to automate production fully so that marketing personnel can build products further lowering costs.

We hope to be able to serve smaller niche markets that are within the scope of the product line but that we have not served previously. We will be working with a novelty manufacturer to identify innovative placements of microchips that have potential for simple games.

Metrics for Success

The AGM product line will be a success if

- Production costs are reduced significantly (at least 25%),
- New products are moved from conception to delivery in significantly less time calendar time (at least 40% less), and
- The rate of new customer recruitment increases significantly (at least 20%).

Operations

AGM has created a new organization to manage the software product line. In this section we describe that organization and explain how the organization will operate.

Organizational Structure

The direct reports to the CEO of AGM, shown in Figure 19, remain the same. Each of these direct reports will have a role in governing the product line organization. Detailed role descriptions can be found in the Management Roles section.

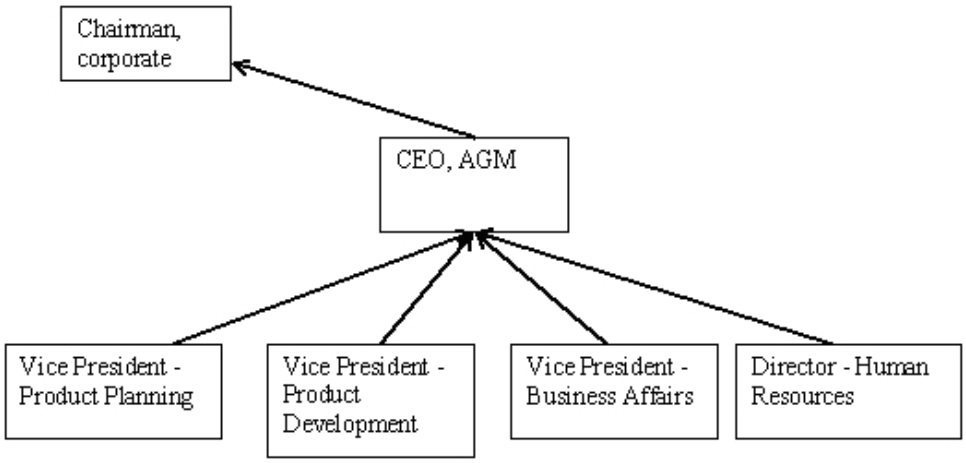

Figure 19: Corporate Structure

The structure of the product line organization is shown in Figure 20. The Vice President for Product Development has the main responsibility and authority for the product line organization. He has appointed the Product Line Manager (PLM), who reports directly to the VPPD.

In consultation with the VPPD, the PLM decided to structure the organization by splitting the roles of core asset building and product building. Initially there is a core asset team and a Brickles product building team. We will add additional teams as other product building projects are chartered. The org chart shows the likely structure for the organization after several product building teams are chartered.

AGM will continue to have the functional teams shown in the organizational chart. These teams report directly to the VPPD. The PLM negotiates with the functional team leads for resources each time a new project is chartered.

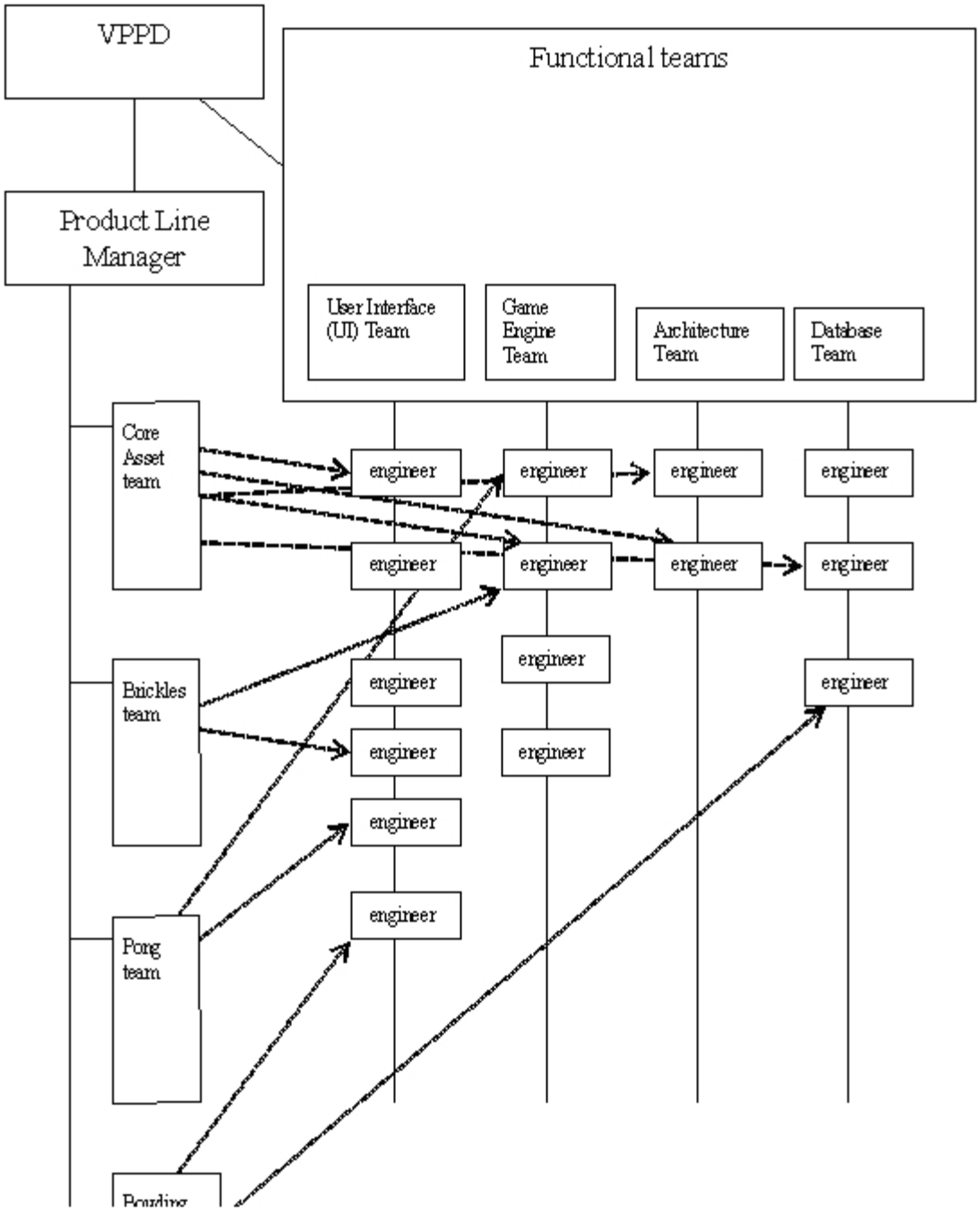

Figure 20: Organizational Structure

Communication

The organizational structure, shown in Figure 20, facilitates communication in a number of ways. Functional team members who are assigned to different building teams will share experiences at functional team meetings. This will cut across the core asset and product building teams providing an informal route for feedback from users of the core assets to builders of the core assets.

The production process, defined in the AGM production plan, also defines a formal avenue of communication between the core asset builders and the product builders. In this process, regular feedback opportunities are explicitly built into the process phases.

Other entities that facilitate communication include the Architecture Control Board (ACB) and the Configuration Change Board (CCB). The ACB owns the product line architecture and evaluates each request for a change to or deviation from the architecture. This board includes several members of the architecture functional team, the PLM, and the Core Asset team technical lead. The board meets as needed to evaluate requests. Periodically the ACB charters a project to refresh the architecture.

The CCB provides central supervision of the development threads. It ensures consistency among projects and ensures that each new product begins with the appropriate baseline. The CCB consists of the PLM and representatives from all the currently chartered projects including the continuing core asset project.

Operational Tools

The core assets are stored in a CVS repository. They are available 24 hours a day. The repository allows the selection of any version of an asset.

A website on the corporate intranet provides easy access to human-readable assets such as documentation plans and reports. The website has links to the CVS repository, tool download sites, and related gaming sites.

Beginning with the second increment of the product line, the development environment is the Eclipse extensible IDE. The community version of Omondo's UML Modeler is used to draw architectural and design models. The IDE has a build facility that associates the code assets needed to build a specific version of a specific product.

Core Asset Development

The core asset development team is chartered by the Product Line Manager (PLM) to produce high quality assets that will support the organization in achieving its goals.

Core Asset Building Roles

Since the core assets cover the breadth of resources needed to produce a product, the AGM Core Asset Team has a wide variety of roles. We list a few of those roles.

- Domain analyst - AGM is taking a domain-specific modeling approach to guiding the analysis and design of its products.

- Software architect - The architect defines structures that result in the desired levels of the specified qualities in the products.
- Process definer - Each core asset has an attached process that describes how to use the asset to build products. Personnel are trained in using the SPEM to define these processes.
- Software developer - Code-based assets are created in accordance with the modules defined in the software architecture.

Production Constraints

The members of the AGM product building team are knowledgeable programmers. Most program construction techniques are within their capabilities. There are no constraints on the product techniques that the core asset team can select for building core assets that the product building teams will not be able to handle.

However, AGM's goal is to increase the automation in product production so that costs are lower and the current product builders are available to join the core asset team. This does constrain the techniques somewhat. Using domain specific approaches so that core assets will later translate quickly into a domain specific language and automatic program generation is a good choice for the long-term success of the organization.

Implications of the Production Strategy

The production strategy guides the core asset developers by describing how the assets will be used to produce products. The core asset team must determine how best to satisfy both the production constraints and the production strategy as they select technologies, processes, and models to use in creating the core asset base.

The production strategy for AGM states:

We will produce the initial products using a traditional iterative, incremental development process using a standard programming language, IDE, and available libraries. We will create domain-based assets, including a product line architecture and software components, for the initial products in a manner that will support a migration to automatic generation of the second and third increment products.

The strategy calls for a migration from manual production to automatic production. The core asset team must plan how to quickly provide for manual production but then modify assets and develop tools to support automatic production.

This strategy implies that even at the first we should follow a domain-centric approach. By building a domain model before building the architecture and other assets, we will be able to use a number of new techniques for creating a domain-specific language. This language will be the foundation for our move to automated production.

Attached Processes

The core asset team attaches a process to each core asset that describes how to use that asset to build products. The attached process fits into the production plan of the product line. We intend to use a

production process that is described using the Software Process Engineering Meta-Model. Figure 21 shows a portion of that figure to illustrate how we have extended the model for our purposes.

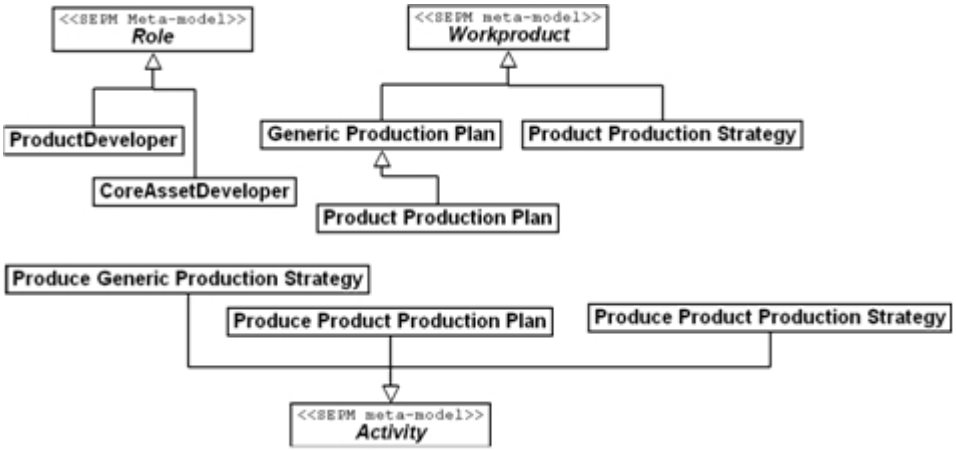

Figure 21: Portion of SPEM for Product Lines

Core Asset Development Scenarios

The following scenarios illustrate various situations regarding core asset development.

Core Asset Acquisition

AGM studies each asset to determine whether it makes sense to build it in house, use one of the legacy assets, commission the construction of the asset by an outside vendor, or purchase an existing product from a vendor to use as the asset. A study team is assigned to determine the best approach. The Product Line Manager assigns a technical manager to chair the group. Members of the core asset development team are assigned based on dependencies with the asset. A representative of purchasing is assigned, as a non-voting member, to support the team. The final decision is made by the Product Line Manager based on the study team's recommendation.

Core Asset Packaging and Delivery

The core asset development team assigned to develop an asset packages it for use and commits it to the core asset base. The packaging for the asset varies with the type and role of the asset. We have decided to use PDF as the packaging for all human-readable documents. The software assets being developed in the first increment will use dynamic link library packaging while assets in the second increment will be packaged in ZIP files. The AGM asset development standards call for specific types of documentation to accompany each asset. The asset package must be reviewed by a team consisting of at least two technical staff, a technical manager, and a technical communications specialist. Once issues identified in the review are resolved, the asset is released into the asset base.

Core Asset Sustainment

Each core asset is assigned an owner when it is placed in the asset base. The owner is usually one of the original developers. The owner tracks trends in the domain and uses of the asset in products. The

asset owner receives feedback from product development teams. The owner creates and maintains a sustainment plan which describes the evolution path of the asset. The asset owner periodically spends a portion of their time sustaining their assets. The exact amount of time available for this activity is determined by the owner's functional team leader in consultation with the PLM.

Product Development

The PLM, in response to a request from either the VPPP or the VPPD, examines a proposed product to determine whether it is within the scope of the product line. The PLM charters a product development project for any product that is added to the product line.

Product Building Roles

The product building teams share all of the roles listed in the Core Asset Building Roles section. In addition

- Product specifier - In the initial phases, products will be specified using features from the feature model prepared by the domain analyst. Eventually this role will be replaced by a less technical role in which a product is specified by selecting features from a list presented by a tool. The product will then be automatically generated.

Implications of the Production Strategy

The production strategy for AGM states:

We will produce the initial products using a traditional iterative, incremental development process using a standard programming language, IDE, and available libraries. We will create domain-based assets, including a product line architecture and software components, for the initial products in a manner that will support a migration to automatic generation of the second and third increment products.

The manner in which the product builders conduct their work is determined by how the core asset team decides to fulfill the production strategy. In the initial increments of the AGM product line, product builders will manually map requirements into the architecture to specific code modules. They will then compile and link code modules. The exact details of this are specified in the AGM Production Plan.

The production strategy will eventually change how the product builders build products and ultimately it will change the skill set required to be a product builder. As the core assets mature to the point they support automatic code generation, the product builders will only need to specify products. Programming skills and even understanding the architecture will be unnecessary.

Using the Core Assets

The AGM Core Asset Base is accessible to all AGM asset and product builders. The software components in the Base are indexed based on the product line architecture. The other assets are arranged based on the development process. The product builder identifies the type of asset and its role in the product and locates the asset.

Each asset is self-descriptive. The asset package includes a description of how to use the asset including pointers to the correct tools to manipulate the asset.

Each asset is associated with tools that product builders use to incorporate the asset into the production process. In increment one of the AGM product the dynamic linked library is packaged with the product deployment package. In increment two the class jar file is included in the product's self-extracting jar deployment package.

Modifying the Core Assets

There will be occasions when a core asset will not quite provide the functions required but deadlines will not allow a formal revision of the core asset. With the approval of the product team lead and the PLM, a product builder modifies the core asset to satisfy the required function. The modification is created as a branch in the version control system. The modification is communicated to the core asset owner. The core asset owner will consider whether this branch should remain separate or the branch should be merged in a new version of the asset.

Providing Feedback About Assets

Checking an asset out of the asset base immediately initiates the feedback request mechanism. This is a triggered event stored in the feedback database. The core asset builder will have specified an interval. At the end of that interval, a feedback request will be emailed to the product builder who checked out the asset.

The product builder will complete this request when they have completed using the asset.

Product Development Scenarios

Constructing a Product Using Only Assets

The AGM product building team assigned to build a product will use an abbreviated development process. (This process is defined by the process definition team and documented.) The team uses a word processor to edit the generic product specification. The team then uses the Eclipse IDE to create the product-specific driving loop. Finally, the product is tested on a standard set of test cases plus any product-specific test cases the team wishes to create.

Constructing a Product with Unique Requirements

The product building team assigned to build a product will use a complete development process. (This process is defined by the process definition team and documented.) The process uses an incremental approach. The effect of the additional requirements is first analyzed in the context of the existing product line requirements. This is done by modifying the game domain model. The team then plans the revisions to existing assets necessary to accommodate new concepts on the expanded domain model. Any new assets that are required are built, usually by deriving them from existing abstract assets. A member of the asset building team is assigned to assist the product builders in the analysis of how to acquire the required assets.

Management

Launch and Adoption Strategies

AGM is following a reactive product line adoption strategy. We are leveraging existing artifacts to expedite the rapid development of the core assets. This strategy allows us to bootstrap our new organization by modifying the existing assets. They are modified to accommodate a wider range of variation.

This strategy will only be successful if we have the discipline to conduct product line-wide analysis and planning activities. The scope and production plan documents and the software architecture must be completed beyond the initial product.

Management Roles

Four levels of management are involved in the AGM product line. Since these do not vary from one product to another we define them by position.

Vice Presidents

The Vice President for Product Development (VPPD) has primary responsibility for the AGM Product Line organization. He has authority over all personnel working on the product line. The Vice President for Product Planning (VPPP) has authority over the scope of the product line. He has final approval on all products assigned for development to the product line organization.

Product Line Manager

The Product Line Manager (PLM) reports to the VPPD with a dotted line relationship with the VPPP. The PLM has day-to-day operational responsibility for the product line operation and content. The PLM has designated authority over certain types of decisions as defined in this document.

Team Leader

The core asset and product builder team leaders report to the PLM. Each has responsibility for achieving the goals specified for their team. The core asset team leader is responsible for creating, managing, and maintaining the inventory of assets needed to build products. The product builder team leader is responsible for producing those products authorized by the PLM.

Project Lead

Each asset and product is developed in a project chartered by the responsible Team Leader. The Project Lead reports to the Team Leader who chartered the project. A project lead has responsibility for achieving the purpose stated in the project charter. The lead has the authority to remove a project member.

Organizational Structure, Roles, and Responsibilities

The AGM software product line organization is housed under the Vice President for Product Development (VPPD). The Product Line Manager (PLM) has responsibility for the product line organization

and reports to the VPPD. The organization has two teams: the core asset builders and the product builders. Each team has a lead who reports to the PLM.

The core asset team lead is responsible for establishing the production capability of the product line organization. This team produces assets such as the software architecture, the generic production plan, and the software components used to build product executables.

The product building team lead is responsible for operating the production capability of the product line organization. Each product is built by chartering a project staffed by members of this team and representatives from the appropriate functional teams.

AGM also has a system of functional teams. Each team provides a certain expertise to any product development project in the company. The functional team leads report to the VPPD. A functional team lead is responsible for sustaining the expertise of the team so that it is available to AGM as needed.

Challenges and Risks to Successful Implementation

Our products have been developed in isolated, stove-piped projects. Each team has had total autonomy. The primary challenge to successful implementation of the product line organization is developing a method of operation that stresses cooperation among projects.

A second challenge is developing a planning capability that is sufficiently robust to support the product line organization. We do not have a history of careful, comprehensive planning. Core assets are built in anticipation of being used in multiple products. The product line organization must be able to coordinate at least 3 product development efforts to be successful.

The primary risk to this effort is that the necessary practices will not be applied sufficiently well to achieve critical momentum. In that event, we may actually be in a worse situation than before trying the product line approach. We are mitigating this risk by studying successful cases of use of the product line approach and by conducting comprehensive planning exercises among the executive team.

Continuous Improvement

One of the goals of the AGM product line organization is to improve our operation over time. The production strategy defines one direction in which to improve. The intention is to mature the core assets to the point that products can be automatically generated from the assets. Data will be collected to guide operational changes.

- The number of modifications per asset per product will quantify how comprehensive the coverage of the core assets is. Capturing what those changes are will give core asset developers specific modifications to make to the core assets.

Management Scenarios

Add a Product to the Product Line

The Vice President for Product Planning (VPPP) initiates new products; however, that does not automatically add the product to the product line. At the request of the VPPP, the PLM charters a feasibility study. The study explores the proposed product in the context of the scope of the product line. The study team makes a recommendation to the PLM who, in consultation with the VPPD, determines whether to accept the new product into the product line or to reject the product.

Begin Development of a Product

In accordance with the agreed upon schedule, the PLM charters a product development project when appropriate. The PLM works with the functional team leads to staff the project.

Assessing Progress

The PLM is responsible to the VPPD for periodically assessing the progress of the product line organization. The PLM requests input from the functional team leads as to the impact of the product line on their teams. The PLM also requests input from each of the chartered project managers.

Interconnections

In this section, interconnections among the three basic areas of responsibility are described. Operational risks are also identified.

Core Assets and Product Building

The core asset team is responsible for providing the product building team high quality assets that are immediately usable. The product building team is responsible for providing the core asset team with feedback about the usefulness of the assets. The product builders also identify, and sometimes create the first generation of, new assets.

A database of deficiency reports (DRs) is maintained by the configuration management team. This tracking mechanism allows the reporter of a deficiency to monitor the resolution of the problem they have reported.

Core Assets and Management

Management is responsible for providing the core asset team the resources needed to carry out their charter. The VPPD provides a budget for the core asset team.

The team provides the VPPD with estimates of costs per asset based on long term tracking of development costs.

Product Production and Management

The VPPD provides the PLM with requirements for the products in the product line including delivery requirements. The PLM either charters a product building project in response to those requirements or rejects the product as being outside the scope of the product line.

Escalation

Issues between the core asset team and the product building teams that cannot be resolved are escalated to the PLM. Issues between the PLM and one of the teams may be further escalated to the VPPD.

Issues between the product line organization and the VPPP are resolved by the AGM CEO.

Problem resolution meetings are scheduled as needed. All resolutions are recorded in the operational procedures appendix of this documentation.

Risk Management

Risk management is everyone's responsibility. The PLM is responsible for maintaining a product line risk assessment. It is updated every time a product line progress report is made to the VPPD. Project leads are responsible for identifying risks to their specific projects. These are maintained as part of the project information.

The operational risks to the product line include

- Accepting a new game in to the product line that is not sufficiently in scope to reap the expected benefits from strategic reuse. This risk increases as the organization gains a reputation for producing products cheaply and quickly. This risk has been partially mitigated by the clear and easily understood scope statement provided in the AGM Product Line Scope document.
- Producing "core assets" that are not sufficiently encompassing to fit all the products. This risk increases in a reactive approach such as AGM is taking because the initial version of an asset may be viewed as the final version by management. This risk has been partially mitigated by having a clear transition plan associated with the AGM architecture.

Architecture

Architecture Documentation

Arcade Game Maker Architecture Documentation Beyond Views

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects and will be available on a variety of platforms. For more details about AGM products, see the Scope section.

This section describes the architecture used by AGM for its product line. It follows the format defined by Clements and colleagues [Clements 03] and supports the architectural model described in the Arcade Game Maker Software Architecture Views section.

The product line consists of nine products: three games (Brickles, Pong, and Bowling) on three platforms (Win32, Linux on wireless devices, and a platform-independent version). The games are graphical and depend heavily on the graphical nature of the environment. How this is handled across different environments will be a key architecture decision.

AGM System Overview

History

A chronology for the company's development efforts can be found in the Introduction.

The AGM product line began when the vice president for products was looking for techniques that would significantly speed up production. He found software product lines to be a promising practice. He assigned a small team to do a new technology study and, if possible, build a business case for adopting the product line approach (see the Business Case section).

The product line comprises three games: Brickles, Pong, and Bowling. Each is a single-player game in which the player scores or loses points by controlling a moving object.

Pong and Bowling are new. Brickles has been available since 1994 in various formats for various platforms.

Description

The games in the AGM product line all have a long history as computer programs. They are simple animations of a few items that interact with a larger number of stationary objects. The games run continuously until the player pauses or ends them.

Each game has a different set of rules. Some of the games have a simple won/lost scoring system, while others award points for certain actions.

Each game has a different set of moving and stationary items.

AGM Architecture View Packet Template

We adopted the view packet template presented by Clements and colleagues [Clements 03]. In this section, we briefly describe each part of it. The template is used in the Arcade Game Maker Software Architecture Views section to organize each view packet.

View Packet Template Description

- **Primary Presentation.** This presentation provides the basic model for this part of the architecture.

- **Element Catalog.** The element catalog includes a description of each element included in a view. The items that are "elements" in a particular view are the atomic items that characterize that view. For example, in the module view, the elements are modules.
  - **Properties of the elements.** 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.
  - **Relations and their properties.** 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.
  - **Element interfaces.** The element's interface shows its publicly available services using method signatures.
  - **Element behavior.** UML diagrams show the element's dynamic behavior. For example, behavior is shown with UML sequence and activity diagrams.
- **Context Diagram.** This presentation places the elements contained in a view packet in the overall architecture.
- **Variation Guide.** This section describes the variations for each element included in a view.
- Architecture Background. This section describes architectural decisions.
- **Rationale.** The rationale documents decisions that resulted in the architecture shown in the view.
  Analysis results. The analysis results provide the data that back up the decisions.
  Assumptions. The assumptions include data that underlie the decisions.
- **Other Information.** This section includes miscellaneous information that didn't fit into another section.

Related View Packets. This section describes the following types of related view packets:

- ones that surround the current view packet at the same level of detail
- ones that include the current view packet at a higher level
- ones that describe additional elements with the current view packet

Mapping Between Views

In general, we use UML to document the architecture. In UML, there are static diagrams that describe definitional units, such as classes, and dynamic diagrams that describe operational units, such as objects. Any operational unit that is used must correspond to a definitional unit. Therefore, there is a general mapping from dynamic diagrams to static diagrams.

Because the underlying development paradigm is object oriented, the architecture has both horizontal and vertical dimensions.

The vertical dimension corresponds to the specialization relation. There are several inheritance hierarchies, and the specialization relation is a major organizing principle for defining components within the product line.

The horizontal dimension corresponds to the association relation and forms the main operational structure of each product. Messages follow the association relation links and, in some cases, exceptions flow back over those links.

Rationale, Background, and Design Constraints

In this section, we document several architecture decisions that resulted in the current architecture.

Display and Data Separation

In an earlier version of the Brickles game, we used the model-view-controller (MVC) architecture. This approach allows for separation of the state of the system from the logic that presents some of that state to the player. The alternative we have considered is a more monolithic architecture that does not separate the data from their presentation.

Model-View-Controller (MVC)

Controllers provide input to the system. The input is routed to the view or the model as appropriate. In Brickles, the keyboard and mouse serve as controllers.

The views present information to the player in a variety of forms. In Brickles, the graphical interface has several fixed items and two movable ones, the puck and the paddle. The mouse controller allows the player to move the paddle, while the system determines the movement of the puck. The view has the responsibility to ask the model for the data it needs to build its presentation. Each view maintains a copy of the game state that is needed for its particular requirements and contains the graphical data to place the game data on the screen.

The model is the computation engine for the game. The model fuels the game by generating animation, recomputing the new location of the puck, receiving mouse events, and computing where to place the paddle. The model is obligated to inform all views when its state has changed.

The MVC pattern focuses on adding an arbitrary number of views to a given model. Tabular and graphical views of a data table might be useful, for example. The tradeoff is that a view updates itself by querying the model for the information it needs after the model has notified the view of a state change. The update operation is shown in a sequence diagram in Figure 22.

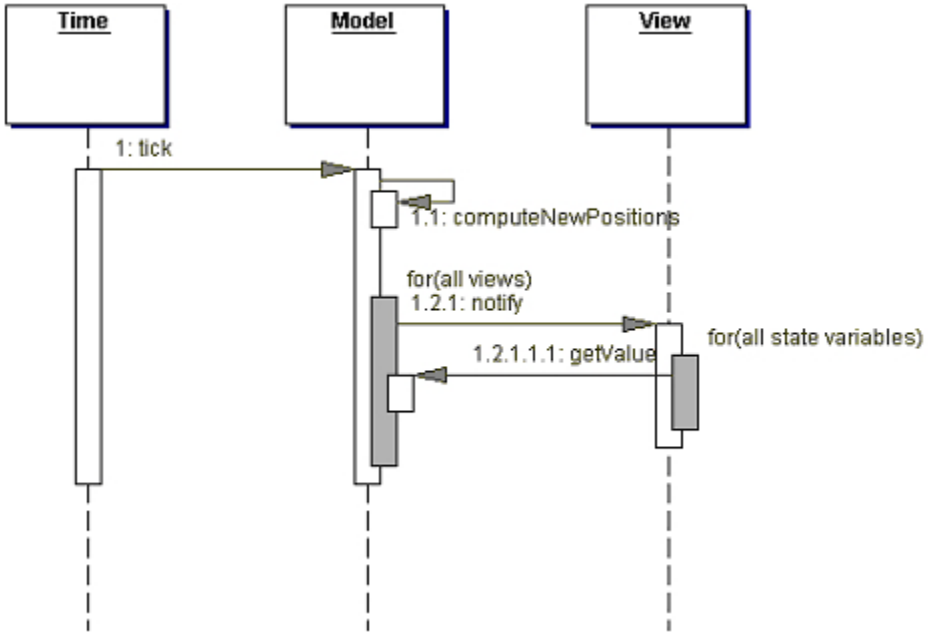

Figure 22: Model-View-Controller (MVC) Update Operation

An alternative architecture, part of which is shown in the next figure, was proposed as we started to develop the product line. In this alternative, the states of the game and the view are blended in various components. Knowledge of the graphics elements, which are usually platform independent, is distributed across the system's entities. Each object knows how to participate in the game, draw itself on the screen, and maintain its own data about the game and graphics. To update the display, each object is asked to paint itself to the screen. The container for all the game elements maintains overall control and sequences the screen updates.

The update of the screen operation is shown in Figure 23. This architecture is compared to the MVC architecture in Table 4.

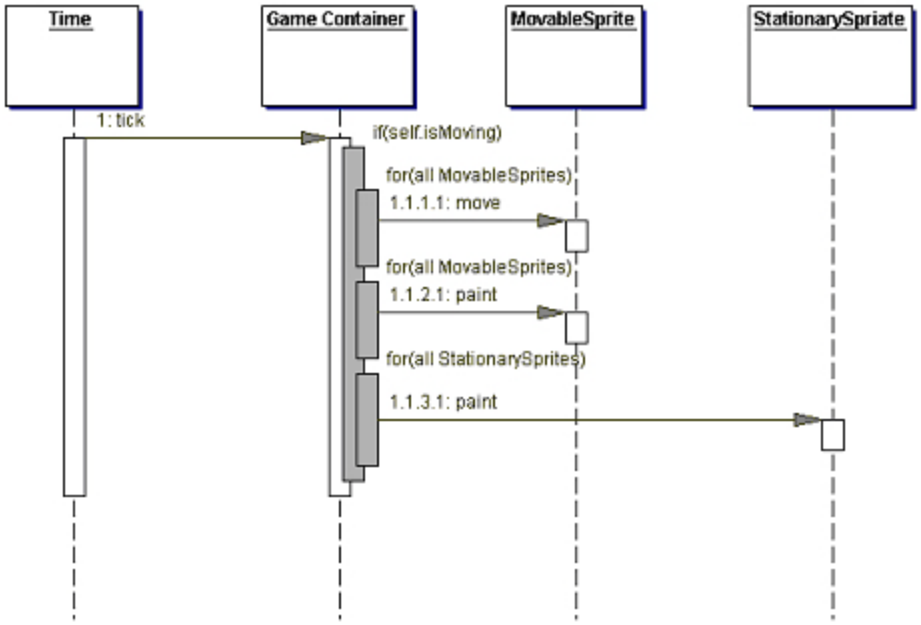

Figure 23: Monolithic Update Operation

Analysis

The advantages and disadvantages of each design are shown in Table 4.

Table 4: Comparison of MVC and Monolithic Approaches

	Advantages	**Disadvantages**
MVC	Multiple views can be added. Knowledge of the platform's graphical capabilities is isolated and straightforward to change.	Much messaging is needed at every update. State is replicated among all views and the model.
Monolithic	Additional views are difficult to add. State is in one place only.	No messaging is needed to update view. Knowledge of platform's graphical capabilities is distributed and thus harder to change for a new platform.

After reviewing the table, the team chose to use the monolithic version of the architecture. In particular, they rated performance as a high priority and adding views to an existing game as a very low priority.

Separating Moving and Non-Moving Items

A major distinction among game items is whether they move. By discriminating between moving and non-moving, the number of items that must be checked and updated every time is greatly reduced.

Generalization

The generalization hierarchy shown in Figure 24 shows one design for the various items that will participate in a game. Sprite is the most general game item. A Sprite has a location, a bounding box based

on that location, and a bitmap that is the visible representation of the element. Two additional items are specialized from Sprite: MovableSprite and StationarySprite. The MovableSprite item adds fields and methods that record whether the item is currently moving or not. The new item also adds the ability to receive clock ticks that drive the animation. Figure 25 shows additional items that are specialized from MovableSprite. Each new item defines its own move method so that each item can behave according to its role.

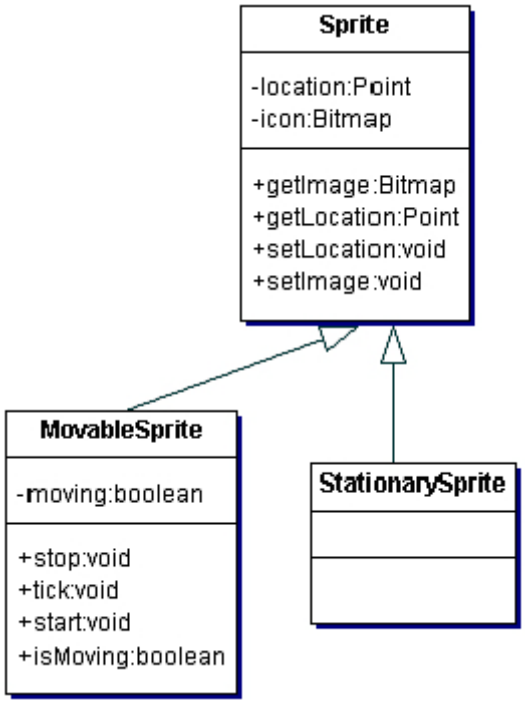

Figure 24: Top-Level Generalization Hierarchy

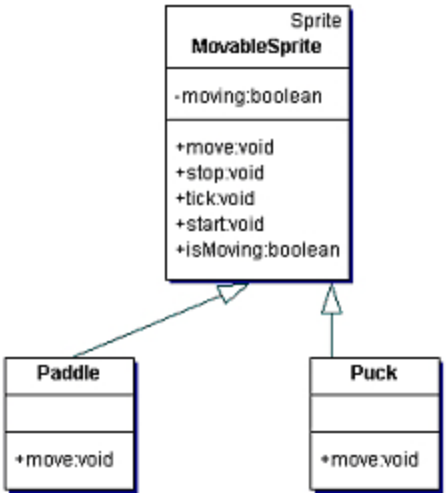

Figure 25: MovableSprite Specializations

An Alternative

Figure 26 illustrates a single item solution in which a single class has all the fields and methods that represent all the items' attributes. When an instance is constructed, parameters are provided to the constructor to make the item behave as needed. When a new "type" of item is needed, this one class is modified to provide the new behavior. One of the parameters provided to the ParameterizedSprite is an instance of a movement algorithm. For moving items, the ParameterizedSprite delegates the movement to this object.

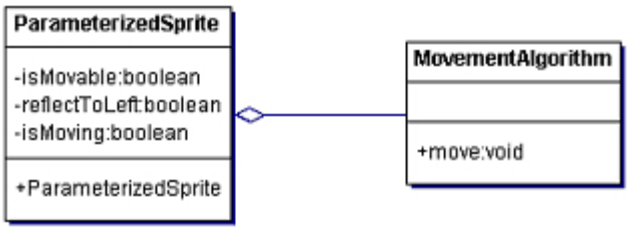

Figure 26: ParameterizedSprite Class

Analysis

The advantages and disadvantages of each design are shown in Table 5.

Table 5: Comparison of Generalization/Specialization and Parameters Approaches

	Advantages	**Disadvantages**
Generalization	Each item contains just what it needs for its definition.	New item types must be defined at compile time.
Specialization	The types are related polymorphically, which allows flexible architectures.	Changes to more general items cause more specific items to change.
Parameters	New types of items can be made dynamically if the parameter objects are already defined.	Each item is the size of all the items combined. Different items cannot be distinguished automatically.

The team decided to use a combination of these approaches. The GameBoard container is instantiated by passing a number of parameters, and individual game items are defined by specializing existing abstract definitions.

Product Usability

Because games must be enjoyable for the players, their graphical design is constrained by usability studies and guidelines. Since three of the products are to be sited on wireless devices, we will use guidelines from that domain. Two that we are following closely are found at the following websites:

- http://www.forum.nokia.com/series40_game_usability_study
- http://www.forum.nokia.com/info/sw.nokia.com/id/d9d9ad4a-efd8-42bd-ba70-dcc35bab9422/Series_40_DP_1_0_Usability_Guidelines_For_J2ME_Games_v1_2_en.pdf.html

Attached Processes

Process for Modifying the Architecture

There are several reasons for architecture changes. Figure 27 describes a process that can be used for any architecture change, but we describe it using a requirements change.

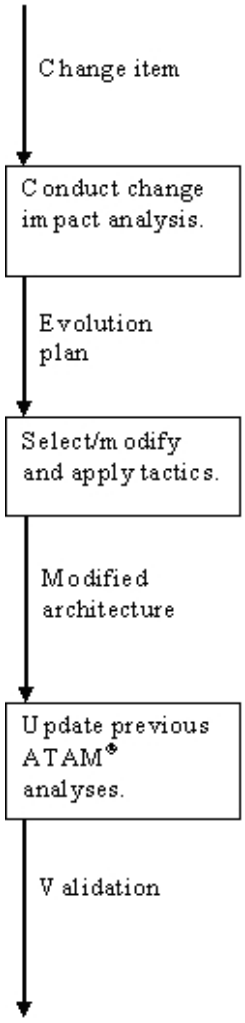

Figure 27: Attached Process for Requirements Change

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 (i.e., within the context provided by the GameBoard).

Process for Using the Architecture to Produce a Product

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

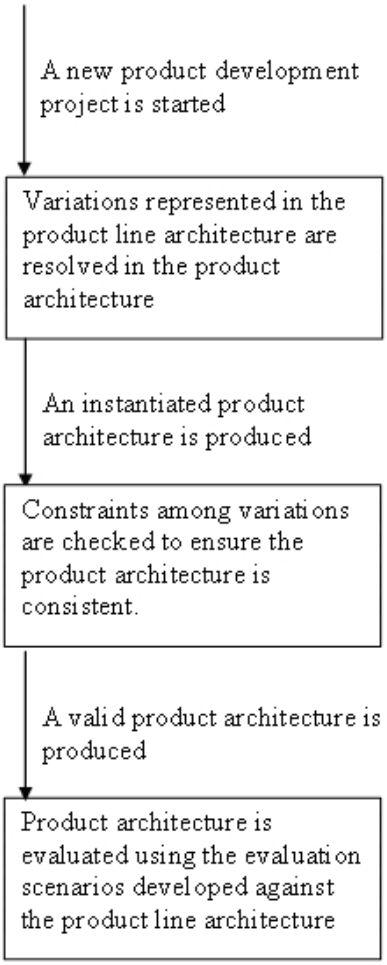

Figure 28: Production Plan

Arcade Game Maker Software Architecture Views

The Arcade Game Maker (AGM) product line is a set of relatively simple systems. This version of the software architecture is simple as well, but it may become more complex as change cases in the requirements model are planned and implemented.

The architecture attains the qualities prescribed for it in the Requirements section: that is, the games are fast and realistic.

This second architecture volume provides detailed models of the games' architectural models. For background information that supports these models, see the Arcade Game Maker Architecture Documentation Beyond Views section.

Allocation Deployment View

Allocation Deployment View Packet 1: Game

As shown in Figure 29, we intend to deploy all products in the AGM product line on a single processor. Each game 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 Game's software architecture.

Primary Presentation

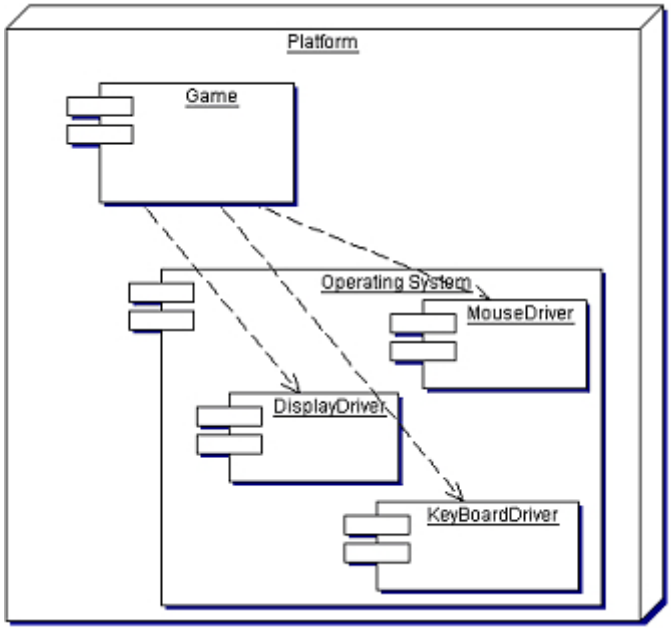

Figure 29: Process Allocation for AGM Games

Game responds to mouse and keyboard events and controls the display through calls provided in the language library.

Element Catalog

1. **Elements and their properties.** Table 6 displays element details.

Table 6: Elements and Responsibilities for Allocation Deployment View Packet 1: Game

Element	Responsibilities
Game	This is the generic representation for an AGM game. Each game will have a specialization of this class as its top level.
Operating System	The usual operating system interface provides drivers for the devices that are needed by the game.

2. **Relations and their properties.** The relationship in this view is dependency. The Game has dependencies on three drivers in the operating system that are realized by calls on library methods.
3. **Element interfaces.** Game's interface is a GUI that provides the basic start/exit/pause behavior.

4. **Element behavior.** Game is the top level of each AGM game. The general behavior is an action loop that runs until the player exits the game or the game is over. Game-specific behavior is provided by the specialized Game class.

Context Diagram

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

Variation Guide

Game is the generic product. For details, see the Module Generalization View Packet 1: Game section.

Architecture Background

Game provides the basic behavior for any arcade game. The specializations add details to provide the specifics of a game.

Other Information

No other information applies.

Related View Packets

- Game internals are provided in the Module Generalization View Packet 1: Game section.
- Specializations of the generalization are provided in the Module Generalization View Packet 1: Game section.

Module Decomposition View

In this section, we describe the basic structure of an AGM game. Game variations are addressed by substitution, parameterization, and specialization. Game-specific behaviors are provided by game-specific implementations of the interfaces in this document.

Module Decomposition View Packet 1: Game

Figure 29 shows the Game component as the representation for the generic product. Figure 30 shows that component's interface as the top-level interface of the system (defined in the GameBoard Interface section) and the other major interfaces that are at the first level of decomposition within the GameInterface (defined later in this section).

Primary Presentation

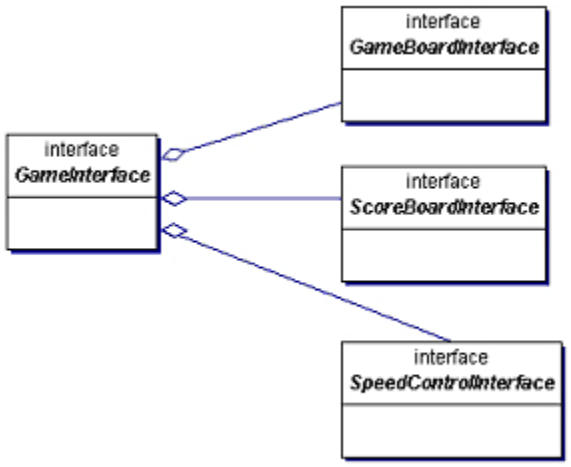

Figure 30: Module View Decomposition of Game Component

Element Catalog

1. **Elements and their properties.** Table 7 displays element details.

Table 7: Elements and Responsibilities for Model Decomposition View Packet 1: Game

Element	Responsibilities
GameBoard interface	This container component holds all the elements needed for the game.
ScoreBoard interface	This interface keeps and presents the score as the game specifies. It is defined in the Score-Board Interface section.
SpeedControl interface	This interface controls how often a tick is issued to the GameBoard. It is defined in the Speed-Control Interface section.

2. **Relations and their properties.** The primary relation is composition. Game is responsible for creating, managing, and killing the elements it composes.
3. **Element interfaces.** Game interface is the program's GUI. It provides the user with Game start, stop, pause, and save behaviors. The other interfaces are documented as follows:
   - GameBoard Interface
   - SpeedControl Interface
   - ScoreBoard Interface
4. **Element behavior.** The behavior is the game that is displayed to the user.

Context Diagram

Game is the top level of the product and the top-level context.

Variation Guide

Game and ScoreBoard will each be replaced by a game-specific implementation as described in the Module Generalization View section.

Architecture Background
Game encapsulates game-specific behavior. It arranges the GameBoard, keeps score, and determines the won/lost status based on its rules. The composed elements are mostly generic and can be reused and rearranged to implement other games.

Other Information
No other information applies.

Related View Packets

- Module Generalization View Packet 1: Game
- ScoreBoard Interface

Module Decomposition View Packet 2: GameBoard

This view shows the class dependencies for the GameBoard class. Figure 31 shows the related classes.

Primary Presentation

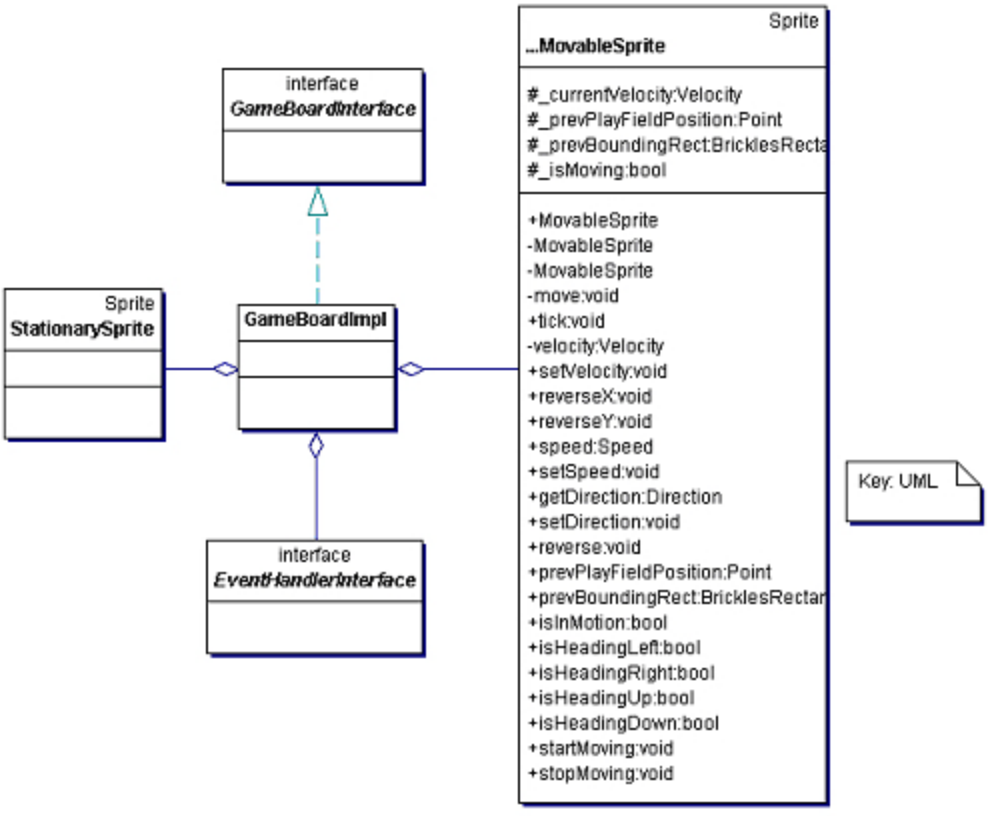

Figure 31: GameBoard Module Decomposition

Element Catalog

1. **Elements and their properties.** Table 8 displays element details.

Table 8: Elements and Responsibilities for Model Decomposition View Packet 2: GameBoard

Element	Responsibilities
EventHandlerInterface	This interface defines a standard interface through which events are passed from the interaction devices.
StationarySprite	This is an abstract class for many items that will be placed on the GameBoard and hit by moving objects. Item behavior varies from reflecting to absorbing moving objects.
MovableSprite	This is an abstract class for many items that will be placed on the GameBoard and moved according to some algorithm. Item behavior varies from perfect reflective bouncing to random movement.

2. **Relations and their properties.** The primary relation is composition. GameBoard composes the elements that make the game behavior.
3. Element interfaces
   - GameBoard Interface
   - Sprite Interface
4. **Element behavior.** No behavior applies.

Context Diagram
GameBoard is shown in context in Figure 30, the Module Decomposition View.

Variation Guide
GameBoard is a container. Game adds the desired MovableSprites and StationarySprites to the GameBoard. The GameBoard can take an arbitrary number of Sprites chosen from the classes shown in Figure 34 and Figure 37, showing the MovableSprite Interface and StationarySprite Interface.

Architecture Background
GameBoard is designed as a component container. It delivers events to the components it holds. For example, it delivers timer ticks to its MovableSprite objects so that they can move.

Other Information
No other information applies.

Related View Packets
Sprites are defined in the Module Generalization View Packet 2: Sprite section.

Module Generalization View

Module Generalization View Packet 1: Game

This view shows the specialization of Game by three classes – shown in Figure 32.

Primary Presentation

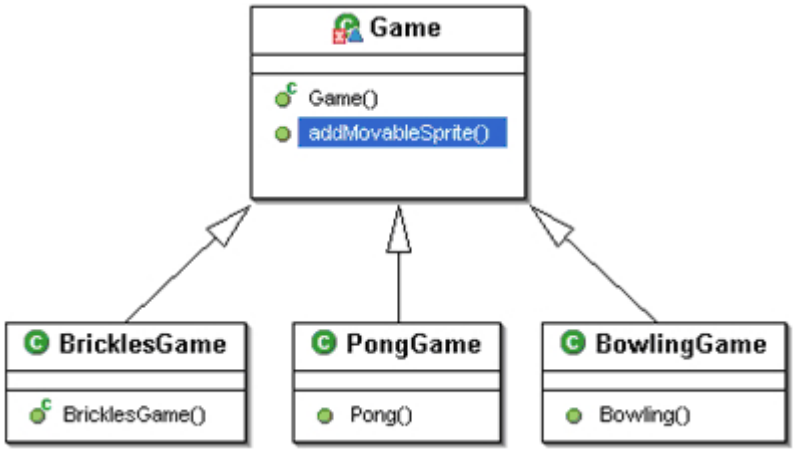

Figure 32: Game Generalization

Element Catalog

1. **Elements and their properties.** Table 9 displays element details.

Table 9: Elements and Responsibilities for Module Generalization View Packet 1: Game

Element	Responsibilities
Game	This abstract class defines game functionality that is common to all products.
BricklesGame	This class adds Brickles-specific behavior.
PongGame	This class adds Pong-specific behavior.
BowlingGame	This class adds Bowling-specific behavior.

2. **Relations and their properties.** The relation is generalization/specialization.
3. **Element interface.** All elements in this view implement the Game interface.
4. **Element behavior.** No behavior applies.

Context Diagram

Game is shown in context in Figure 30.

Variation Guide

No variation within each element.

Architecture Background

Each Game class inherits the abstract Game class, including the basic simulation loop that is a protected method in the abstract class.

Other Information

No other information applies.

Related View Packets

No packets apply.

Module Generalization View Packet 2: Sprite

This view shows the classes that specialize the Sprite class (see Figure 33).

Primary Presentation

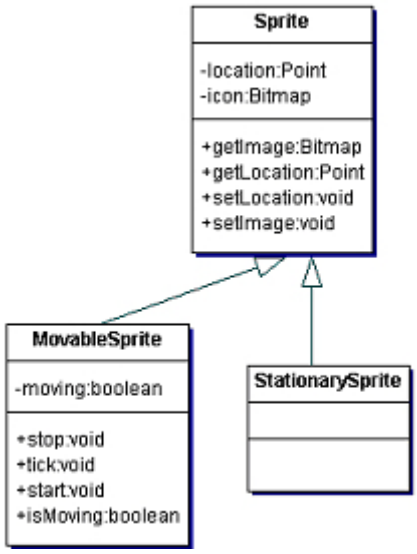

Figure 33: Sprite Basic Classes

Element Catalog

1. **Elements and their properties.** Table 10 displays element details.

Table 10: Elements and Responsibilities for Module Generalization View Packet 2: Sprite

Element	Responsibilities
Sprite	This abstract class defines the generic behavior and state for any element on the screen including position and bitmap.
MovableSprite	This class defines the moving behavior of screen elements and adds it to the general behavior.
StationarySprite	This class defines the obstacle behavior of screen elements and adds it to the general behavior.

2. **Relations and their properties.** The relation is generalization/specialization.
3. **Element interfaces.** No interfaces apply.
4. **Element behavior.** No behavior applies.

Context Diagram

The two specializations are shown in Figure 31, the GameBoard Module Decomposition.

Variation Guide
No variation applies.

Architecture Background
The abstract class is used for two purposes: (1) as a common type for defining parameters; and (2) to define common behavior that will be shared by a number of subclasses.

Other Information
No other information applies.

Related View Packets

- Module Generalization View Packet 3: MovableSprite
- Module Generalization View Packet 4: StationarySprite

Module Generalization View Packet 3: MovableSprite

This view shows the classes that specialize MovableSprite (see Figure 34).

Primary Presentation

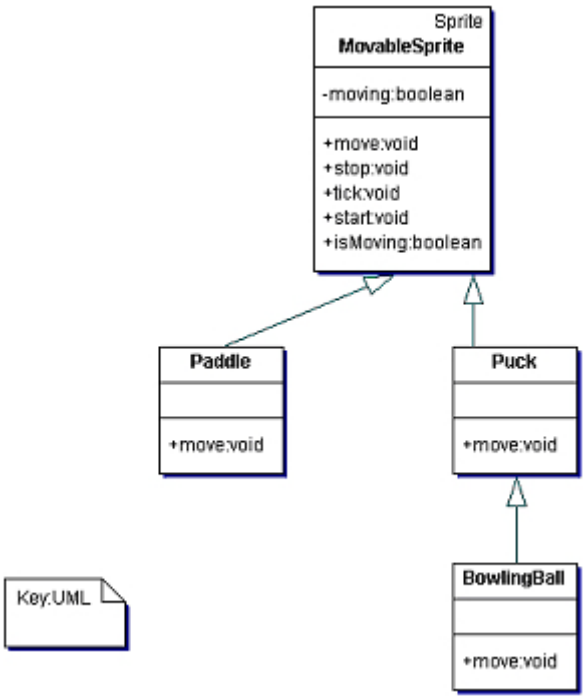

Figure 34: MovableSprite Specialization View

Element Catalog

1. **Elements and their properties.** Table 11 displays element details.

Table 11: Elements and Responsibilities for Module Generalization View Packet 3: MovableSprite

Element	Responsibilities
MovableSprite	This class defines the moving behavior of screen elements and adds it to the general behavior.
Paddle	This class adds the constrained movement of a Sprite that travels horizontally only.
Puck	This class adds the moving behavior of a bouncing ball in a gravity field.
BowlingBall	This class adds the behavior of a ball being thrown with various spin values.

2. **Relations and their properties.** The relation is generalization/specialization.
3. **Element interfaces.**
   - MovableSprite Interface
   - StationarySprite Interface
4. **Element behavior.** The principle elements that move are paddles, pucks, and bowling balls. They may be further specialized for other games.

Figure 35 presents the abstract algorithm for moving Sprites around the GameBoard.

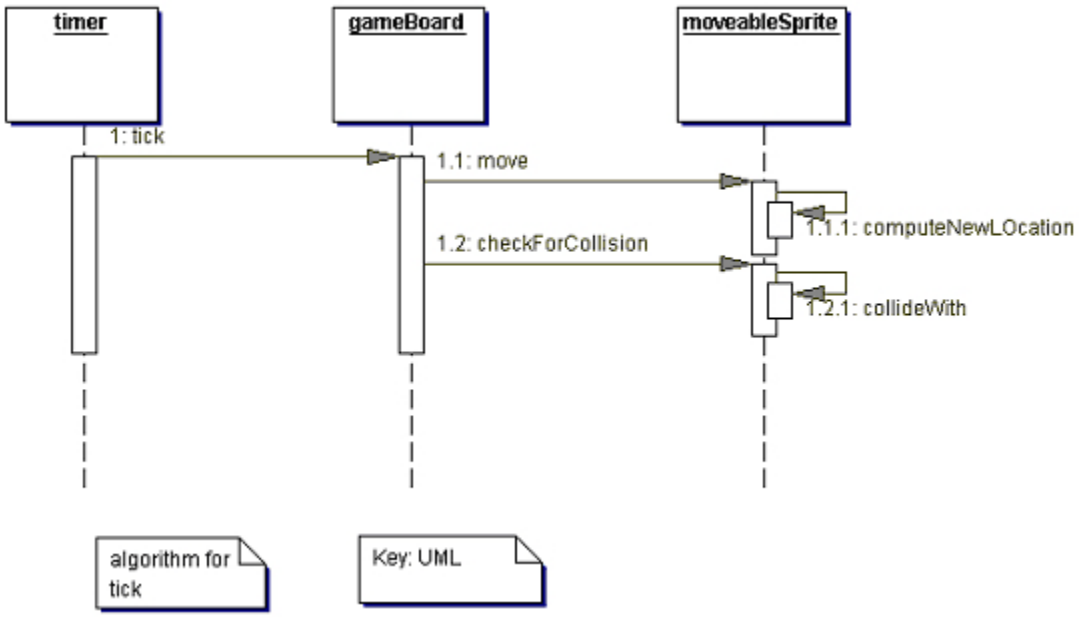

Figure 35: Basic Move Algorithm for MovableSprites

Context Diagram

Figure 36 shows the context within which MovableSprite is defined.

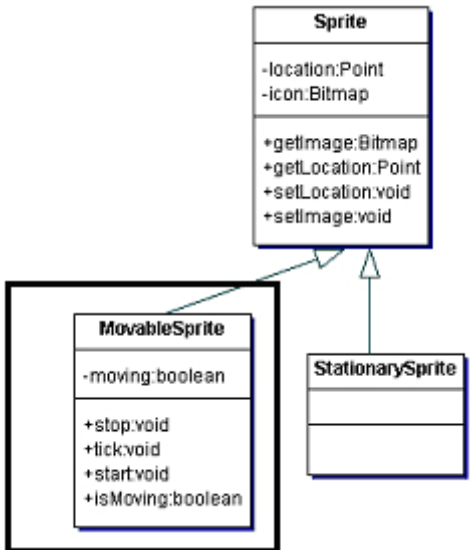

Figure 36: Context for MovableSprite View Packet

Variation Guide
The variation is in which class to select. Individual classes only vary by the MovableSprite's initial location on the playing field.

Architecture Background
Specializations are added to this structure because MovableSprites with different behaviors from those already defined are needed to populate a game.

Other Information
No other information applies.

Related View Packets
No packets apply.

Module Generalization View Packet 4: StationarySprite

This view describes the classes that specialize StationarySprite (see Figure 37).

Primary Presentation

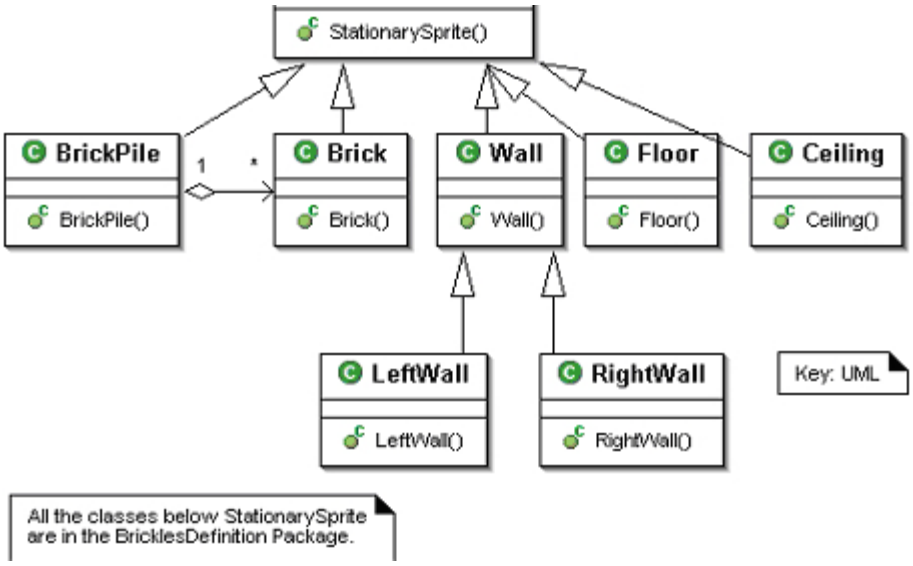

Figure 37: StationarySprite Classes

Element Catalog

1. **Elements and their properties.** Table 12 displays element details.

Table 12: Elements and Responsibilities for Module Generalization View Packet 4: StationarySprite

Element	Responsibilities
StationarySprite	This class defines the obstacle behavior of screen elements and adds it to the general behavior.
BrickPile	This class is a container that holds Bricks.
Brick	This class is an obstacle that breaks when hit by a MovableSprite.
Wall	This class reflects MovableSprites according to Newtonian physics.
LeftWall	This specialization of Wall knows that it is to the left of the board.
RightWall	This specialization of Wall knows that it is to the right of the board.
Floor	This class is an obstacle that absorbs MovableSprites.
Ceiling	This class reflects MovableSprites.

2. **Relations and their properties.** The relation is generalization/specialization.
3. **Element interfaces.** These elements all implement the StationarySprite interface defined in the StationarySprite Interface section.
4. **Element behavior.** Figure 38 shows the role of the StationarySprites in detecting collisions. After each tick, the GameBoard updates the position of each MovableSprite to its new location and

then compares the BoundingBox of each MovableSprite with the BoundingBox of every StationarySprite. When an overlap occurs, a collision has been detected and a CollisionException is thrown.

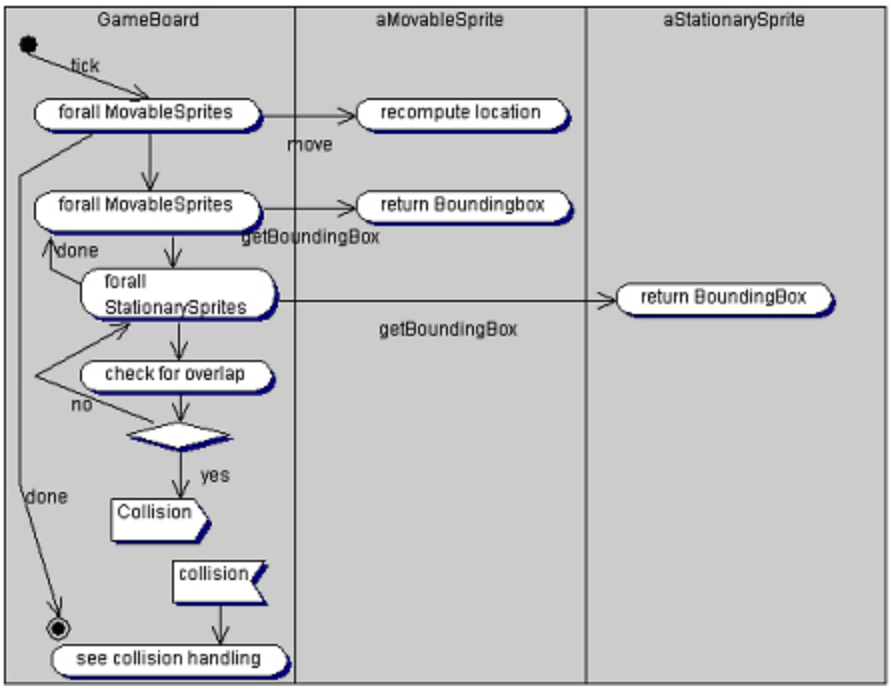

Figure 38: Collision Detection Algorithm

Figure 39 shows how a collision is handled. The CollisionException is caught, and a message is sent to the StationarySprite involved in the collision. The StationarySprite follows its behavior and sends the appropriate message to the MovableSprite.

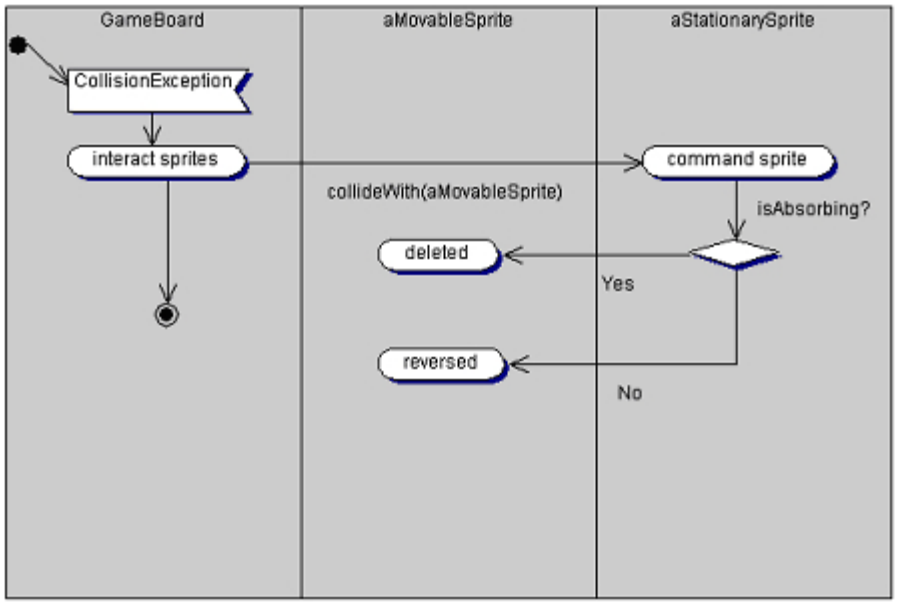

Figure 39: Handling a Collision

Context Diagram

Figure 40 shows the context within which StationarySprite is defined.

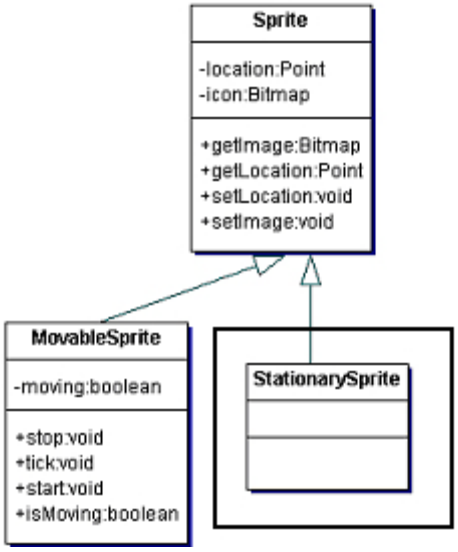

Figure 40: Context for StationarySprite View Packet

Variation Guide

The variation is in which class to select. Individual classes only vary by the MovableSprite's initial location on the playing field.

Architecture Background

This is the design that adds the most value.

Other Information
No other information applies.

Related View Packets
No other packets apply.

Component-and-Connector View

In this section, the Allocation Deployment View and Module Decomposition View are supplemented with an operational view (see Figure 41) using a component-and-connector view type. This view provides a look at the computation flow through the system.

Component-and-Connector View Packet 1: Control Loop

The game-specific implementation of the Game interface provides the main control loop that drives the game. As such, it must be implemented for each specific game.

Primary Presentation

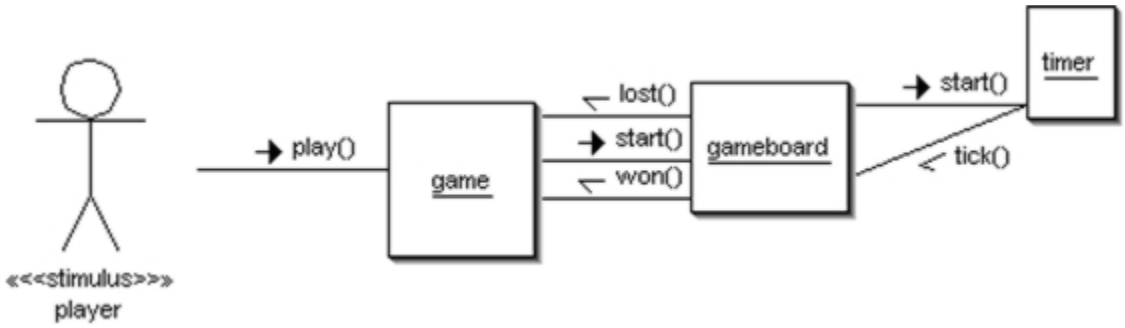

Figure 41: Control Loop

Element Catalog

1. Elements and their properties. Table 13 displays element details.

Table 13: Elements and Responsibilities for Component-and-Connector View Packet 1: Control Loop

Element	Responsibilities
Player	The player of the game is responsible for initiating play.
Game	Game provides the user interface and the game-specific control loop and calls start on the GameBoard.
GameBoard	The GameBoard controls the timer and responds to its ticks until the game is stopped, paused, or completed.
Timer	The timer delivers ticks at specified intervals that correspond to the setting of the SpeedControl.

2. **Relations and their properties.** The relation shown in Figure 41 is method invocation. Each arrow represents a synchronous or asynchronous invocation of a method on the called object.
3. **Element interfaces.**

- Allocation Deployment View
- GameBoard Interface

4. **Element behavior.** No behavior applies.

Context Diagram

This is the top-level system as shown in Figure 29.

Variation Guide

The variation in this scenario comes when a game-specific variant of Game is substituted. This diagram does not change, but the conditions under which the won and lost messages are sent do change.

Architecture Background

Separating the timer from the GameBoard allows the speed of the game to be changed very easily. The timer controls movement on the screen and is isolated from the game to provide autonomy of the GameBoard.

Other Information

No other information applies.

Related View Packets

No other packets apply.

GameBoard Interface

Interface Identity

GameBoard

Resources Provided

Resource Syntax

Board(Point p, Size s, EventHandlerDefinitions ehd)
void startMovement()
void stopMovement()
void setSpeed(int newValue)
int getSpeed()
void tick()
void addMovablePiece(IComponent ic, String s)
void addMovablePiece(IComponent ic)
void removeMovablePiece(IComponent ic)
VOID ADDSTATIONARYPIECE(ICOMPONENT IC, STRING S)
void addStationaryPiece(IComponent ic)
VOID REMOVESTATIONARYPIECE(ICOMPONENT IC)

void resetList()
boolean isMember(Icomponent ic)
boolean isMoving()

Resource Semantics

Board(Point p, Size s,EventHandlerDefinitions ehd)

Pre: true
void startMovement()
Post: self.isMoving = true

Pre: true
void stopMovement()
post: isMoving = false

pre: true
void setSpeed(int newValue)
post: self.getSpeed() = newValue

pre: true
int getSpeed()
post: return speed

pre: true
void tick()
post: none

pre: true
void addMovablePiece(IComponent ic, String s)
post: self.isMember(ic) = true

pre: true
void addMovablePiece(IComponent ic)
post: self.isMember(ic) = true

pre: self.isMember(ic)
void removeMovablePiece(IComponent ic)
post: self.isMember(ic) = false

pre: true
void addStationaryPiece(IComponent ic, String s)
post: self.isMember(ic) = true

pre: true
void addStationaryPiece(IComponent ic)
post: self.isMember(ic) = true

pre: self.isMember(ic)
VOID REMOVESTATIONARYPIECE(ICOMPONENT IC)
post: self.isMember(ic) = false

pre: true
void resetList()
post: forall ic.isMember(IComponent) self.isMember(ic)=false

pre: true
Boolean isMember(Icomponent ic)
Post: return true if ic is in Board

Pre: true
Boolean isMoving()
Post: return true if this is in moving state

Resource Usage Restrictions

- Only one instance of GameBoard may be present in a system.
- This component has no dependencies on the physical rules or the game rules.

Locally Defined Data Types

- EventHandler Definitions
- MovableSprite Interface
- StationarySprite Interface

Error Handling

Sprites may throw exceptions that are caught by the game instead of the GameBoard implementation. They are *not* errors. They are an integral part of the logic of the game being played.

Variation Provided

The main variation in GameBoard is the set of event handlers. They are defined in a single object that is provided as a parameter to the constructor. Definitions cover all mouse and keyboard events.

Quality Attribute Characteristics

GameBoard is the basis for the animation. It provides an implementation that results in the smooth movement of the moving Sprites. “Smooth movement” is defined as the game piece changing locations without unexpected pauses caused by the refresh rate of the graphics as opposed to the movement of the mouse or other controller.

Element Requirements

GameBoard requires the following elements from the .Net library:

- System.ComponentModel.Container
- System.ComponentModel.IComponent

The container holds the IComponents. Its list facilities manipulate the Sprites.

Rationale and Design Issues

The GameBoard interface provides a single point to which games come to place elements on the GameBoard that is shown to the player. The interface separates the creation of the GameBoard object from the configuration process.

Usage Guide

The rules of the game are distributed among the game elements (the Sprites). Because the GameBoard is a container for the action, there are few constraints on the use of the container. The state machine for the GameBoard is shown in Figure 42.

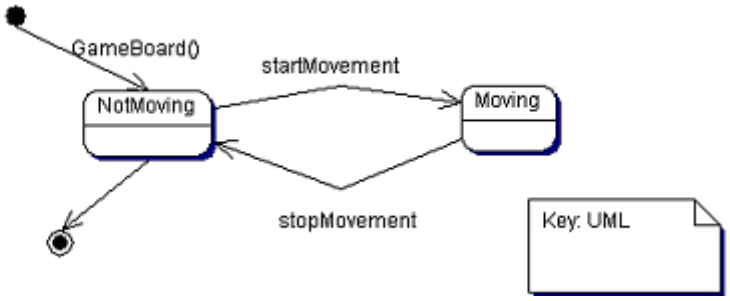

Figure 42: State Machine for the GameBoard

Usage of the GameBoard

The state machine is purposely simple and does not preclude future games from adding Sprites while the game is in progress (not applicable in current games).

Sprite Interface

Interface Identity

Sprite

Resources Provided

Resource Syntax

```
Sprite(Point p, Size s)
void collideWith(Sprite m);
void paint(PaintEventArgs e);
void setBoundingBox();
```

PictureBox getPicture();
Rectangle getBoundingBox();
Rectangle getBoundingBox(Sprite s);
bool overLaps(Sprite s);
bool overLaps(Rectangle r);

Resource Semantics

Pre: true
Sprite(Point p, Size s)
Post: self exists

Pre: true
void collideWith(Sprite m);
post:

pre: true
void paint(PaintEventArgs e);
POST: CURRENT STATE OF SELF IS REPRESENTED ON SCREEN

pre: true
void setBoundingBox();
post: self's bounding box is updated

pre: true
PictureBox getPicture();
Post: return the bounding box of image

Pre: true
Rectangle getBoundingBox();
Post: return the bounding box of Sprite

Pre:true
Rectangle getBoundingBox(Sprite s);
Post: return self.boundingBox

Pre: true
bool overLaps(Sprite s);
post: return true if s.getBoundingBox().overlaps(self.getBoundingBox())

pre: true
bool overLaps(Rectangle r);
post: return true if self.getBoundingBox().overlaps(r))

Resource Usage Restrictions

This interface specifies abstract concepts. It is not intended to be used.

Locally Defined Data Types

No types apply.

Error Handling

No error handling applies.

Variation Provided

No variation applies.

Quality Attribute Characteristics

No specific qualities apply. Sprite is too abstract.

Element Requirements

A Rectangle element must have been defined.

Rationale and Design Issues

This definition fills an important domain role. It represents the Sprite concept, which has a long history in computer-based games in our design.

Usage Guide

Sprite is used as the basis for concrete component definitions.

MovableSprite Interface

Interface Identity

MovableSprite (specialization of previous section: Sprite Interface)

Resources Provided

Resource Syntax

MovableSprite(Point p, Size s)
void paint(PaintEventArgs e)
bool moving()
void startMoving()
void stopMoving()
void reverseX()
void reverseY()
void move()

Resource Semantics

Pre: true
MovableSprite(Point p, Size s)
Post: self exists

Pre: true
void paint(PaintEventArgs e)
post: current state of self is represented on screen

pre: true
bool moving()
post: return true if self is moving otherwise false

pre: true
void startMoving()
post: self.moving = true

pre: true
void stopMoving()
post: self.moving = false

pre: true
void reverseX()
post: X component of direction is reversed

pre: true
void reverseY()
post: Y component of direction is reversed

pre: self.moving = true
void move()
post: self has moved to new Location

Resource Usage Restrictions

A MovableSprite can be used anywhere a Sprite or MovableSprite is specified.

Locally Defined Data Types

No types apply.

Error Handling

No error handling applies.

Variation Provided

The types of MovableSprites differ from each other in three ways:

1. Each has its own icon on the screen.

2. Each provides its own behavior when it is involved in a collision.
3. Each provides its own algorithm that controls the path it follows as it moves.

Quality Attribute Characteristics

The division between stationary and movable Sprites is a performance enhancement effort. That is, by only having some of the game pieces recomputed a new location on each pulse of the game, there is less overhead for the animation.

Element Requirements

No additional state variables, beyond those in Sprite, are required.

Rationale and Design Issues

Stationary and movable Sprites are treated separately for efficiency. In most games, players score points when stationary and moving Sprites collide. Because collision checking occurs after each movement, this is a significant point at which to enhance performance. By dividing Sprites into two stationary and movable groups, collision checking takes less time.

Usage Guide

MovableSprites are the main actors in a game. They provide the action that keeps the player's attention. MovableSprites are added to the GameBoard at product creation time by the product builder.

StationarySprite Interface

Interface Identity

StationarySprite (specialization of StationarySprite Interface's Sprite)

Resources Provided

Resource Syntax

StationarySprite(Point p, Size s)

Resource Semantics

Pre: true
StationarySprite(Point p, Size s)
POST: SELF EXISTS

Resource Usage Restrictions

A StationarySprite can be used anywhere a Sprite or StationarySprite is specified.

Locally Defined Data Types

No types apply.

Error Handling

No error handling applies.

Variation Provided

Each StationarySprite can be instantiated as absorbing or non-absorbing. When an absorbing StationarySprite collides with a MovableSprite, the MovableSprite is deleted from the game.

Quality Attribute Characteristics

The division between stationary and movable Sprites is a performance enhancement effort. That is, by only having some of the game pieces recomputed a new location on each pulse of the game, there is less overhead for the animation.

Element Requirements

No additional state variables, beyond those in Sprite, are required.

Rationale and Design Issues

Stationary and movable Sprites are treated separately for efficiency. In most games, players score points when stationary and moving Sprites collide. Because collision checking occurs after each movement, this is a significant point at which to enhance performance. By dividing Sprites into two stationary and movable groups, collision checking takes less time.

Usage Guide

StationarySprites form a game's boundaries and obstacles in a game and are added to a GameBoard as the game is constructed.

ScoreBoard Interface

Interface Identity

ScoreBoard

Resources Provided

Resource Syntax

ScoreBoard(String newFileName, String initialScore)
void storeScore()
void getScore()
void setScore(String newScore)
void showScores()

Resource Semantics

Pre: there exists a file with name newFileName
ScoreBoard(String newFileName, String initialScore)
Post: self.getScore() = initialScore

Pre: true
void storeScore()
post: score has been stored

pre: true
int getScore()
post: return current score

pre: true
void setScore(String newScore)
post: self.getScore() = newScore

pre: true
void showScores()
post: true

Resource Usage Restrictions

File system must have 1 KB of available space.

Locally Defined Data Types

No types apply.

Error Handling

An exception is thrown if the newFileName parameter does not exist or there are insufficient permissions to create that file.

Variation Provided

Only the string in which the initial and current scores are stored can change. No variation is provided.

Quality Attribute Characteristics

The scoreboard must be readable. This restricts the combinations of background and text colors.

Element Requirements

- The element requires the file system API to open and close the file.
- The element is a specialization of the Panel component that provides windowing capability.

Rationale and Design Issues

The score is maintained as a string within the ScoreBoard, and each game converts its score to a string. Strings allow different games to have different formats and numbers of scores. For example, Brickles has a single score, Pong has two simultaneous scores, and Bowling has scores for all 10 frames.

Usage Guide

This simple component provides a simple interface. The component is placed in the game window.

SpeedControl Interface

Interface Identity

SpeedControl

Resources Provided

Resource Syntax

SpeedControl(int initialSpeed)
void setSpeed(int newSpeed)
int getSpeed()
void resetSpeed()

Resource Semantics

pre: true
SpeedControl(int initialSpeed)
post: self.speed = initialSpeed

pre: true
void setSpeed(int newSpeed)
post: self.speed = newSpeed

pre: true
int getSpeed()
post: return = speed

pre: true
void resetSpeed()
post: speed = defaultSpeed

Resource Usage Restrictions

No restrictions apply.

Locally Defined Data Types

No types apply.

Error Handling

No error handling applies.

Variation Provided

No variation applies.

Quality Attribute Characteristics

No characteristics apply.

Element Requirements

No requirements apply.

Rationale and Design Issues

This is a very simple structure.

Usage Guide

Just create an instance.

EventHandler Definitions

Interface Identity

EventHandler Definitions

Resources Provided

Resource Syntax

void MouseDown(object sender,MouseEventArgs e);
void MouseUp(object sender,MouseEventArgs e);
Rectangle MouseMove(object sender,MouseEventArgs e);
void MouseEnter(object sender,EventArgs e);
void MouseLeave(object sender,EventArgs e);
void KeyDown(object sender,KeyEventArgs e);

Resource Semantics

pre: mouse is attached and mouse driver has been loaded
void MouseDown(object sender,MouseEventArgs e);
post: MouseDown action has been executed

pre: self.isInvoked(MouseDown())
void MouseUp(object sender,MouseEventArgs e);
post: MouseUp action has been executed

pre: mouse is attached and mouse driver has been loaded
Rectangle MouseMove(object sender,MouseEventArgs e);
Post: MouseMove action has been executed

pre: mouse has entered the window for which this is the mouse event handler
void MouseEnter(object sender,EventArgs e);
post: MouseEnter action has been executed

pre: mouse has left the window for which this is the mouse event handler
void MouseLeave(object sender,EventArgs e);
post: MouseLeave action has been executed

pre: keyboard is attached and the keyboard driver has been loaded and a key has been down
void KeyDown(object sender,KeyEventArgs e);
post: KeyDown action has been executed

Resource Usage Restrictions

Devices must be attached.

Locally Defined Data Types

No types apply.

Error Handling

No error handling applies.

Variation Provided

The actions taken as a result of each hardware event can be redefined in each new implementation of this interface. In fact, each game uses a different implementation of this interface, which is provided as a parameter for the GameBoard component.

Quality Attribute Characteristics

Each handler routine should be sufficiently efficient to meet the “smooth movement” characteristic of the animation. The MouseMove event will be invoked frequently, so that implementation is very efficient. The response time need not meet the maximum number of events that the hardware is capable of generating since this will vary from one controller to another.

Element Requirements

No outside elements are required by the routines because of the interface. Each implementation may use some specific outside elements.

Rationale and Design Issues

This interface allows the GameBoard component to be generic. Each game requires a different set of actions when the player moves the mouse or presses or releases a button. Placing these actions in an object allows the GameBoard to be a container that delegates the events to the parameter object.

Usage Guide

This interface is stateless. The methods are invoked by events in the order in which they occur.

Evolved Architecture

The requirements model defines a couple of change cases. The high-level architecture shown in Figure 29, the Process Allocation for AGM Games, will evolve to the more complex structure shown in Figure 43 as those change cases are implemented. The evolved architecture contains the database in which the scores and state of the game will be stored.

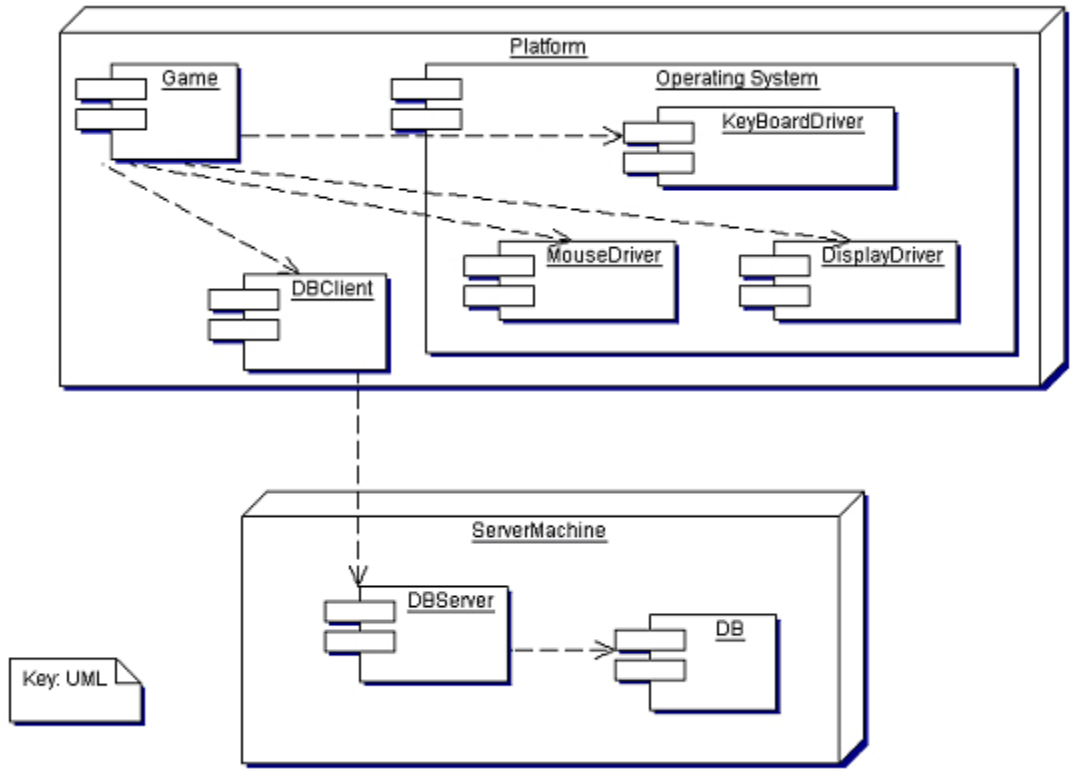

Figure 43: Evolved Architecture

Architecture Evaluation

This section presents the results of an architecture evaluation of the Arcade Game Maker (AGM) product line software architecture, which took place in Clemson, SC, on February 5, 2004. This evaluation was performed by Luminary Software and followed the SEI Architecture Tradeoff Analysis

Method® (ATAM®)[8], a method for evaluating a software system's architectural decisions in light of desired system quality attributes.

ATAM evaluations have two main phases. In Phase 1, the evaluation team interacts with the architect and a few other key stakeholders (e.g., a project manager or customer/marketing representative). The evaluation team and the system stakeholders walk through all the steps of the ATAM, gathering information about the system, its important quality attributes, and its architecture. We begin analyzing architectural decisions in light of the quality attributes to uncover risks and tradeoffs. The evaluation team continues the analysis after the Phase 1 meeting, interacting with the architect as necessary to elicit the necessary information. This interaction typically takes several weeks.

In Phase 2, the evaluation team walks a larger group of stakeholders through all the steps of the ATAM and the output from Phase 1. With this larger group, important quality attributes are illuminated, and the architecture's ability to support those goals continues.

Due to the simplicity of the AGM product line architecture, Phase 1 and Phase 2 were combined. No non-technical stakeholders were involved at the latter phase.

The system stakeholders (architects, managers, developers, testers, integrators, etc.) participating in the AGM product line ATAM evaluation are listed in Table 14.

Table 14: System Stakeholders Participating in the AGM Product Line Evaluation

Name	Organization	Email	Role

The ATAM evaluation team members and their assigned roles in the AGM ATAM evaluation are listed in Table 15.

Table 15: Evaluation Team for the AGM Product Line Evaluation

Name	Organization	Email	Role

[8] Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

The Architecture Tradeoff Analysis Method (ATAM)

The ATAM depends on an architecture being suitable (or not) only in the context of specific quality attributes that it must impart to the system. The ATAM uses stakeholder perspectives to produce scenarios that define the qualities of interest for the system under consideration. Scenarios give specific instances of usage, performance requirements, growth requirements, types of failures, possible threats, and likely modifications. After important quality attributes are identified in detail, the architectural decisions relevant to each one can be illuminated and analyzed with respect to their appropriateness.

The ATAM steps are carried out in two phases. In Phase 1, the evaluation team interacts with the system's primary decision makers: architects, managers, and perhaps a marketing or customer representative. During Phase 2, a larger group of stakeholders is assembled, including developers, testers, maintainers, administrators, and players. The two-phase approach insures that the analysis is based on a broad and appropriate range of perspectives.

Phase 1

1. **Present the ATAM**. Evaluators explain the method.
2. **Present business drivers.** Appropriate system representatives present an overview of the system, its requirements, business goals, context, and the architectural quality drivers.
3. **Present architecture.** The system or software architect (or another lead technical person) presents the architecture.
4. **Identify architectural approaches.** The system or software architects present general architectural approaches to achieve specific qualities. The evaluation team creates a list and adds approaches gleaned during Step 3 or learned during their pre-exercise review of the architecture documentation (e.g., "a cyclic executive is used to ensure real-time performance"). Known architectural approaches have known quality attribute properties, and these approaches will help carry out the analysis steps.
5. **Generate quality attribute utility tree.** Participants build a utility tree, which is a prioritized set of detailed statements about what quality attributes are most important for the architecture to support (e.g., performance, modifiability, reliability, or security) and specific scenarios that express these attributes.
6. **Analyze architectural approaches.** The evaluators and the architects map the utility tree scenarios to the architecture to see how it responds to each important scenario.

Phase 2

1. **Present the ATAM.** Evaluators explain the method.
2. **Recap of Steps 2 - 6 of Phase 1.** Evaluators recap Phase 1.

3. **Brainstorm and prioritize scenarios.** The stakeholders brainstorm additional scenarios that express specific quality concerns. After brainstorming, the group chooses the most important ones using a facilitated voting process.
4. **Analyze architectural approaches.** As in Step 6 of Phase 1, the evaluators and architects map the high-priority brainstormed scenarios to the architecture.
5. **Present results.** A presentation and final report are produced that capture the results of the process and summarize the key findings.

Scenario analysis produces the following results:

- **A collection of sensitivity and tradeoff points.** A sensitivity point is an architectural decision that affects the achievement of a particular quality. A tradeoff point is an architectural decision that affects more than one quality attribute (possibly in opposite ways).
- **A collection of risks and non-risks.** A risk is an architectural decision that is problematic in light of the quality attributes that it affects. A non-risk is an architectural decision that is appropriate in the context of the quality attributes that it affects.
- **A list of issues (or decisions not yet made).** During an evaluation, issues not directly related to the architecture may arise. They may have to do with an organization's processes, personnel, or other special circumstances. The ATAM process records them so that they can be addressed by other means. The list of decisions not yet made arises from the stage of the evaluation's life cycle. An architecture represents a collection of decisions. Not all relevant decisions may have been made at the time of the evaluation, even when designing the architecture. The development team knows about some of these decisions and has put them on a list for further consideration. Others are news to the development team and stakeholders.

Results of the overall exercise also include the summary of the business drivers, the architecture, the utility tree, and the analysis of each chosen scenario. All these results are recorded so that all stakeholders can verify that the results have been identified correctly.

The number of scenarios analyzed during the evaluation is controlled by the amount of time allowed for the evaluation, but the process insures that the most important ones are addressed.

After the evaluation, the evaluators document it, uncovered information, and the framework for ongoing analysis that they discovered. This documentation is the report in your hands. A detailed description of the ATAM process is provided by Clements, Northrop, Kazman, and Klein [Clements 02, Kazman 00].

Business Drivers Presentation

The AGM product line manager described AGM's business objectives for the AGM product line and identified the following product and architectural goals:

1. **Maintain low cost.** Many of the products are intended either for AGM to give away via the Web or for customers to giveaways at conventions.

2. **Offer the player an interesting game-playing experience.** People who play games are easily frustrated and quick to scorn games that they believe to be inferior. The game must be sufficiently fast and realistic.

For a complete list of driving business requirements, see the Business Case section.

Architecture Presentation

AGM's lead software architect presented the architecture of the AGM product line and identified the following approaches:

- a modified model-view-controller (MVC) front end
- a component-container structure for the model
- an event-driven error-handling mechanism

Several views are presented below. For more architectural details, see the Arcade Game Maker Architecture Documentation Beyond Views and Arcade Game Maker Software Architecture Views sections.

In Figure 44, the GameBoard is the central feature of the architecture. It is a container that includes all the game's action. The StationarySprites are the obstacles into which the MovableSprites collide. Movement and collision are the two major actions that occur in the games. Figure 45 illustrates how obstacles are defined.

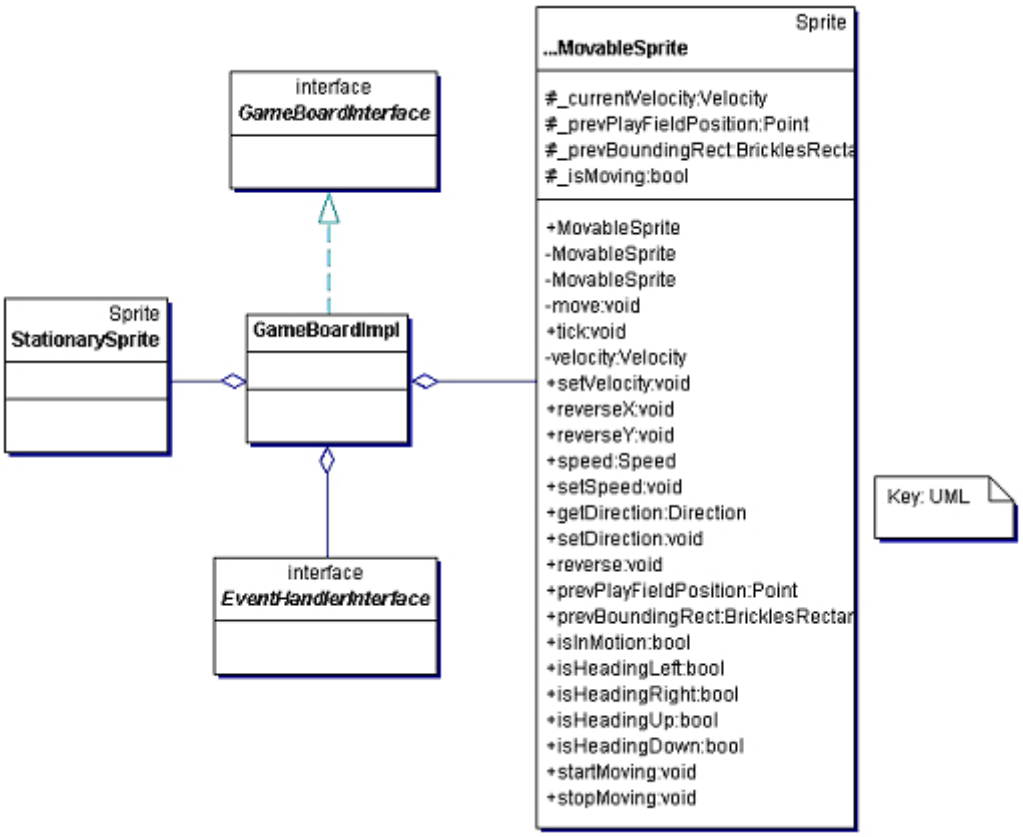

Figure 44: GameBoard-Centric View

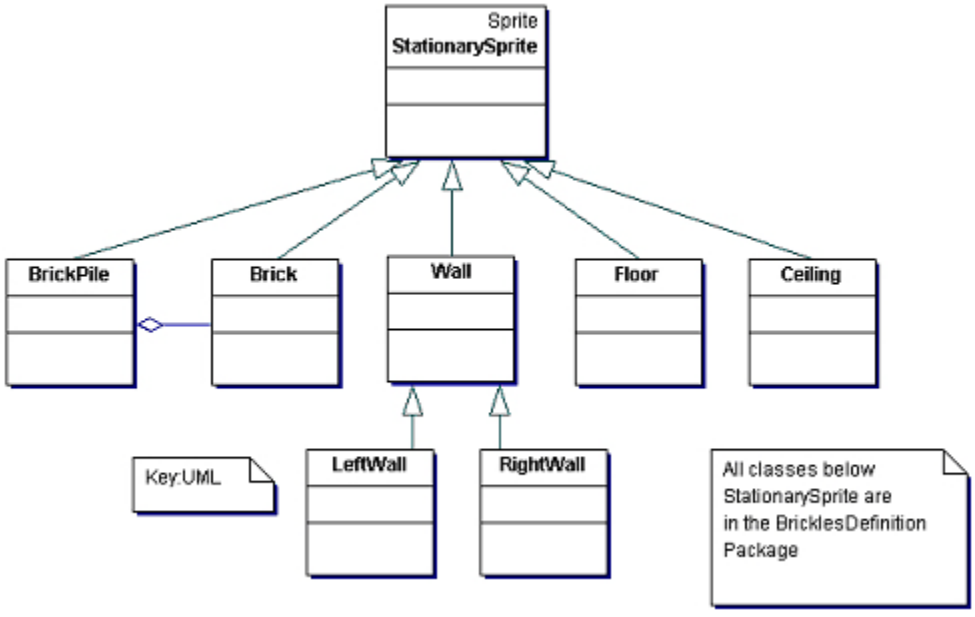

Figure 45: Specialization Hierarchy for StationarySprites

Utility Tree

The utility tree (shown in Table 16 and Table 17) provides a vehicle for translating the quality attribute goals articulated in the business drivers presentation to “testable” quality attribute scenarios. The tree contains Utility as the root node, which is an expression of the overall “goodness” of the system. For the *AGM product line*, the second level nodes were *performance and modifiability*.

Specific concerns are included under each quality attribute. These concerns arise from considering the quality-attribute-specific stimuli and responses that the architecture must address. For example, for the *AGM product line, performance* was broken into *two concerns: (1) the total number of Sprites on the screen and (2) how many Sprites have to be checked for collisions after each tick of the game clock.*

Finally, these concerns are represented by a small number of scenarios that are leaves (or levels) of the utility tree.

A scenario represents a use or modification of the architecture, applied not only to determine if the architecture meets a functional requirement, but also (and more significantly) for predicting system qualities such as performance, reliability, modifiability, and so forth.

The scenarios at the leaves of the utility tree are prioritized along two dimensions: *(1) importance to the system and (2) perceived risk in achieving this goal.* These scenarios are prioritized relative to each other as high, medium, or low risk.

Table 16: Phase 1: Quality Attribute Utility Tree - Performance

Quality Attribute	**Performance**	
Attribute Concerns	A. The growing number of StationarySprites may degrade game performance.	
Scenarios	1. With each tick of the simulation clock, each StationarySprite is checked to determine whether it has been hit by one of the MovableSprites. The next tick cannot be handled until all current Sprites have been checked for collisions.	(H,H)
	2. All StationarySprites are created at the start of a new match for a game. As the number of Sprites increases, so does the startup time for a game, which may become unacceptable.	(H,H)
	3.	
Attribute Concerns	B. The rate of ticks generated by the timer may be too slow for smooth animation.	
Scenarios	1. The player sees the animation as jerky motion.	(H,L)
	2.	
	3.	
Attribute Concerns	C. The rate of ticks generated by the timer may be too fast for the player to react.	
Scenarios	1. The player cannot move the paddle sufficiently fast to intercept the puck.	(H,L)
	2.	
	3.	

Table 17: Phase 1: Quality Attribute Utility Tree - Modifiability

Quality Attribute	**Modifiability**	
Attribute Concerns	A. The games' range of scoring procedures may be too broad to fit within the architectural approach.	
Scenarios	1. Pong and Brickles use simple counting schemes while Bowling requires more complex logic to compute the score.	(H,H)
	2.	
	3.	
Attribute Concerns	B. The modified MVC approach we are using may not be sufficiently flexible to accommodate later additions to the product line.	
Scenarios	1. The addition of Pinball may require so many more display attributes that a full-blown MVC would be a better choice.	(H,H)
	2.	
	3.	

Scenario Generation/Prioritization

In addition to the scenarios at the leaves of the utility tree, a scenario elicitation process allows stakeholders to *contribute* additional scenarios that reflect their concerns and understanding of how the architecture will accommodate their needs. A particular scenario may have implications for many stakeholders: for a modification, one stakeholder may be concerned with the difficulty of a change and its performance impact, while another may be interested in how the change will affect the integrability of the architecture. Table 18 shows scenarios collected by a round-robin brainstorming activity. After the scenarios were generated, they were prioritized using a voting process in which participants were given individual votes that they could allocate to any scenario or scenarios.

Table 18: Phase 2: Brainstormed Scenarios

Scenario	**Scenario Text**	**Votes**
1.	A player downloads a game executable to his computer. When he attempts to play the game, his computer does not have sufficient memory and the operating system reboots. He thinks that our game caused the problem and we lose his goodwill.	5
2.		

Analysis of Architectural Approaches

A-1 from the Phase 1 utility tree and the only scenario from Phase 2's generation/prioritization process were examined in detail vis-à-vis the AGM architecture.

Performance A-1: With each tick of the simulation clock, each StationarySprite is checked to determine whether it has been hit by one of the MovableSprites. The next tick cannot be handled until all current Sprites have been checked for collisions. *This is part of the main action loop. The current implementation uses a simple approach that searches all Sprites in the system. If the product does not meet performance goals, consider implementing a more sophisticated scheme (e.g., a screen grid system).*

Modifiability A-1: Pong and Brickles use simple counting schemes, while Bowling requires more complex logic to compute the score. *The architecture contains an interface so the scoring scheme can be readily changed.*

Phase 2: A player downloads a game executable to his computer. When he attempts to use the game, his computer does not have sufficient memory and the operating system reboots. He thinks that our game caused the problem and we lose his goodwill. *The architecture does not contain a start-up module that requires minimal memory and checks for the required amount of memory so that it can issue a message to the player before terminating.*

Risks, Sensitivities, and Tradeoffs

The analyses of architectural approaches by applying selected scenarios and the ensuing discussions uncovered a set of risks, sensitivities, and tradeoffs.

Risks

The collected risks are as follows:

Risk 1: The highest risk is the potential for player dissatisfaction. The current architecture supports a very simple graphical model. The graphics are not as realistic as other available games. If players are dissatisfied, they may have a negative impression of the company, and the goodwill expected from the free games will not accrue.

Non-Risks

The following items were identified as areas in which no risk is involved:

Non-Risk 1: Scoring and accurately representing the games. Each game is sufficiently simple that the scoring is clear and easily implemented.

Sensitivities

The collected sensitivities are as follows:

Sensitivity 1: The number of graphical elements in a game. If games were added to the product line that used a larger number of elements, many of the quality attributes would degrade.

Tradeoffs

The collected tradeoffs are as follows:

Tradeoff 1: There has been a tradeoff between memory use and modifiability. The full MVC architecture results in a duplication of data between the model and the views. Some pieces of data might be in every view plus the model. This duplication is not a problem for the free versions but would have been one for the wireless device versions. The modified MVC architecture only stores the data in one place, but it is much harder to add additional views. Because this is a fundamental architecture element, it was not made a variation.

Tradeoff 2: There has been a tradeoff between performance and simplicity. As stated previously in this paper, the graphical area of the game interface could have been divided into regions to speed collision checking. To reduce development costs, a single region was used.

Risk, Sensitivity, and Tradeoff Themes

The set of identified risks, sensitivities, and tradeoffs prompted a discussion of architecture themes. The following themes represent the key architectural issues that pose potential problems for the success of the venture:

Scalability. The architecture's ability to accommodate additional games is questionable. Because the architecture is very simple to keep costs low, certain techniques (e.g., dividing the graphics field into regions to speed collision checking) have not been used, and the architecture's ability to support a large number of screen objects while meeting performance goals is uncertain. The modified MVC architectural pattern will not easily support additional windows for displaying different information.

Realism. The realism of the games is questionable. The architecture does not support the sophisticated input and output devices that make interaction more natural. The graphics are very simple to allow the games to run on almost any hardware.

Conclusions and Next Steps

The results of the ATAM evaluation showed the need to modify the existing architecture to include user and system models that allow the player to control more aspects of the game. The detailed design process will assess whether this should be done through menu selections or a configuration file. The next step is to revise the architecture to accommodate these models.

The performance of the current design is adequate (just barely). We will not do anything explicitly to improve performance, but the work described above cannot degrade performance.

If this product line proves successful, we will revisit the architecture and modify it to support a second product line.

Unit Test Plans

Unit Test Plan Template

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about AGM products, see the Scope section. This section presents the unit test plan template, which is based on the report by McGregor [McGregor 01b]. Product line organizations use it to capture how each code unit is tested.

Readership

The unit test plan is intended primarily for core asset development teams. Managers can use it to determine what resources are needed to test assets, and technical team members can use it to develop a complete unit-specific test plan.

Template for the Test Plan

The AGM product line organization based its test plans on the Institute of Electrical and Electronics Engineers (IEEE) 829 standard. Each test plan will include the 15 sections listed below. For more details, see McGregor's report [McGregor 01b].

1. Introduction
2. Test Items
3. Tested Features
4. Features Not Tested (Per Cycle)
5. Testing Strategy and Approach
    a. Syntax
    b. Description of Functionality
    c. Arguments for Tests
    d. Expected Output
    e. Specific Exclusions
    f. Dependencies
    g. Success/Failure Criteria for Test Cases
    h. Pass/Fail Criteria for the Complete Test Cycle
6. Entrance and Exit Criteria
7. Test Suspension Criteria and Resumption Requirements
8. Test Deliverable and Status Communication Vehicles
9. Testing Tasks
10. Hardware and Software Requirements
11. Problem Determination and Correction Responsibilities

12. Staffing and Training Needs/Assignments
13. Test Schedules
14. Risks and Contingencies

Analyses and Standards

This section describes the techniques and agreed-upon standards used to test each component in the AGM product line. For justification and context, see the report by McGregor [McGregor 01b].

Coverage Standards

Functional Test Cases

For each service on a component, construct a test case for every clause of the postcondition. Test the unit invariant before and after each service invocation. Testing can be done in conjunction with the service test cases.

Structural Test Cases

Construct a test case for each sequence of statements. Optimize this effort by first executing the functional tests while running a code-coverage tool. Then construct test cases to cover sequences of statements that were not covered by the functional test cases. Be certain that this test case includes all exceptional sequences.

Analyses

Test Suite Construction

The AGM product line organization has decided to use the test case selection techniques described by McGregor [McGregor 01b]. Read that material before constructing test suites.

Incremental Test Analyses

After the initial test suites have been created, different techniques are used to maintain the test suites. These techniques should be applied every time the unit is changed.

Change Impact Analysis. Use the results from the change impact analysis that was conducted by the developers. That analysis will have identified those portions of the asset that will be modified when the change is implemented.

Diff. Use a tool such as diff to show the exact difference between two versions of the unit so that tests can be modified to address only those differences.

Modifying the Unit Test Plan

Modify a specific unit test plan every time the unit being tested is changed using the techniques described in the Analyses section. Modify the generic unit test plan when it is shown that the techniques are not producing effective test cases and the standards are not producing satisfactory results. That modification process is shown in Figure 46.

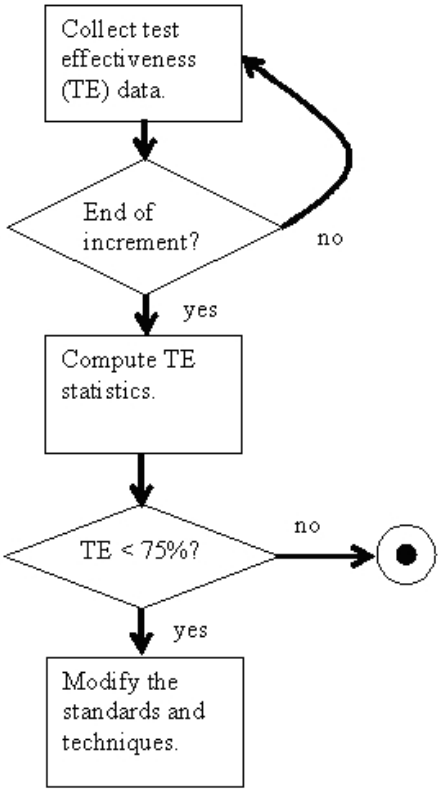

Figure 46: Attached Process

TE is computed for each unit test as shown below:

$$TE = \frac{TotalDefects - DefectsFoundAfterTesting}{TotalDefects}$$

Identified defects are cataloged and then analyzed to determine their origin. At the end of a product line increment, the TE is computed on a component-by-component basis. When the average TE goes below 75%, make test coverage standards more comprehensive.

Unit Test Plan: Velocity/Direction/Speed Cluster

This is the test plan for the Velocity/Direction/Speed cluster of classes. The basic classes in this cluster support the animation of movable sprites.

The Arcade Game Maker (AGM) product line organization based its test plans on the Institute of Electrical and Electronics Engineers (IEEE) 829 standard.

Test Items

This plan tests the Velocity, Direction, and Speed classes. These classes are tested together because they form a tightly coupled cluster. The Direction and Speed classes are primitive and require math

functions only from the underlying language library. The Velocity class uses instances of the Direction and Speed classes as shown in Figure 47.

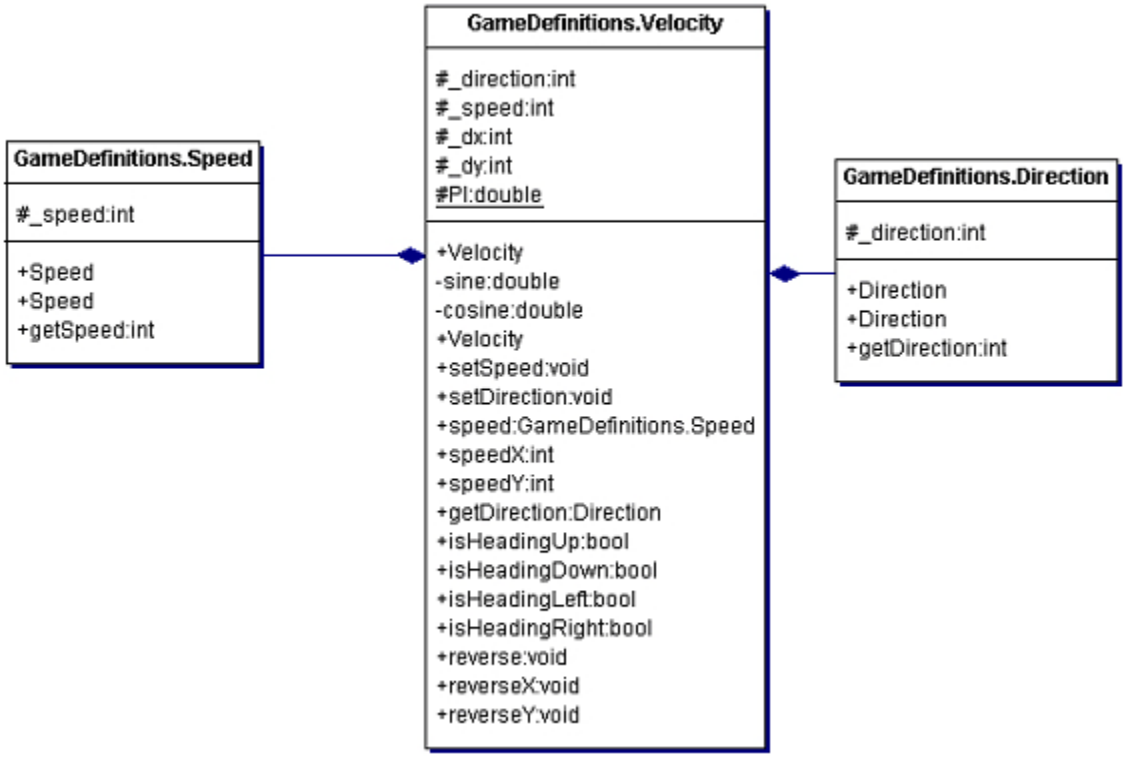

Figure 47: Cluster

Tested Features

The tested features are described in terms of Velocity. The Speed and Direction classes will be tested only to the extent that they are required to support Velocity. If they are used later without Velocity, additional testing will be required. Table 19 shows the three increments for developing the cluster. Functionality will be tested in the order shown.

Table 19: Development Increments

Increment	Functionality	Schedule
1	Define the basic functionality for Velocity. Complete the functionality for Direction and Speed.	
2	Revise Speed and Direction. Implement complete Velocity interface.	
3	Optimize the cluster.	

Features Not Tested (Per Cycle)

All features of Velocity will be tested during this cycle. Velocity is fundamental to correcting Sprite movement.

Testing Strategy and Approach

Syntax

The unit test cases are written in C#, as is the product code.

Description of Functionality

The Speed and Direction classes are simple encapsulation classes that hold a value and provide access to it. As such, a few basic values are used as shown in Table 20.

Table 20: Variables Involved in Tests

Variable	Type	Constraints
Speed	Integer	None
Direction	Integer	0<=direction<=360

Arguments for Tests

Table 21 and Table 22 show the analysis of data types used in Speed and Direction. Because Velocity is a combination of the Speed and Direction classes, its test table (shown in Table 23) is constructed from Speed and Direction.

A negative speed is not possible, but the class does not prevent negative values (an optimization decision).

A default constructor must also be tested.

Table 21: Speed Class

Test #	Speed Value
1	5
2	0
3	-5
4	Not entered

Table 22: Direction Class

Test #	Direction Value
1	0
2	360
3	27
4	90
5	410
6	Nothing

Table 23: Velocity Test Value Combinations

Velocity Test Value Combinations			
Test #	**Speed Value**	**Direction Value**	**Expected Result**
1	5	0	
2	5	360	
3	5	27	
4	5	90	
5	5	410	
6	0	0	
7	0	360	
8	0	27	
9	0	90	
10	0	410	
11	-5	0	
12	-5	360	
13	-5	27	
14	-5	90	
15	-5	410	

Expected Output

The output will vary for different tests. The next five tables provide the expected output for each method in the class. For Set/Get Direction, a default constructor must also be tested.

Set/Get Direction			
Test #	**Speed Value**	**Direction Value**	**Expected Result**
1	5	0	0
2	5	360	0
3	5	27	27
4	5	90	90
5	5	410	50
6	0	0	0
7	0	360	0
8	0	27	27
9	0	90	90
10	0	410	50
11	-5	0	0
12	-5	360	0
13	-5	27	27
14	-5	90	90
15	-5	410	50

SpeedX			
Test #	**Speed Value**	**Direction Value**	**Expected Result**
1	5	0	5
2	5	360	5
3	5	27	4
4	5	90	0
5	5	410	3
6	0	0	0
7	0	360	0
8	0	27	0
9	0	90	0

10	0	410	0
11	-5	0	-5
12	-5	360	-5
13	-5	27	-4
14	-5	90	0
15	-5	410	-3

SpeedY			
Test #	**Speed Value**	**Direction Value**	**Expected Result**
1	5	0	0
2	5	360	0
3	5	27	2
4	5	90	5
5	5	410	3
6	0	0	0
7	0	360	0
8	0	27	0
9	0	90	0
10	0	410	0
11	-5	0	0
12	-5	360	0
13	-5	27	-2
14	-5	90	-5
15	-5	410	-3

isHeadingUp			
Test #	**Speed Value**	**Direction Value**	**Expected Result**
1	5	0	False

2	5	360	False
3	5	27	False
4	5	90	False
5	5	410	False
6	0	0	False
7	0	360	False
8	0	27	False
9	0	90	False
10	0	410	False
11	-5	0	False
12	-5	360	False
13	-5	27	True
14	-5	90	True
15	-5	410	True

isHeadingDown			
Test #	**Speed Value**	**Direction Value**	**Expected Result**
1	5	0	False
2	5	360	False
3	5	27	True
4	5	90	True
5	5	410	True
6	0	0	False
7	0	360	False
8	0	27	False
9	0	90	False
10	0	410	False
11	-5	0	False
12	-5	360	False

13	-5	27	False
14	-5	90	False
15	-5	410	False

Specific Exclusions

There are no specific exclusions.

Dependencies

Speed and Direction tests have no dependencies. Velocity tests rely on the correct operation of the Speed and Direction classes.

Success/Failure Criteria for Test Cases

Each test passes if it exactly matches a numeric or Boolean value.

Pass/Fail Criteria for the Complete Test Cycle

The cluster must pass all applied tests. As fundamental and critically important classes, they must operate as expected all the time.

Entrance and Exit Criteria

The unit test phase runs concurrently with the unit development phase. Test case construction begins while the unit's specification is being developed. Feedback from test-case development allows the developer to identify and correct inconsistencies and ambiguities in the specification.

The unit test phase is exited when the developer has implemented the required unit and it has passed all tests required by the standards defined in the test plan template.

Test Suspension Criteria and Resumption Requirements

Unit tests are suspended for one of two reasons:

1. The available functionality has been adequately tested and has passed, but additional functionality remains to be developed.
2. The available functionality has not passed the tests, and the developer has sufficient information to make another development pass.

Tests are resumed after the developer has constructed additional functionality or revised the existing functionality.

Test Deliverable and Status Communication Vehicles

For every unit test (of a class or a cluster of classes), the developer delivers the test class. In this case, a class named Velocity Test is to be delivered.

Testing Tasks

Create Tables

The tables in an earlier section (Arguments for Tests) record the analysis of data values, constraints, and combinations of values.

Define DotUnit Test Class

Each test case in the tables becomes a method in the test class. An example is shown in Figure 48.

```
using System;
using dotUnit.Framework;
using GameDefinitions;

namespace DotUnit
{
        /// <summary>
        /// Summary description for VelocityTest.
        /// </summary>
        public class VelocityTest: dotUnit.Framework.TestCase
        {
                /// <summary>
                /// constructor
                /// </summary>
                /// <param name="name">Name for the test suite</param>
                public VelocityTest(string name) : base(name)
                {

                }

                /// <summary>
                /// test #1 in the Speed class test plan
                /// </summary>
                public void TestSpeed1()
                {

                        Speed s = new Speed(5);
                        AssertEquals(s.getSpeed(), 5);
                }

                /// <summary>
                /// test #2 in the Speed class test plan
                /// </summary>
                public void TestSpeed2()
                {

                        Speed s = new Speed(0);
                        AssertEquals(s.getSpeed(), 0);
                }
```

Figure 48: Test Class

Execute DotUnit

Use the GUIRunner to execute the tests. These tests can be repeated whenever the cluster has been changed.

Evaluate Results

Differences between expected results and the execution are examined. Either the unit under test or the tests themselves may be causing the error. After test cases have been used successfully for some time, it is more likely that the unit under test is wrong, rather than the tests.

Hardware and Software Requirements

No special hardware is required for the tests of this cluster.

The DotUnit executable and the accompanying framework classes are required for building test classes.

Problem Determination and Correction Responsibilities

The developer is responsible for making corrections. Cluster users are responsible for reporting any problems to the cluster owner.

Staffing and Training Needs/Assignments

The developer of the Velocity/Direction/Speed cluster has viewed the DotUnit training module.

Test Schedules

The unit test schedule parallels the development schedule as shown in Table 24.

Table 24: Test Schedule

Increment	Functionality	Schedule
1	Define the basic functionality for Velocity. Complete functionality for Direction and Speed.	Week of May 15
2	Revise Speed and Direction. Implement complete Velocity interface.	Week of May 22
3	Optimize the cluster.	Week of July 5

Risks and Contingencies

The probability of problems with this cluster is low because simple primitive classes are involved. However, the classes are critical to the success of the product line, so they must be tested thoroughly.

If these classes are not correct, there is the risk that the visible action will not be realistic and players will be frustrated.

Production Plans

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects, and it will be available on a variety of platforms. For more details about the scope, see the Scope section.

This production plan describes the general procedure for producing products with the current AGM asset base. The Management Information section provides a template for product-specific production plans. Product line organizations use production plans to capture how the product teams will build a new product, and a product-specific plan is developed before each product is constructed. This plan follows the outline provided by Chastek and McGregor [Chastek 02a].

Readership

The production plan provides a template for product-specific production plans and is intended primarily for product development teams. Managers can use the product-specific plan to determine the resources required to produce a product, and technical members can use it to actually produce a product.

Timeline

The production plan is a product of the core asset team. All of the base classes illustrated in Figure 49 have been implemented and are available to the product-specific development teams.

Strategic View of Product Development

Assumptions

Two broad assumptions apply:

1. The company has an existing pool of software developers who are highly technical. They have fielded products on a variety of hardware platforms and are accustomed to being involved down to the driver level of the software
2. The product line contains games that are similar in content but that differ in platform. Differences in platform translate into differences in graphics implementation, which is a major feature of these products.

Qualities

We will briefly describe two types of properties: (1) product and (2) production process.

Product Qualities

Players will enjoy the games if they have a colorful display and realistic action.

- A new game element should add to the display quality and portray the item it represents.
- Game action must be fast enough to demand the player's attention. If the number of elements slows the game, alternatives must be investigated.

- Game action must meet the player's expectation. The motion and reactions of movable elements must be realistic. As elements are added to the game, their actions and boundaries must be correctly set through parameters so that collisions look real.

Production Process Qualities

The production process in the AGM product line is largely manual. Developers are very technical, and the process allows for hands-on manipulation of the product. Providing a tool that automates the production process to a high degree would have frustrated the developers and wasted development resources.

Products Possible from Available Assets

The products that are possible from the available assets are simple, animated games that involve moving and stationary objects. Each game has an area in which all play occurs and implements a set of physical rules that control movement. Using the incremental approach, the asset base is only developed to the extent needed to construct the current set of products. As additional increments are completed, the variety of games that are possible may increase.

Production Strategy

The production strategy includes a domain-based design approach and a manual construction approach. The requirements analysis and architecture development will be based on domain information. For each new product, software developers will manually specialize assets, gather other domain-based core assets, and then build an executable by running a compiler and linker.

Core assets are being built incrementally. Therefore, earlier product teams will have fewer available assets than later product teams. Earlier product teams (particularly the freeware development team) will identify core asset candidates for use in the later increments.

The formal statement of the strategy[9] is as follows:

We will position ourselves as the leading provider of rapidly customized, high-performance, low-cost games by producing products that are easily modified, have better performance than our competitors, are sufficiently low cost to deter potential competitors from entering the market, and require sufficiently few resources to allow their use on any embedded computer. We will produce the initial products using a traditional iterative, incremental development process that uses a standard programming language, integrated development environment (IDE), and available libraries. We will create domain-based assets, including a product line architecture and software components, for the initial products in a manner that will support a migration to automatic generation of the second and third increment products.

---

9 For the complete rationale for this production strategy, see the "For the record" memorandum from the VPPD, Memo 04-01.

Overview of Available Core Assets

Source Code Naming Conventions

The following naming conventions are used:

- Game-specific classes begin with the name of the game and are created in a package devoted to that game.
- Interface definitions end with “Interface.”
- Class names begin with capital letters.
- Variable names begin with lowercase letters.
- Suite methods for the DotUnit test case begin with “test.”

Analysis-Level Assets

The requirements document includes the following assets:

- The domain analysis model organizes the concepts in the main domain (games) and provides the attributes and relationships of each concept.

  Developers who are new to the domain should study this asset to understand the product’s background.
- The feature model shows the features of products.
- The use case model provides a superset of product requirements. Select the appropriate use cases for your product.

High-Level Design Assets

The main high-level design asset is the software architecture, which is described in the Arcade Game Maker Architecture Documentation Beyond Views and Arcade Game Maker Software Architecture Views sections. Figure 49 shows the architecture’s base classes. Product-specific versions of these classes should be defined at a high level in each product-specific production plan.

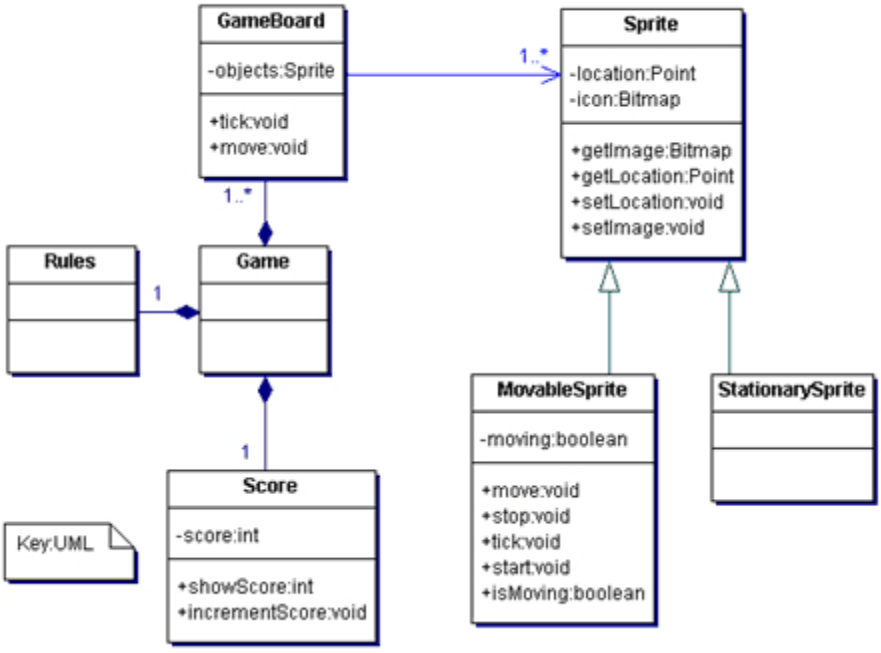

Figure 49: Base Classes

Source Code

Each interface in the existing architecture has been implemented. The C# components are named according to the guidelines in the Source Code Naming Conventions section.

Test Cases

This section describes *existing* test cases. Use an existing test case if it covers functionality or features included in the new product.

Test cases are not yet available for all code assets.

Unit Tests

Individual unit tests are constructed using the DotNet testing framework and are available in the Unit Test Plans section. The source code is in the form of classes.

The following unit test classes are currently available:

- The Velocity/Direction/Speed Cluster of tests

Integration Tests

If integrating units results in a component, test it at the API level. (If it results in a GUI, see the following System Test Plans section.)

System Tests

System tests are currently done manually. Each system test is a scenario derived from a specific use case. When a use case is applied to more than one product, the related test cases can also be applied to that new product. These test cases are documented in the test plan for a product.

Inputs and Dependencies

Inputs

Game inputs are as follows:

- mouse and keyboard events
- names of the files to save the score and the state of the game (when the change cases for saving are implemented)

Dependencies

Game dependencies are as follows:

- graphics library of the programming language
- operating system (when the change case for saving a game is implemented)

Variations

Absorbing vs. Reflecting

The stationary game elements participate in the game by providing collision behavior. Core assets include two major behaviors:

1. A stationary element may reflect the movable element according to the laws of physics.
2. A stationary element may absorb the movable element so that it is deleted from the game.

A parameter on each element determines which of these behaviors is performed.

Event Handling

The event handling routines vary from one game to another. An implementation of the EventHandlerDefinitions interface is provided as a parameter to the GameBoard component. Each mouse and keyboard event is handled as defined in the implementation of the parameter.

Detailed Production Process

Product teams will use the following four-step production process that was developed as Brickles and Pong freeware games were built:

1. identify the product, define it, and analyze it incrementally
2. design the product

3. build the product
4. test the product

The exact content of the steps is determined by the attached processes of the core assets that are selected to be used to produce the product.

Identify, Define, and Analyze the Product Incrementally

This step includes the following three tasks:

1. Identify products.

   (Products are already identified. Because each product is a single game, this task was accomplished when you identified games during the product line planning.) *Replace this with the scoping rules defined in the scope document that determine whether a product is in scope or not.*
2. Define the rules of the game.

   There are several versions of most of these games, so the product team must first decide on a set of rules to implement.
3. Analyze features for the new game that are variations from previous games, and identify existing features that must change for this game. *Replace this with the attached process of the feature model. That process should describe how to create a product configuration by resolving the variations in the feature model.*

Design the Product

This step includes the following tasks:

1. Derive the product-specific architecture from the product line architecture. *Replace this with the attached process from the software architecture that describes how to resolve the variation in the product line architecture.*
2. Plan how to provide those features from existing components. *Add the attached processes from the selected components. These attached processes should contribute to the feature set and to the test suites.*
3. Plan how to provide the remaining features from new assets.
4. Design the product-specific implementation of the game interface.
5. Design the new implementation of the EventHandlerDefinitions interface. *Replace this with the attached process for the interface definition. This will provide guidance on implementing the interface.*

Build the Product

This step includes the following 11 tasks:

1. Start new ClassLibrary in a Visual Studio Project using *Game_name* Definitions as its name. Use this class for all new classes except the game definition.
2. Start a new Windows Application in a Visual Studio Project using *Game_name*.

3. Copy form1.cs from a previous product line project.
4. Change the namespace name to *Game_name*.
5. Configure the new GameBoard.
6. *Replace this with the attached process from the configuration management plan that will guide the product developers in managing the build.*
7. Copy data.txt from a previous game's working directory. This is the resource file for the game.
8. Edit data.txt to reflect the new game.
9. Write the game-specific classes needed for the game.
10. Compile the resource file.
11. Copy the compiled resource file to the Debug directory.

Test the Product

This step includes the following five tasks:

1. Test each core asset, by inspection or execution, as it is created or revised. If the asset is revised, revise the previous testing materials and reapply them to the new version of the asset.
2. For a code asset, code the unit test asset as a DotUnit test class. *Replace this with the attached process from the abstract unit test suites that describes how to derive concrete test cases from the abstract test cases.*
3. Revise the initial *generic* game system test set for each new game.
4. Create a game-specific system test set for each new game as shown in Figure 50.
5. Maintain system test cases as text documents and apply them manually. *Add the attached process for each of the features selected for this product. This process will point to test cases and test procedures for that feature.*

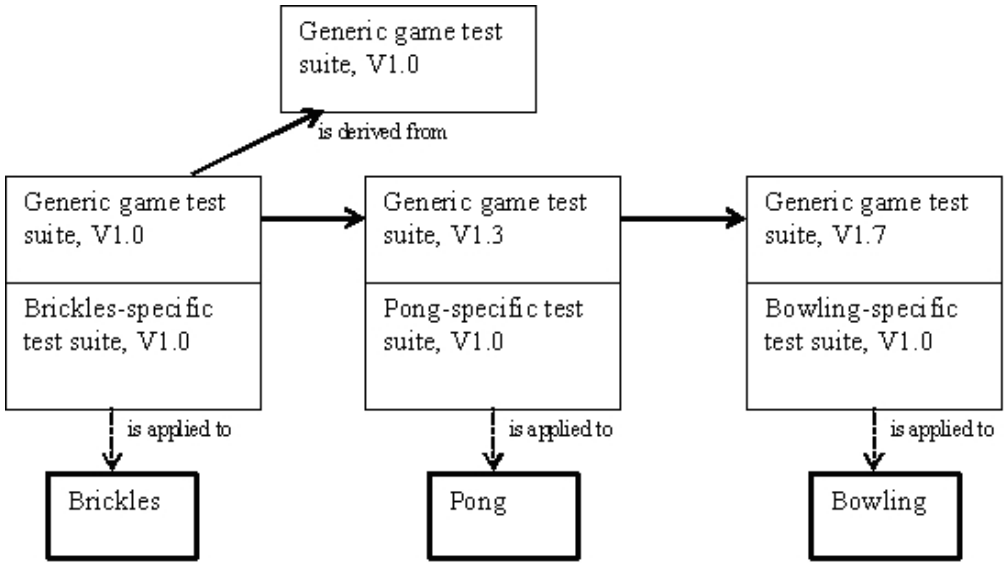

Figure 50: System Test Cases Related to Products

Tailoring the Production Plan to a Product-Specific Production Plan

Overall, the product line production plan is very generic and applies to all products built using the current asset base. However, some parts of the plan must be modified for a specific product. Those portions are marked by indicating where to insert the attached processes for the core assets selected to support product-specific development.

In the Detailed Production Process section, each subsection includes directions on how to use a specific attached process in building the production plan.

The Management Information section includes the two most important and obvious sections that must be modified: the schedule and the bill of materials (BOM). Since we are using a manual product production approach, the schedule defines which personnel are needed when. The BOM provides a method for tracking the use of core assets.

Management Information

Schedule

Table 25 provides a schedule template that includes all steps from the process described in the Detailed Production Process section. For the product-specific production plan, update the entries in the Performed By and Estimate/Date Performed columns.

Table 25: Schedule Template

Process Step	Product-Specific Task	Performed By	Estimate/Date Performed
Product Identification	Done during product planning	Product planning	Done
Product Definition	Define the rules of the new game.	Analyst	≅ 0.5 day
Product Analysis	Analyze features.	Analyst	≅ 0.5 day
Product Design	Identify new elements needed by the game.	Designer	≅ 2 hours
	Identify changes to existing elements.	Designer	≅ 0.5 day
	Design product-specific implementation of the game interface.	Designer	≅ 2 hours
	Design product-specific implementation of EventHandlerDefinitions.	Designer	≅ 2 hours
Product Build	Create new implementations and make changes to existing classes.	Developer	≅ 0.5 days
	Create new .Net projects for libraries and the application.	Developer	≅ 0.5 days
	Create the new make file.	Automated/developer	≅ 0.5 days
Product Test	Create unit tests for new elements.	Developer	≅ 0.5 day

	Modify existing unit tests for existing elements.	Developer	≅ 0.5 day
	Execute unit tests.	Developer	≅ 0.5 day
	Modify/extend the product test suite.	Tester	≅ 1 day
	Execute the product tests.	Tester	≅ 0.5 day

Production Resources

The primary resources are the Visual Studio .Net environment and the organization's UML modeling tool.

During the analysis and design steps, extend the UML model to include any new elements that must be defined.

Use the .Net environment to create any new components that are required. After all components are created, use the environment to build an executable.

Bill of Materials (BOM)

The BOM allows you to track the use of core assets. Table 26 includes a high-level list of available code assets. For the product-specific production plan, update the table with only those specific assets that will be used.

Table 26: Bill of Materials (BOM) Template

Component	Source	Cost
Product-specific Game component	In-house	$0
Generic GameBoard	In-house	$0
Required Sprites (add specifics here)	In-house	$0
Implementation of EventHandlerDefinitions interface	In-house	$0

Product-Specific Details

The rules of the game are the most unique parts of the product. Distribute them across the EventHandlerDefinitions and the Game component, which is the container for the entire game implementation. Most of the rules enforced by Sprites are common across most (if not all) of the games. Develop the EventHandlerDefinitions component (like the Game component) specifically for a game.

Pay attention to the animation loop, which is the single most important product-specific detail in the Game class. It defines the game's sequence of events.

Metrics

Use the following two metrics to evaluate the product production process: (1) the number of new lines of code and (2) the number of unique lines of code.

New Lines of Code

This metric describes the number of lines of new code that must be written for this product. This new code may or may not be used in another product later. A high value indicates more effort required to produce the product and impacts cost and schedule.

Unique Lines of Code

This metric describes the percentage of the lines of code that are unique to this product. This metric changes over time: it can be difficult to predict which code gets reused or remains unique. A high value indicates less similarity between products and indicates a longer payback time.

Attached Processes[10]

The main attached process that describes how to create the product specific production plan is defined in the Tailoring the Production Plan to a Product-Specific Production Plan section.

Constructing the Production Plan

Figure 51 shows a high-level version of building a production plan. For details, review Chastek and McGregor's work [Chastek 02a].

---

10 This section is the "attached process" described by Clements and Northrop [Clements 02]. The process in Constructing the Production Plan defines how the production plan is initially built. Changing the Production Plan focuses on modifying the existing production plan of the product line.

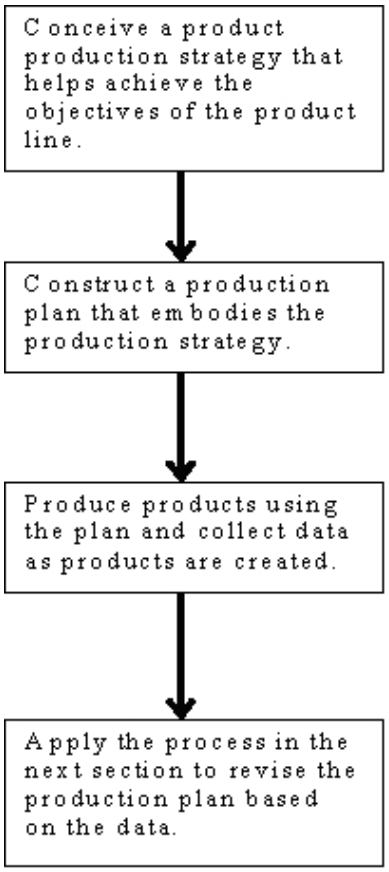

Figure 51: High-Level Process for Constructing a Production Plan

Changing the Production Plan

Figure 52 describes the change process. After each product is produced, data are collected and used to update the core asset base for the next increment's products. After those changes are reflected in the architecture documentation and the generic production plan, the plan is reviewed to identify inconsistencies between it and the core asset base. The core asset team member who owns the plan initiates the review.

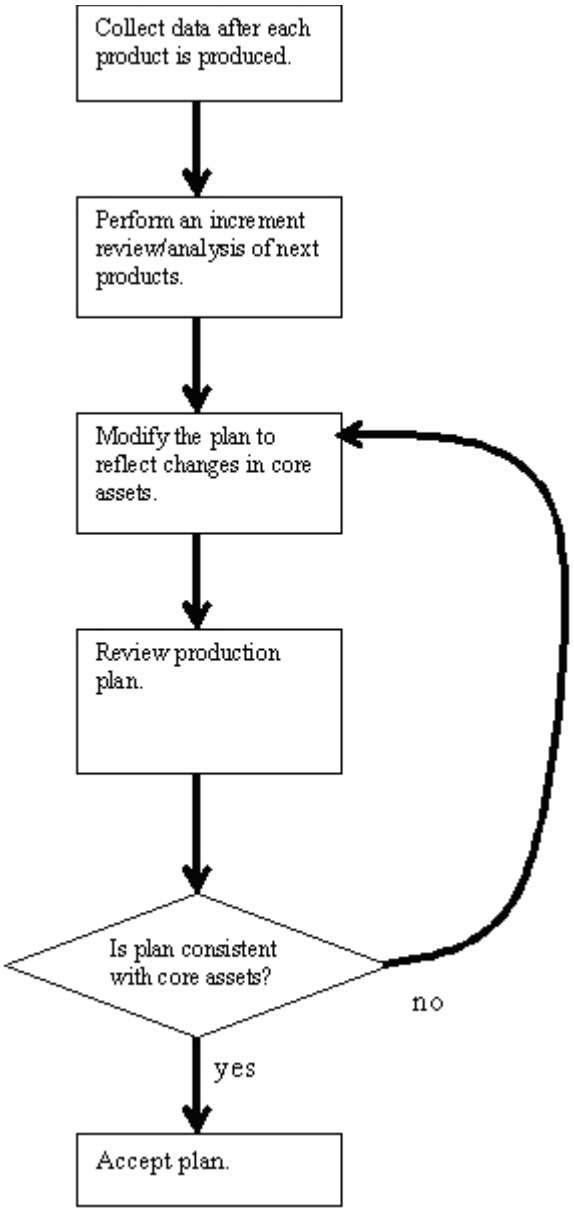

Figure 52: Production Plan Change Process

System Test Plans

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about AGM products, see the Scope section.

This section presents the system test plan template, which is based on the report by McGregor [McGregor 01b]. Product line organizations use it to capture how a new product is tested.

Readership

The system test plan template is intended primarily for product development teams. Managers can use it to determine what resources are needed to produce a product, and technical team members can use it to develop a detailed process for testing a product.

Plan

The AGM product line organization based its test plans on the Institute of Electrical and Electronics Engineers (IEEE) 829 standard. Each test plan will include the 15 sections listed below. For more details, see McGregor's report [McGregor 01b].

1. Introduction
2. Test Items
3. Tested Features
4. Features Not Tested (Per Cycle)
5. Testing Strategy and Approach
    a. Syntax
    b. Description of Functionality
    c. Arguments for Tests
    d. Expected Output
    e. Specific Exclusions
    f. Dependencies
    g. Success/Failure Criteria for Test Cases
6. Pass/Fail Criteria for the Complete Test Cycle
7. Entrance and Exit Criteria
8. Test Suspension Criteria and Resumption Requirements
9. Test Deliverable and Status Communication Vehicles
10. Testing Tasks
11. Hardware and Software Requirements
12. Problem Determination and Correction Responsibilities
13. Staffing and Training Needs/Assignments
14. Test Schedules
15. Risks and Contingencies

Analyses

This section describes the techniques and agreed-upon standards used to test each component in the AGM product line. For justification and context, see the report by McGregor [McGregor 01b].

Coverage Standards

The system test coverage standard is based on use case information in the Requirements section. The frequency and criticality fields for each use case determine the correct level of test coverage. Both are rated on high, medium, and low scales. Table 27 shows the combinations of these values and the coverage standard defined for each.

Table 27: Coverage Standards

Value Combination Frequency, Criticality	Coverage Criteria		Details
High, high	Main	X	For each tested scenario, construct test cases for all pair-wise combinations of scenario variables that are defined in the system state machine.
	Alternative	X	
	Exceptional	X	
High, low	Main	X	For each tested scenario, construct test cases for all pair-wise combinations of scenario variables that are defined in the system state machine.
	Alternative	X	
	Exceptional		
Medium, high	Main	X	For each tested scenario, construct test cases for some pair-wise combinations of scenario variables that are defined in the system state machine.
	Alternative		
	Exceptional		
Medium, low	Main	X	For each tested scenario, construct test cases for some pair-wise combinations of scenario variables that are defined in the system state machine.
	Alternative		
	Exceptional		

Examples

The analyses needed to support the creation of test cases from the use cases are conducted by filling out the next two tables. Example values are provided in the tables.

Data Type Analysis		
Variable	**Data Type**	**Equivalence Classes**
age	Integer	0 - 18; 19-21; 22-65; 66+

Test Case	Variable 1 (age)	Variable 2	Variable 3	Variable 4
1	12			
2	19			
3	56			
4	67			

Modifying the System Test Plan

Modify a specific unit test plan every time the unit being tested is changed using the techniques described in the previous section. Modify the generic unit test plan when it is shown that the techniques are not producing effective test cases and the standards are not producing satisfactory results. That modification process is shown in Figure 53.

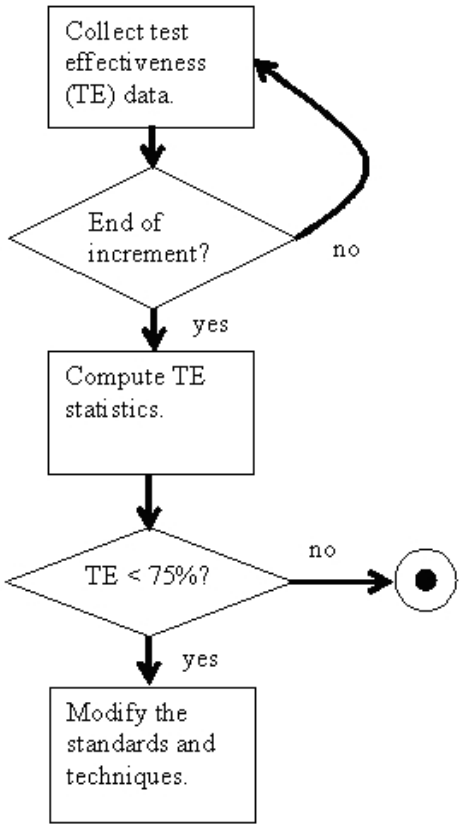

Figure 53: Modifying a Unit Test Plan

Attached Process

TE is computed for each unit test as shown below:

$$TE = \frac{TotalDefects - DefectsFoundAfterTesting}{TotalDefects}$$

Identified defects are cataloged and then analyzed to determine their origin. At the end of a product line increment, the TE is computed on a component-by-component basis. When the average TE goes below 75%, make test coverage standards more comprehensive.

Arcade Game Maker Pedagogical Product Line: Brickles Product

Brickles User Manual

The Arcade Game Maker (AGM) product line organization is producing a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about the products, see Arcade Game Maker Pedagogical Product Line: Scope.

Readership

The Brickles User Manual is for game players. It describes how to install and play Brickles.

Install Brickles

Install Brickles as follows:

1. Unzip the distribution file.
2. Provide access to the .Net runtime environment for the executable.
3. Double-click the Brickles icon.

The game is opened and initialized as shown in Figure 54.

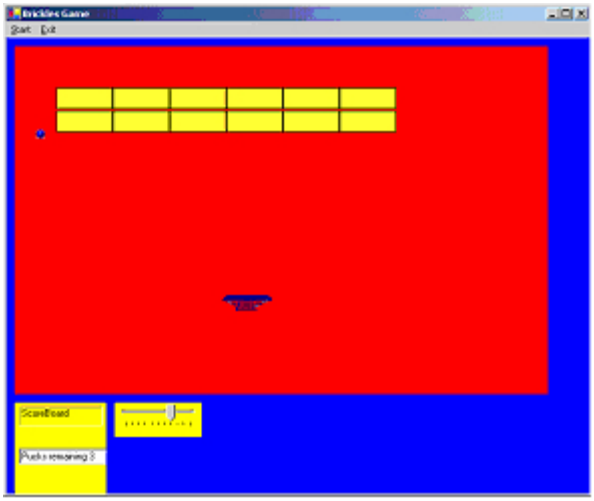

Figure 54: Initialized Brickles Display

Modes of Operation

Brickles has three modes of operation:

1. **Initialized**
   The executable is loaded and the display is initialized.
   To enter this mode, double-click the Brickles icon.
2. **Playing**
   Animation begins (elements move around the screen).
3. To enter this mode, left-click within the boundaries of the playing area.
4. **Paused**
   Animation stops (all elements are stationary).
   To enter this mode, hold down the left mouse button.
   To resume play, release the button.

Rules for Brickles

Brickles has a few simple rules:

- A player gets three pucks to break all of the bricks in the brick pile.
- If a puck hits the floor, it is removed from play.
- If the player breaks all the bricks, he wins.
- If the player runs out of pucks, he loses.

Test Brickles

Test Brickles as follows:

1. If you closed the game, double-click on the Brickles icon.

2. Left-click within the boundaries of the playing area.
   Animation begins.
3. Let the puck collide into a brick or the floor.
   Based on the rules, the puck is absorbed or changes direction according to the laws of physics.
4. Hold down the left mouse button.
   The game is paused.
5. Release the left mouse button.
   The game is resumed.
6. On the toolbar, click Help.
   Online help is displayed.
7. On the toolbar, click File > Exit.
   The game exits.

Advanced Features

No advanced features are implemented in this version.

Brickles Test Plan

The product to be tested will be downloaded free of charge as a means of attracting new clients. It represents Arcade Game Maker (AGM) and must be high quality. This plan defines tests that are the last in a series of quality assurance tasks.

This plan follows the IEEE-829 standard for test documentation. To complete it, we perform the analyses required to develop the Brickles test plan.

Test Items

The Brickles game is a subset of the total functionality needed for the product line. The set of use cases is shown in Figure 55.

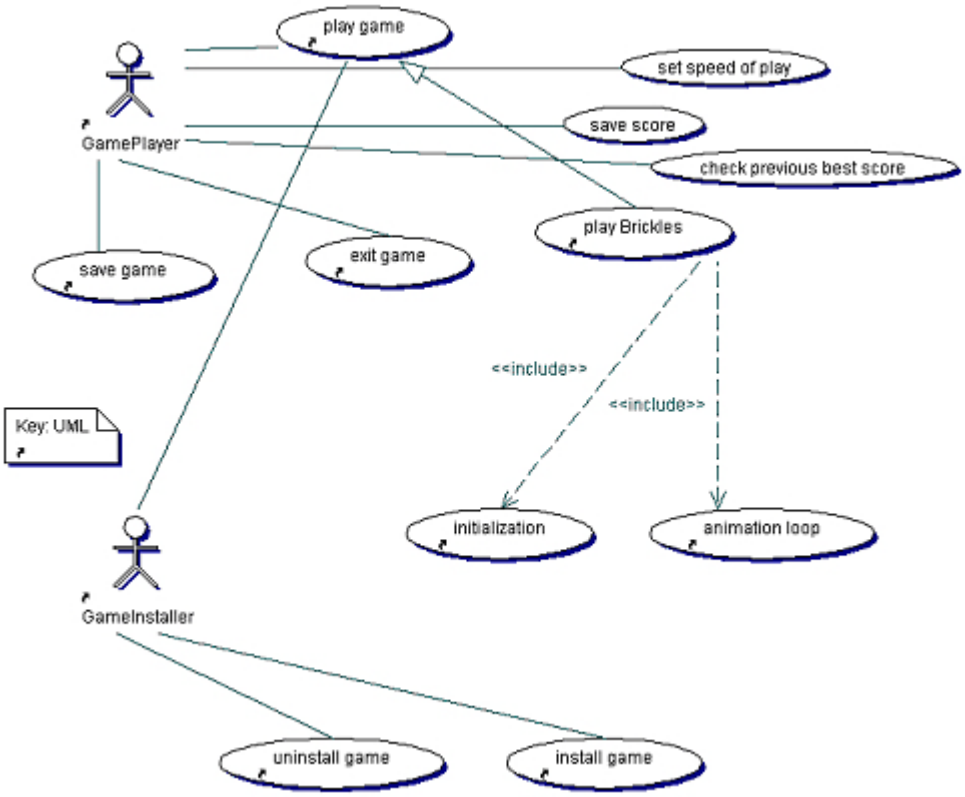

Figure 55: Brickles Use Cases

The following use cases apply to Brickles:

- AGM001 - Play game
- AGM002 - Exit game
- AGM006 - Play Brickles
- AGM009 - Initialization
- AGM010 - Animation loop
- AGM011 - Install game
- AGM012 - Uninstall game
- Change case - Save game
- Change case - Set speed of play
- Change case - Check previous best score
- Change case - Save score

For the full text for these use cases, refer to the Requirements section.

Tested Features

Except for features listed in the next section, all features described in Figure 55 will be tested. Review each of the relevant use cases in the Requirements section.

Features Not Tested (Per Cycle)

The following features will not be tested:

- change cases identified in the use case model but not yet implemented
- product line requirements that do not pertain to the current product

Testing Strategy and Approach

Syntax

The player communicates with the Brickles program using the mouse and keyboard.

Description of Functionality

The functionality being tested is the basic Brickles game running in a simple Win32 environment.

Arguments for Tests

Table 28 includes an analysis of the scenarios in the use cases shown in Figure 55, and Table 29 includes the expected results.

Table 28: Variable Data Types

Variable	Data Type	Equivalence Classes
Left mouse button	boolean	Up, down, down & up
Mouse location	Point	Inside window, outside window
Number of pucks lost	int	0, 0 < *x* < 3, 3
Number of bricks remaining	int	0, all, 0 < brickCount < all

Expected Output

Table 29: Expected Output

Test Case	Left Mouse Button	Mouse Location	Pucks Lost	Bricks Remaining	Expected Results
1	Down & up	Inside window	0	0	Won dialog
2	Down & up	Inside window	1	0	Won dialog
3	Down & up	Inside window	2	0	Won dialog
4	Down & up	Inside window	3	>0 and <all	Lost dialog
5	Down & up	Inside window	3	all	Lost dialog

6	Down	Outside window	Don't care	>0	No effect
7	Down & up and then down & up	Inside window	2	0	Won dialog

Specific Exclusions

No functionality will be excluded.

Dependencies

These tests assume that all the components of the Brickles game have been completed and unit tested.

Test Case Success/Failure Criteria

The expected results are all discrete. A test case will be a success if the actual program results exactly match the expected results shown in Table 29.

Pass/Fail Criteria for the Complete Test Cycle

All tests must pass for the product to be released. The use case set is a minimal set of requirements.

Entrance Criteria/Exit Criteria

Subsets of the system test suite will be executed as soon as a build is achieved.

The system test phase is exited when the release criteria described in the Pass/Fail Criteria for the Complete Test Cycle section are achieved.

Test Suspension Criteria and Resumption Requirements

Testing is suspended after the tests for all available features have been applied. Testing resumes when a new feature is added to the build.

Test Deliverables/Status Communications Vehicles

The test plan and use cases are placed in the configuration management system so that they can be used for each system revision. These assets will also contribute to creating the test cases for other products.

System Test Plans template is also a deliverable from this first system test phase. The template is derived by parameterizing any Brickles-specific information.

A test report will be produced. It includes test results and the status of system testing.

Testing Tasks

The test process has three main phases: (1) analysis, (2) construction, and (3) execution.

Analysis

The scenarios in the use cases are the basis for the analysis. Each scenario is first examined for varying data. Varying data types are used as the basis for the initial data table. Specific data variables are instantiated and assigned values.

Construction

The test cases will be executed directly from the data tables. No software will be constructed.

Execution and Evaluation

The tests will be executed by a trained tester who will evaluate the response produced by each test and rate it as passed or failed.

Hardware and Software Requirements

The Brickles game runs on any hardware that supports the .Net Common Language Runtime (CLR). The system tests will be performed on a machine other than the development machines.

Problem Determination and Correction Responsibilities

The developers will be responsible for reading the test report, determining the cause of each failure, and correcting the underlying defects.

Staffing and Training Needs/Assignments

This testing will initially be carried out by executing the game. No special training is required for this activity. If a capture/playback tool is adopted later, tool training will be required.

Test Schedules

Within two working days of receiving a developer-certified build, system test personnel will test and report.

Risks and Contingencies

If system testing is not completed in a timely manner, there may not be enough time to repair all defects. This is a high risk given the need to develop a test plan template concurrently. This can be mitigated to some extent by scheduling work on the template after the Brickles test plan is completed but before the Brickles system is ready for test.

Analyses

This section describes the techniques and agreed-upon standards used to test each product in the AGM product line. They are justified and put into context by McGregor [McGregor 01b].

Coverage Standards

The system test coverage standard is based on use case information in the Requirements section. The frequency and criticality fields for each use case determine the correct level of test coverage. Both are rated on high, medium, and low scales. Table 30 shows the combinations of these values and the coverage standard defined for each.

Table 30: Coverage Standards

Value Combination Frequency, Criticality	Coverage Criteria		Details
High, high	Main	X	For each tested scenario, construct test cases for all pair-wise combinations of scenario variables.
	Alternative	X	
	Exceptional	X	
High, low	Main	X	For each tested scenario, construct test cases for all pair-wise combinations of scenario variables that are defined in the system state machine.
	Alternative	X	
	Exceptional		
Medium, high	Main	X	For each tested scenario, construct test cases for some pair-wise combinations of scenario variables that are defined in the system state machine.
	Alternative		
	Exceptional		
Medium, low	Main	X	For each tested scenario, construct test cases for some pair-wise combinations of scenario variables that are defined in the system state machine.
	Alternative		
	Exceptional		

Examples

The analyses needed to support the creation of test cases from the use cases are conducted by filling out the tables below. Example values are provided in the following tables.

Data Type Analysis		
Variable	**Data Type**	**Equivalence Classes**
age	int	0-18; 19-21; 22-65; 66+

Test Case Construction				
Test Case No.	**Variable 1 (age)**	**Variable 2**	**Variable 3**	**Variable 4**
1	12			
2	19			
3	56			
4	67			

Modifying the System Test Plan

Modify the specific system test plan every time the requirements for the system being tested are changed using techniques described in the Examples section. Modify the generic system test plan when techniques are not producing effective test cases and standards are not producing satisfactory results. That modification process is shown in Figure 56.

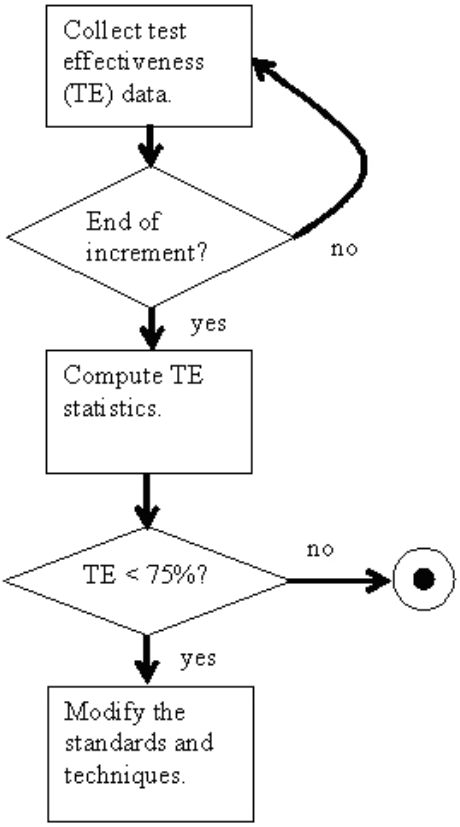

Figure 56: Attached Process

TE is computed for each unit test as shown below:

$$TE = \frac{TotalDefects - DefectsFoundAfterTesting}{TotalDefects}$$

Identified defects are cataloged and then analyzed to determine their origin. At the end of a product line increment, the test effectiveness is computed on a component-by-component basis. When the average TE goes below 75%, test coverage standards are made more comprehensive.

Brickles Test Report

The Brickles product was tested according to the plan defined in the Brickles system test plan. The product passed all of the tests, and this report provides the results of the testing conducted on August 1, 2003.

Test Items

The basis for the tests are the use cases shown here. The Brickles game is a subset of the total functionality needed for the product line. The set of use cases that form the basis for the tests is shown in the diagram.

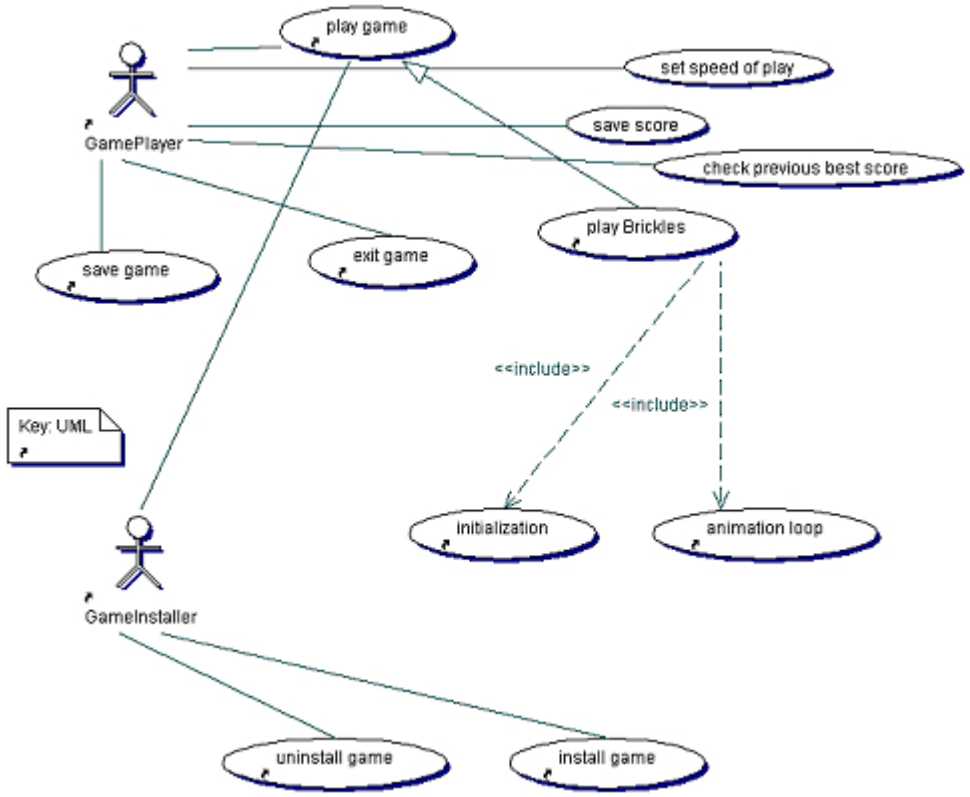

Figure 57: Brickles Use Cases

The following use cases apply to Brickles:

- AGM001 - Play game
- AGM002 - Exit game
- AGM006 - Play Brickles
- AGM009 - Initialization
- AGM010 - Animation loop
- AGM011 - Install game
- AGM012 - Uninstall game
- Change case - Save game
- Change case - Set speed of play
- Change case - Check previous best score

- Change case - Save score

For the full text of these use cases, refer to the Requirements section.

Testing Strategy and Approach

Syntax

The player interacts with the Brickles program using the mouse and the keyboard.

Description of Functionality

The functionality being tested is the basic Brickles game running in a simple Win32 environment.

Arguments for Tests

An analysis of scenarios in the use cases is depicted in Table 31. Assign variables to values that can vary, and determine each variable's data type by examining specifications and the context. For each data type, define a set of equivalence classes. These classes are sets of values within the variable's data type for which product behavior is expected to be equivalent.

Table 31: Use Case Analysis

Variable	Data Type	Equivalence Classes
Left mouse button	boolean	Up, down, down & up
Mouse location	Point	Inside window, outside window
Number of pucks lost	int	0, 0 < numberOfPucksLost < 3, 3
Number of bricks remaining	int	0, all, 0 < brickCount < all

Expected Output

As test cases are built, an oracle (usually a human) defines what constitutes a correct answer. Table 32 shows the expected results for each test case.

Table 32: Expected Output

Test Case	Left Mouse Button	Mouse Location	Pucks Lost	Bricks Remaining	Expected Results
1	Down & up	Inside window	0	0	Won dialog
2	Down & up	Inside window	1	0	Won dialog
3	Down & up	Inside window	2	0	Won dialog
4	Down & up	Inside window	3	>0 and <all	Lost dialog
5	Down & up	Inside window	3	all	Lost dialog
6	Down	Outside window	Don't care	>0	No effect

7	Down & up and then down & up	Inside window	2	0	Won dialog

Observed Results

Observed Results		
Test Case	**Expected Results**	**Observed Results**
1	Won dialog	Passed
2	Won dialog	Passed
3	Won dialog	Passed
4	Lost dialog	Passed
5	Lost dialog	Passed
6	No effect	Passed
7	Won dialog	Passed

Brickles Production Plan

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about the scope, see the Scope section.

The Brickles production plan describes how the Brickles product will be produced in the AGM product line. Product line organizations use production plans to capture how the product teams will build a new product, and a product-specific plan is developed before each product is constructed. This plan follows the outline provided by Chastek and McGregor [Chastek 02a].

Readership

The Brickles production plan provides a template for product-specific production plans and is intended primarily for product development teams. Managers can use the product-specific plan to determine the resources required to produce a product, and technical members can use it to actually produce a product.

Timeline

The production plan is a product of the Brickles product team. All base classes illustrated in Figure 58 have been implemented and are available to the product-specific development teams.

None of the Brickles-specific tasks have been performed.

We were planning to produce the Brickles product from these assets, and producing this production plan is part of the project planning.

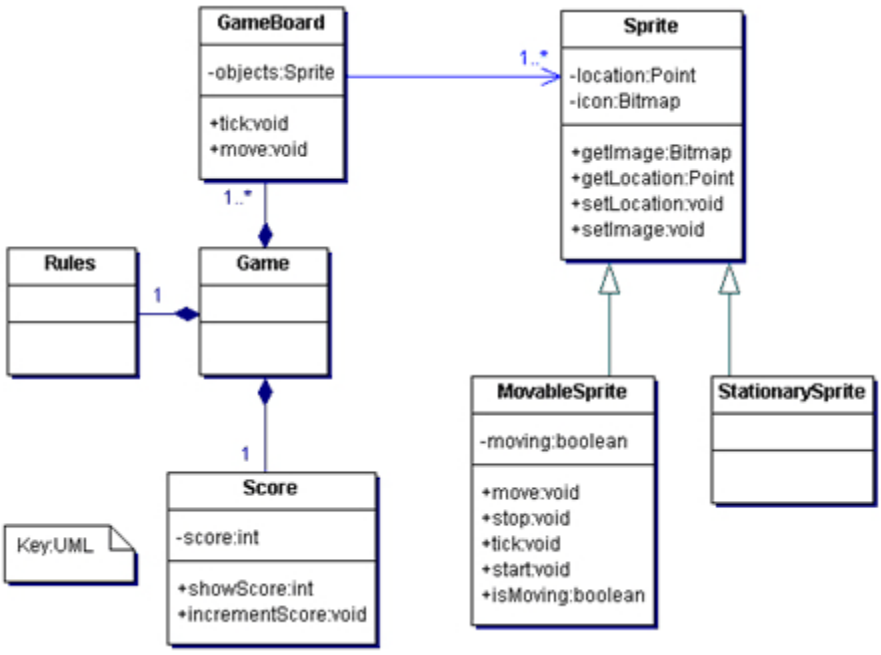

Figure 58: Base Classes

Strategic View of Product Development

Assumptions

Two broad assumptions apply:

1. The company has an existing pool of software developers who are highly technical. They have fielded products on a variety of hardware platforms and are accustomed to being involved down to the driver level of the software.
2. The product line contains games that are similar in content but that differ in platform. Differences in platform translate into differences in graphics implementation, which is a major feature of these products.

Qualities

We will briefly describe two types of properties: (1) product and (2) production process.

Product Qualities

Players will enjoy the games if they have a colorful display and realistic action.

- A new game element should add to the display quality and accurately portray the item it represents. Brickles will add three colorful graphics as game elements.
- Game action must be fast enough to demand the player's attention. If the number of elements slows the game, alternatives must be investigated. After all game elements are added, the Brickles system test will verify the game speed.

- Game action must meet the player's expectation. The motion and reactions of movable elements must be realistic. As elements are added to the game, their actions and boundaries must be correctly set through parameters so that collisions look real. The Brickles system test will verify that interactions of game elements look realistic.

Production Process Qualities

The production process in the AGM product line is largely manual. Developers are very technical, and the process allows for hands-on manipulation of the product. Providing a tool that automates the production process to a high degree would have frustrated developers and wasted development resources.

Products Possible from Available Assets

The products that are possible from the available assets are simple, animated games that involve moving and stationary objects. Each game has an area in which all play occurs and implements a set of physical rules that control movement. Using the incremental approach, the asset base is only developed to the extent needed to construct the current set of products. As additional increments are completed, the variety of possible games may increase.

Production Strategy

The production strategy includes a domain-based design approach and a manual construction approach. The requirements analysis and architecture development will be based on domain information. For each new product, software developers will manually specialize assets, gather other domain-based core assets, and then build an executable by running a compiler and linker.

Core assets are being built incrementally. Therefore, earlier product teams will have fewer available assets than later product teams. Earlier product teams (particularly the freeware development team) will identify core asset candidates for use in the later increments.

The formal statement of the strategy[11] is as follows:

We will position ourselves as the leading provider of rapidly customized, high-performance, low cost games by producing products that are easily modified, have better performance than our competitors, are sufficiently low cost to deter potential competitors from entering the market, and require sufficiently few resources to allow their use on any embedded computer. We will produce the initial products using a traditional iterative, incremental development process that uses a standard programming language, integrated development environment (IDE), and available libraries. We will create domain-based assets, including a product line architecture and software components, for the initial products in a manner that will support a migration to automatic generation of the second and third increment products.

---

11 For the complete rationale for this production strategy, see the "For the record" memorandum from the VPPD, Memo 04-01.

Overview of Available Core Assets

Source Code Naming Conventions

The following naming conventions are used:

- Game-specific classes begin with the name of the game and are created in a package devoted to that game.
- Interface definitions end in “Interface.”
- Class names begin with capital letters.
- Variable names begin with lowercase letters.
- DotUnit test case suite methods begin with “test.”

Analysis-Level Assets

The requirements document includes the following assets:

- The domain analysis model organizes the concepts in the main domain (games) and provides the attributes and relationships of each concept. Developers who are new to the domain should study this asset to understand the product’s background.
- The feature model shows the features of products.
- The use case model provides a superset of product requirements.
  Select the appropriate use cases for your product.

High-Level Design Assets

The main high-level design asset is the software architecture, which is described in the Arcade Game Maker Architecture Documentation Beyond Views and Arcade Game Maker Software Architecture Views sections. Figure 59 shows the architecture’s base classes. Product-specific versions of these classes should be defined at a high level in each product-specific production plan.

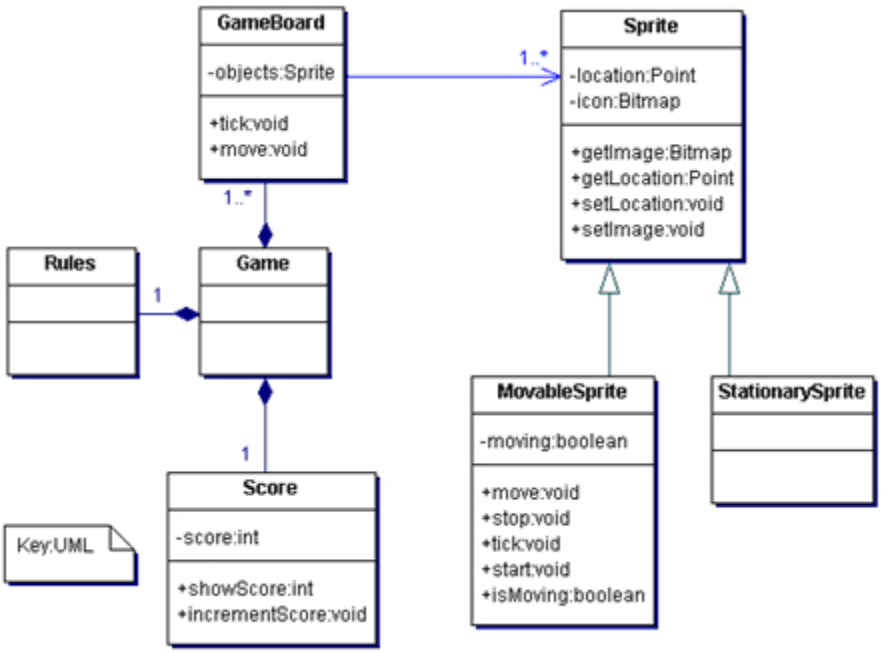

Figure 59: Base Classes

Source Code

Each interface in the existing architecture has been implemented. The C# components are named according to the guidelines in the Source Code Naming Conventions section.

Test Cases

This section describes *existing* test cases. Use an existing test case if it covers functionality or features included in the new product. Test cases are not yet available for all code assets.

Unit Tests

Individual unit tests are constructed using the DotNet testing framework and are available from the Unit Test Plans section. The source code is in the form of classes.

The following unit test classes are currently available:

- Velocity/Speed/Direction cluster (part of our pilot effort)

Integration Tests

If integrating units result in a component, test it at the API level. (If it results in a GUI, see the following System Test Plans section.)

The following integration tests are currently available:

- none

System Tests

System tests are currently done manually. Each system test is a scenario derived from a specific use case. When a use case is applied to more than one product, the related test cases can also be applied to that new product. These test cases are documented in the test plan for a product.

The following test plans are currently available:

- Brickles (created as part of the Brickles planning process in parallel with this document)

Inputs and Dependencies

Inputs

Game inputs are as follows:

- mouse and keyboard events
- names of the files that save the score and the state of the game (when the change cases for saving are implemented)

Dependencies

Game dependencies are as follows:

- graphics library of the programming language
- operating system (when the change case for saving a game is implemented)

Variations

Absorbing vs. Reflecting

The stationary game elements participate in the game by providing collision behavior. Core assets include two major behaviors:

1. A stationary element may reflect the movable element according to the laws of physics.
2. A stationary element may absorb the movable element so that it is deleted from the game.

A parameter on each element determines which of these behaviors is performed.

Event Handling

The event-handling routines vary from one game to another. An implementation of the EventHandlerDefinitions interface is provided as a parameter to the GameBoard component. Each of the mouse and keyboard events is handled as defined in the implementation of the parameter.

Detailed Production Process

Product teams will use the following four-step production process that was developed as Brickles and Pong freeware games were built:

1. identify the product, define it, and analyze it incrementally
2. design the product
3. build the product
4. test the product

Identify, Define, and Analyze the Product Incrementally

This step includes the following tasks:

1. Identify products. (Products are already identified. Because each product is a single game, this was accomplished when you identified games during the product line planning.)
2. Define the rules of the game. There are several versions of most of these games so the product team must first decide on a set of rules to implement.
3. Analyze features for the new game that are variations from previous games, and identify existing features that must change for this game.

Design the Product

This step includes the following three tasks:

1. Plan how to provide those features from existing components.
2. Plan how to provide the remaining features from new assets.
3. Design the new implementation of the EventHandlerDefinitions interface.

Build the Product

This step includes the following 10 tasks:

1. Start a new ClassLibrary in a Visual Studio Project using *Game_name* Definitions as its name. Use this class for all new classes except the game definition.
2. Start a new Windows Application in a Visual Studio Project using *Game_name*.
3. Copy form1.cs from a previous product line project.
4. Change the namespace name to *Game_name*.
5. Configure the new GameBoard.
6. Copy data.txt from a previous game's working directory. This is the resource file for the game.
7. Edit data.txt to reflect the new game.
8. Write the game-specific classes needed for the game.
9. Compile the resource file.
10. Copy the compiled resource file to the Debug directory.

Test the Product

This step includes the following six tasks:

1. Test each core asset, by inspection or execution, as it is created or revised.
2. If the asset is revised, revise the previous testing materials and reapply them to the new version of the asset.
3. For a code asset, code the unit test asset as a JUnit test class.
4. Revise the initial generic game system test set for each new game.
5. Create a game-specific system test set for each new game as shown in Figure 60.
6. Maintain system test cases as text documents and apply them manually.

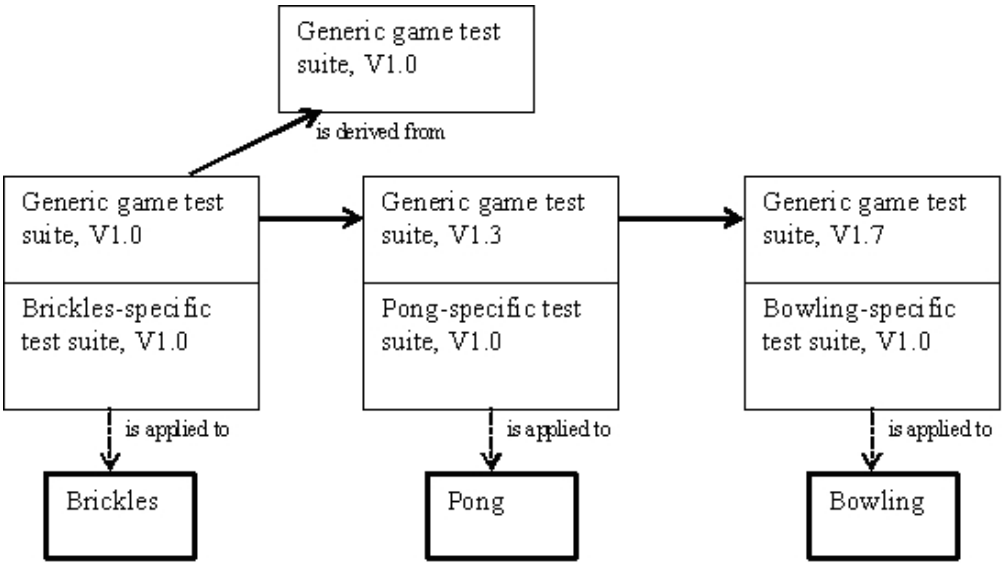

Figure 60: System Test Cases Related to Products

Tailoring the Production Plan to a Product-Specific Production Plan

Overall, the production plan template is very generic and applies to all products built using the current asset base. However, some parts of the plan must be modified for a specific product.

The next section, Management Information, includes the two most important and obvious sections that must be modified: the schedule and the bill of materials (BOM). Since we are using a manual product production approach, the schedule defines which personnel are needed when. The BOM provides a method for tracking the use of core assets.

Management Information

Schedule

Table 33 provides a schedule template that includes all the steps from the process described in the Detailed Production Process section. For the product-specific production plan, update the entries in the Performed By and Estimate/Date Performed columns.

Table 33: Schedule Template

Process Step	Product-Specific Task	Performed By	Estimate/Date Performed
Product Identification	Done during product planning	Product planning	Done
Product Definition	Define the rules of the new game.	Analyst (Gary)	≅ 0.5 days (8/5/03)
Product Analysis	Analyze the rules of the new game.	Analyst (Gary)	≅ 0.5 days (8/5/03)
Product Design	Identify new elements needed by the game.	Designer (Jason)	≅ 2 hours (8/6/03)
	Identify changes to existing elements.	Designer (Jason)	≅ 0.5 days (8/6/03)
	Design product-specific implementation of the game interface.	Designer (Jason)	≅ 2 hours (8/7/03)
	Design product-specific implementation of EventHandlerDefinitions.	Designer (Jason)	≅ 2 hours (8/7/03)
Product Build	Create new implementations and make changes to existing classes.	Developer (Bob)	≅ 0.5 days (8/10/03)
	Create new .Net projects for libraries and the application.	Developer (Bob)	≅ 0.5 days (8/10/03)
	Create the new make file.	Automated/developer	≅ 0.5 days (8/10/03)
Product Test	Create unit tests for new elements.	Developer (Bob)	≅ 0.5 days (8/10/03)
	Modify existing unit tests for existing elements.	Developer (Bob)	≅ 0.5 days (8/11/03)
	Execute unit tests.	Developer (Bob)	≅ 0.5 days (8/11/03)
	Modify/extend the product test suite.	Tester (John)	
	Execute the product tests.	Tester (John)	≅ 0.5 days (8/12/03)

Production Resources

The primary resources are the Visual Studio .Net environment and the organization's UML modeling tool.

During the analysis and design steps, extend the UML model to include any new elements that must be defined.

Use the .Net environment to create any new required components. After all components are created, use the environment to build an executable.

Bill of Materials (BOM)

The BOM allows you to track the use of core assets. Table 34 includes a high-level list of available code assets. For the product-specific plan, update the table with only those specific assets that will be used.

Table 34: Bill of Materials (BOM) Template

Component	Source	Cost
Product-specific Game component	In-house	$1,000
The generic GameBoard	In-house	$ 500
Required Sprites: Puck, Paddle, Brick, BrickPile, Wall, Ceiling, Floor	In-house	$ 700
Implementation of EventHandlerDefinitions interface: BricklesEventHandlerDefinitions	In-house	$ 600

The senior personnel are billed at $75/hour, and junior personnel are billed at $50/hour. In Table 33, there are 18 hours of senior analyst and designer times and 29 hours of junior developer and tester time. All costs are in-house components, so the BOM has a total cost of $2,800 for the non-core assets.

Product-Specific Details

The rules of the game are the most unique parts of the product. Distribute them across the EventHandlerDefinitions and the Game component, which is the container for the entire game implementation. Most of the rules enforced by Sprites are common across most (if not all) of the games. Develop the EventHandlerDefinitions component (like the Game component) specifically for a game.

Pay attention to the animation loop, which is the single most important product-specific detail in the Game class. It defines the game's sequence of events.

Metrics

Use the following two metrics to evaluate the product production process: (1) the number of new lines of code and (2) the number of unique lines of code.

New Lines of Code

This metric describes the number of lines of new code that must be written for this product. This new code may or may not be used in another product later. A high value indicates more effort required to produce the product and impacts cost and schedule.

Unique Lines of Code

This metric describes the percentage of the lines of code that are unique to this product. This metric changes over time: it can be difficult to predict which code gets reused or remains unique. A high value indicates less similarity between products and indicates a longer payback time.

Attached Processes[12]

Constructing the Production Plan

Figure 61 shows a high-level version of building a production plan. For details, review Chastek and McGregor's work [Chastek 02a].

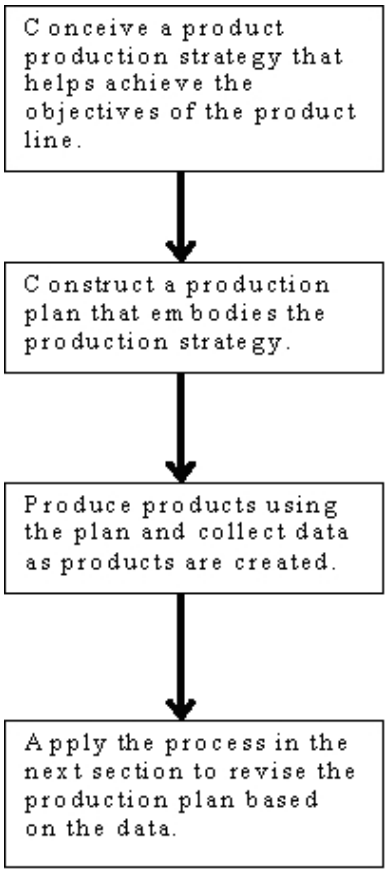

Figure 61: High-Level Process for Constructing a Production Plan

Changing the Production Plan

Figure 62 describes the change process. After each product is produced, data are collected and used to update the core asset base for the next increment's products. After those changes are reflected in the architecture documentation and the generic production plan, the plan is reviewed to identify inconsistencies between it and the core asset base. The core asset team member who owns the plan initiates the review.

---

12 This section is the "attached process" described by Clements and Northrop [Clements 02]. The process in Constructing the Production Plan defines how the production plan is initially built. The process in Changing the Production Plan focuses on modifying the existing production plan of the product line.

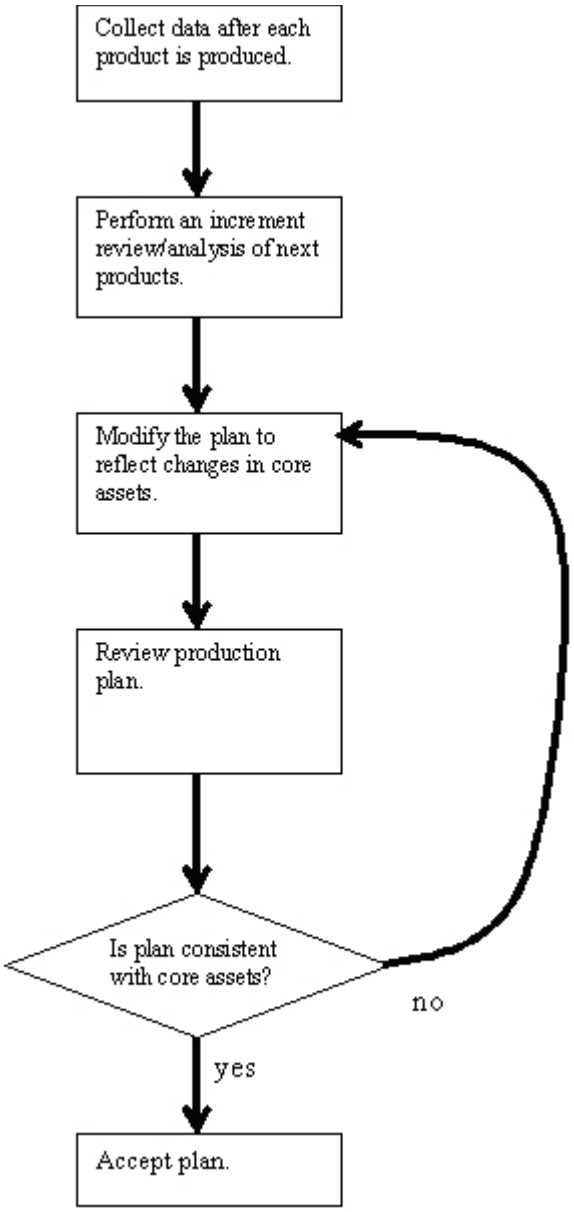

Figure 62: Process to Change the Production Plan

Brickles Wireless Tier Production Plan

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about the scope, see the Scope section.

This production plan describes how the Brickles wireless tier product will be produced in the AGM product line. Product line organizations use production plans to capture how the product teams will

build a new product, and a product-specific plan is developed before each product is constructed. This plan follows the outline provided by Chastek and McGregor [Chastek 02a].

Readership

This Brickles wireless tier production plan provides a template for product-specific production plans and is intended primarily for product development teams. Managers can use the product-specific plan to determine the resources required to produce a product, and technical members can use it to actually produce a product.

Timeline

This production plan is a product of the Brickles wireless tier product team. All of the base classes illustrated in Figure 63 have been implemented and are available to the product-specific development teams.

The Brickles-specific tasks had been performed for the freeware tier product. None of the product-specific tasks have been carried out yet for the wireless tier.

We were planning to produce the Brickles product for the wireless tier from these assets. Unfortunately, the freeware tier products used C#, which has no Mobile Information Device Profile (MIDP) API and forced us to switch to Java. Doing so required porting the code assets to Java and revising the Java APIs.

Producing this production plan is part of the project planning.

Strategic View of Product Development

Assumptions

Two broad assumptions apply:

1. The company has an existing pool of software developers who are highly technical. They have fielded products on a variety of hardware platforms and are accustomed to being involved down to the driver level of the software.
2. The product line contains games that are similar in content but that differ in platform. Differences in platform translate into differences in graphics implementation, which is a major feature of these products.

Qualities

We will briefly describe two types of properties: (1) product and (2) production process.

Product Qualities

Players will enjoy the games if they have a colorful display and realistic action.

- A new game element should add to the display quality and accurately portray the item it represents. Brickles will add three colorful graphics as game elements.

- Game action must be fast enough to demand the player's attention. If the number of elements slows the game, alternatives must be investigated. After all game elements are added, the Brickles system test will verify the game speed.
- Game action must meet the player's expectation. The motion and reactions of movable elements must be realistic. As elements are added to the game, their actions and boundaries must be correctly set through parameters so that collisions look real. The Brickles system test will verify that interactions of game elements look realistic.

Production Process Qualities

The production process in the AGM product line is largely manual. Developers are very technical, and the process allows for hands-on manipulation of the product. Providing a tool that automates the production process to a high degree would have frustrated developers and wasted development resources.

Products Possible from Available Assets

The products that are possible from the available assets are simple, animated games that involve moving and stationary objects. Each game has an area in which all play occurs and implements a set of physical rules that control movement. Using the incremental approach, the asset base is only developed to the extent needed to construct the current set of products. As additional increments are completed, the variety of possible games may increase.

Production Strategy

The production strategy includes a domain-based design approach and a manual construction approach. The requirements analysis and architecture development will be based on domain information. For each new product, software developers will manually specialize assets, gather other domain-based core assets, and then build an executable by running a compiler and linker.

Core assets are being built incrementally. Therefore, earlier product teams will have fewer available assets than later product teams. Earlier product teams (particularly the freeware development team) will identify core asset candidates for use in the later increments.

The formal statement of the strategy[13] is as follows:

> *We will position ourselves as the leading provider of rapidly customized, high-performance, low-cost games by producing products that are easily modified, have better performance than our competitors, are sufficiently low cost to deter potential competitors from entering the market, and require sufficiently few resources to allow their use on any embedded computer. We will produce the initial products using a traditional iterative, incremental development process that uses a standard programming language, integrated development environment (IDE), and available libraries. We will create domain-based assets, including a product line*

---

13 For the complete rationale for this production strategy, see the "For the record" memorandum from the VPPD, Memo 04-01.

architecture and software components, for the initial products in a manner that will support a migration to automatic generation of the second and third increment products.

Overview of Available Core Assets

Source Code Naming Conventions

The following naming conventions are used:

- Game-specific classes begin with the name of the game and are created in a package devoted to that game.
- Interface definitions end in “Interface.”
- Class names begin with capital letters.
- Variable names begin with lowercase letters.
- JUnit test case suite methods begin with “test.”

Analysis-Level Assets

The requirements document includes the following assets:

- The domain analysis model organizes the concepts in the main domain (games) and provides the attributes and relationships of each concept. Developers who are new to the domain should study this asset to understand the product’s background.
- The feature model shows the features of products.
- The use case model provides a superset of product requirements. Select the appropriate use cases for your product.

High-Level Design Assets

The main high-level design asset is the software architecture, which is described in the Arcade Game Maker Architecture Documentation Beyond Views and Arcade Game Maker Software Architecture Views sections. Figure 63 shows the architecture’s base classes. Product-specific versions of these classes should be defined at a high level in each product-specific production plan.

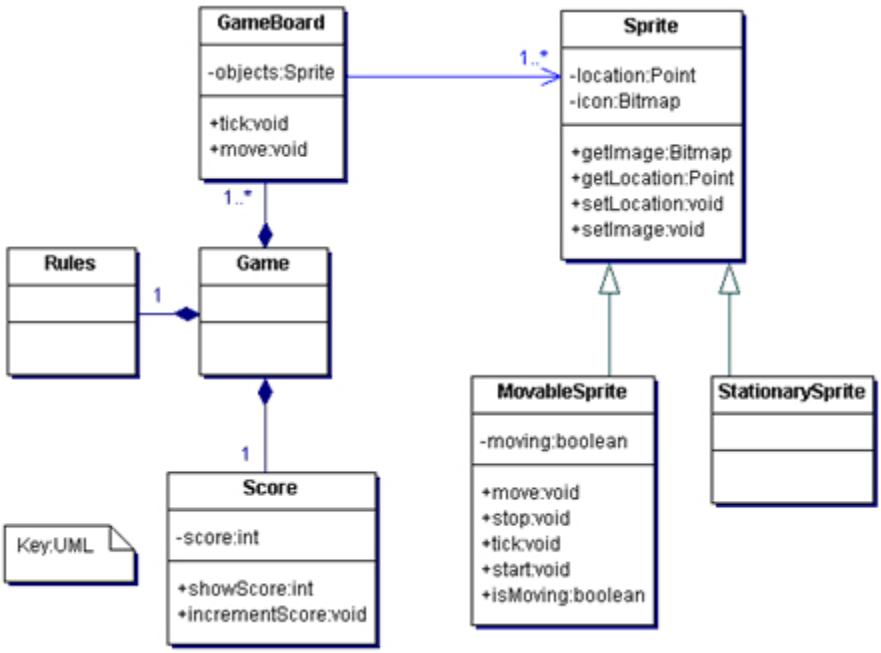

Figure 63: Base Classes

Source Code

Each interface in the existing architecture has been implemented. The C# components are named according to the guidelines in Source Code Naming Conventions section.

Test Cases

This section describes *existing* test cases. Use an existing test case if it covers functionality or features included in the new product.

Test cases are not yet available for all code assets.

Unit Tests

Individual unit tests are constructed using the DotNet testing framework and are available from the Unit Test section. The source code is in the form of classes.

The following unit test classes are currently available:

- Velocity/Speed/Direction cluster (part of our pilot effort)

Integration Tests

If integrating units result in a component, test it at the API level. (If it results in a GUI, see the following System Tests section.)

The following integration tests are currently available:

- none

System Tests

System tests are currently done manually. Each system test is a scenario derived from a specific use case. When a use case is applied to more than one product, the related test cases can also be applied to that new product. These test cases are documented in the test plan for a product.

The following test plans are currently available:

- Brickles (created as part of the Brickles planning process in parallel with this document)

Inputs and Dependencies

Inputs

Game inputs are as follows:

- stylus events (this is a variation point among products: in the freeware tier products, events were mouse and keyboard)
- names of the files that save the score and the state of the game (when the change cases for saving are implemented)

Dependencies

Game dependencies are as follows:

- graphics library of the programming language (choosing one was problematic: the wireless tier must use the MIDP API, which is different from that of standard C# or Java)
- MIDP and subsequently Palm operating system
- operating system (when the change case for saving a game is implemented)

Variations

Absorbing vs. Reflecting

The stationary game elements participate in the game by providing collision behavior. Core assets include two major behaviors:

1. A stationary element may reflect the movable element according to the laws of physics.
2. A stationary element may absorb the movable element so that it is deleted from the game.

A parameter on each element determines which of these behaviors is performed.

Event Handling

The event-handling routines needed vary from one game to another and from one tier to another.

For the wireless tier, a set of Palm hardware events is handled using the WABI library for Java.

Detailed Production Process

Product teams will use the following four-step production process that was developed as Brickles and Pong freeware games were built:

1. identify the product, define it, and analyze it incrementally
2. design the product
3. build the product
4. test the product

Identify, Define, and Analyze the Product Incrementally

This step includes the following tasks:

1. Identify products. (Products are already identified. Because each product is a single game, this task was accomplished when you identified games during the product line planning.)
2. Define the rules of the game. There are several versions of most of these games, so the product team must first decide on a set of rules to implement.
3. Analyze features for the new game that are variations from previous games, and identify existing features that must change for this game.

Design the Product

This step includes the following three tasks:

1. Plan how to provide those features from existing components.
   Port from C# to Java, class by class, and then replace C#-specific APIs with Java APIs.
2. Plan how to provide the remaining features from new assets.
   Derive newly required assets from the newly ported Java assets.
3. Design the new implementation of the EventHandlerDefinitions interface.
   Replace mouse and keyboard events with the stylus event handlers.

Build the Product

This step includes the following nine tasks:

1. Start a new Eclipse project using Game_name Definitions as its name.
   Use this class for all new classes except the game definition.
2. Start a new class using Game_name.
3. Change the namespace name to Game_name.
4. Configure the new GameBoard.
5. Copy data.txt from a previous game's working directory.
   This is the resource file for the game.
6. Edit data.txt to reflect the new game.

7. Write the game-specific classes needed for the game.
8. Compile the resource file.
9. Copy the compiled resource file to the directory containing the classes.

Test the Product

This step includes the following six tasks:

1. Test each core asset, by inspection or execution, as it is created or revised.
2. If the asset is revised, revise the previous testing materials and reapply them to the new version of the asset.
3. For a code asset, code the unit test asset as a JUnit test class.
4. Revise the initial generic game system test set for each new game.
5. Create a game-specific system test set for each new game as shown in Figure 64.
6. Maintain system test cases as text documents and apply them manually.

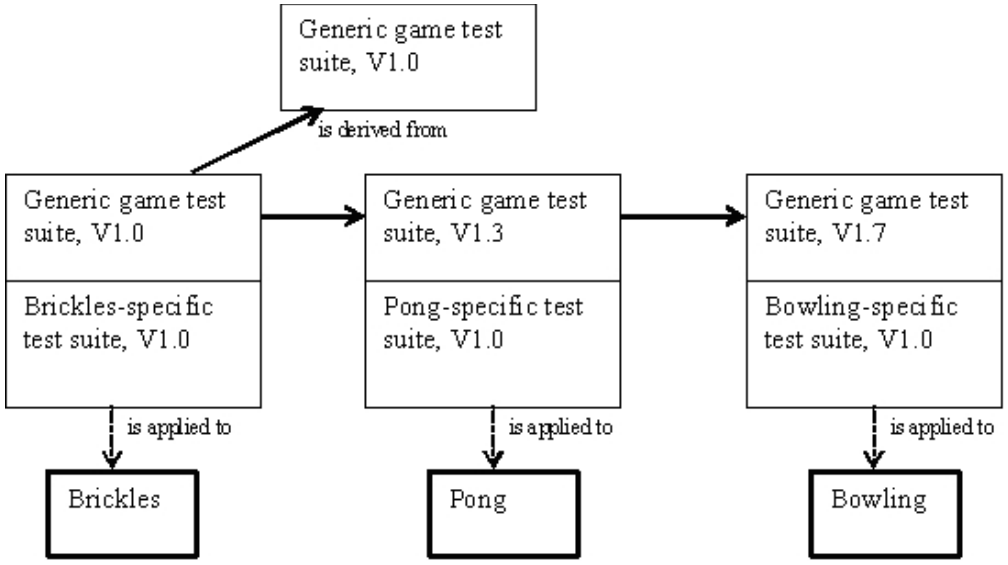

Figure 64: System Test Cases Related to Products

Tailoring the Production Plan to a Product-Specific Production Plan

This is the specialized version of the generic production plan. The details below are for informational purposes only.

Overall, the production plan template is very generic and applies to all products built using the current asset base. However, some parts of the plan must be modified for a specific product.

The next section includes the two most important and obvious sections that must be modified: the schedule and the bill of materials (BOM). Since we are using a manual product production approach,

the schedule defines which personnel are needed when. The BOM provides a method for tracking the use of core assets.

Management Information

Schedule

Table 35 provides a schedule template that includes all the steps from the process described in the Detailed Production Process section. For the product-specific production plan, update the entries in the Performed By and Estimate/Date Performed columns.

Table 35: Schedule Template

Process Step	Product-Specific Task	Performed By	Estimate/Date Performed
Product Identification	Done during product planning	Product planning	Done
Product Definition	Define the rules of the new game.	Analyst (Gary)	≅ 0.5 day (10/5/04)
Product Analysis	Analyze the features.	Analyst (Gary)	≅ 0.5 day (10/5/04)
Product Design	Identify new elements needed by the game.	Designer (Jason)	≅ 2 hours (10/6/04)
	Identify changes to existing elements.	Designer (Jason)	≅ 6 hours (10/6/04)
	Design product-specific implementation of the game interface.	Designer (Jason)	≅ 1 day (10/7/04)
	Design product-specific implementation of EventHandlerDefinitions.	Designer (Jason)	≅ 1 day (10/8/04)
Product Build	Create new implementations and make changes to existing classes.	Developer (Bob)	≅ 2 days (10/9/04)
	Create new Eclipse projects for libraries and the application.	Developer (Bob)	≅ 0.5 hour (10/10/04)
	Create the new make file.	Automated/developer	≅ 2 days (10/10/04)
Product Test	Create unit tests for new elements.	Developer (Bob)	≅ 2 days (10/11/04)
	Modify existing unit tests for existing elements.	Developer (Bob)	≅ 1 day (10/13/04)
	Execute unit tests.	Developer (Bob)	≅1 day (10/14/04)
	Modify/extend the product test suite.	Tester (John)	≅1 day (10/15/04)
	Execute the product tests.	Tester (John)	≅ 1 day (10/16/04)

Production Resources

The primary resources are the Eclipse JDT environment and the organization’s UML modeling tool.

During the analysis and design steps, extend the UML model to include any new elements that must be defined.

Use the Eclipse environment to create any new required components. After all components are created, build an executable.

Bill of Materials (BOM)

The BOM allows you to track the use of core assets. Table 36 includes a high-level list of available code assets. For the product-specific plan, update the table with only those specific assets that will be used.

Table 36: Bill of Materials (BOM) Template

Component	Source	Cost
Product-specific Game component	In-house	$1,000
Generic GameBoard	In-house	$ 725
Required Sprites: Puck, Paddle, Brick, BrickPile, Wall, Ceiling, Floor	In-house	$1,950
Implementation of EventHandlerDefinitions interface: BricklesEventHandlerDefinitions	In-house	$2,050

The senior personnel are billed at $75/hour, and junior personnel are billed at $50/hour. In Table 35, there are 32 hours of senior analyst and designer times and 66.5 hours of junior developer and tester time. All costs are in-house components, so the BOM has a total cost of $5,725 for the assets needed for this product.

Product-Specific Details

The rules of the game are the most unique parts of the product. Distribute them across the EventHandlerDefinitions and the Game component, which is the container for the entire game implementation. Most of the rules enforced by Sprites are common across most (if not all) of the games. Develop the EventHandlerDefinitions component (like the Game component) specifically for a game.

Pay attention to the animation loop, which is the single most important product-specific detail in the Game class. It defines the game's sequence of events.

Metrics

Use the following two metrics to evaluate the product production process: (1) the number of new lines of code and (2) the number of unique lines of code.

New Lines of Code

This metric describes the number of lines of new code that must be written for this product. This new code may or may not be used in another product later. A high value indicates more effort required to produce the product and impacts cost and schedule.

Unique Lines of Code

This metric describes the percentage of the lines of code that are unique to this product. This metric changes over time: it can be difficult to predict which code gets reused or remains unique. A high value indicates less similarity between products and indicates a longer payback time.

Attached Processes[14]

Constructing the Production Plan

Figure 65 shows a high-level version of building a production plan. For details, review Chastek and McGregor's work [Chastek 02a].

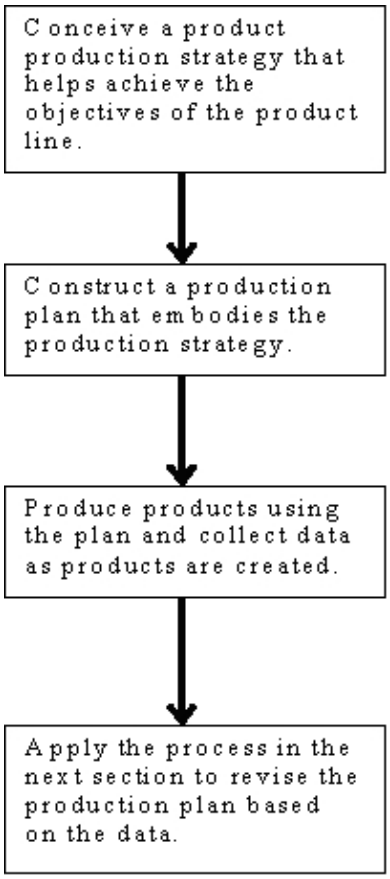

Figure 65: High-Level Process for Constructing a Production Plan

---

[14] This section is the "attached process" described by Clements and Northrop [Clements 02]. The process in the Constructing the Production Plan section defines how the production plan is initially built. The process in the Changing the Production Plan section focuses on modifying the existing production plan of the product line.

Changing the Production Plan

Figure 66 describes the change process. After each product is produced, data are collected and used to update the core asset base for the next increment's products. After those changes are reflected in the architecture documentation and the generic production plan, the plan is reviewed to identify inconsistencies between it and the core asset base. The core asset team member who owns the plan initiates the review.

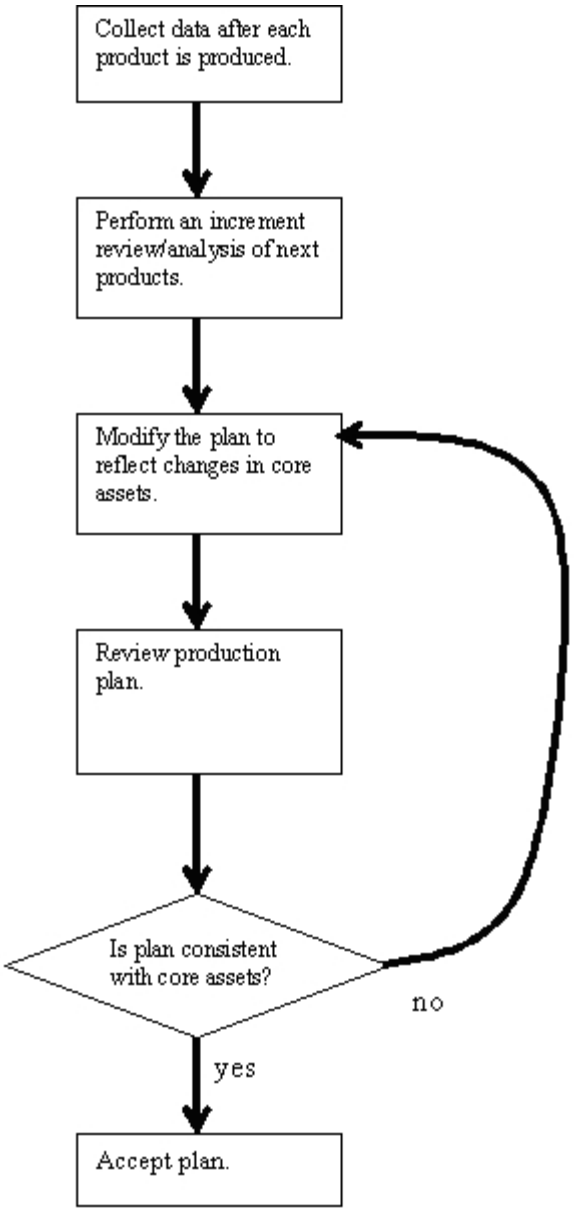

Figure 66: Process to Change the Production Plan

Arcade Game Maker Pedagogical Product Line: Pong Product

The Arcade Game Maker (AGM) product line organization is producing a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about the products, see the Scope section.

Readership

The Pong User Manual is for game players. It describes how to install and play Pong.

Install Pong

Install Pong as follows:

1. Unzip the distribution file.
2. Provide access to the .Net runtime environment for the executable.
3. Double-click the Pong icon.
   The game is opened and initialized as shown in Figure 67.

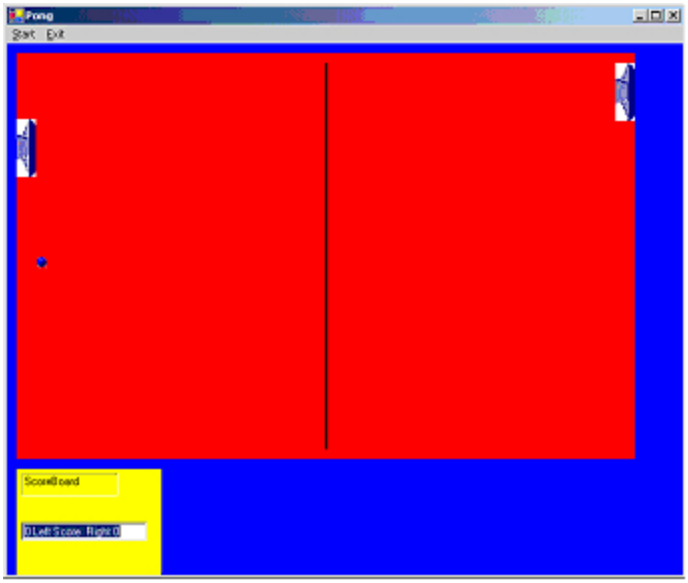

Figure 67: Initialized Pong Display

Modes of Operation

Pong has three modes of operation:

1. Initialized: The executable is loaded and the display is initialized.
   To enter this mode, double-click the Pong icon.
2. Playing: Animation begins (elements move around the screen).
   To enter this mode, left-click within the boundaries of the playing area.

3. Paused: Animation stops (all elements are stationary).
   To enter this mode, hold down the left mouse button. To resume play, release the button.

Rules for Pong

This version of Pong is for one player. It has a few simple rules:

- The objective is to keep the puck in play as long as possible.
- The player keeps the puck in play by moving the mouse to control paddles at both ends of the playing field.
- The player controls one paddle (the one in the same half of the playing field as the mouse icon) at a time.
- The puck bounces off the paddle and the top and bottom sides of the playing field.
- The puck is absorbed by the left and right sides of the playing field.

Test Pong

Test Pong as follows:

1. If you closed the game, double-click on the Pong icon.
2. Left-click within the boundaries of the playing area.
   Animation begins.
3. Let the puck collide into the walls.
   Based on the rules, the puck is absorbed or changes direction according to the laws of physics.
4. Hold down the left mouse button.
   The game is paused.
5. Release the left mouse button.
   The game is resumed.
6. On the toolbar, click Help.
   Online help is displayed.
7. On the toolbar, click File > Exit.
   The game exits.

Advanced Features

No advanced features are implemented in this version.

Arcade Game Maker Pedagogical Product Line: Bowling Product

The Arcade Game Maker (AGM) product line organization is producing a series of arcade games ranging from low to high obstacle count with a range of interaction effects. For more details about the products, see the Scope section.

Readership

The Bowling User Manual is for game players. It describes how to install and play Bowling.

Install Bowling

Install Bowling as follows:

1. Unzip the distribution file.
2. Provide access to the .Net runtime environment for the executable.
3. Double-click the Bowling icon.
   The game is opened and initialized as shown in Figure 68.

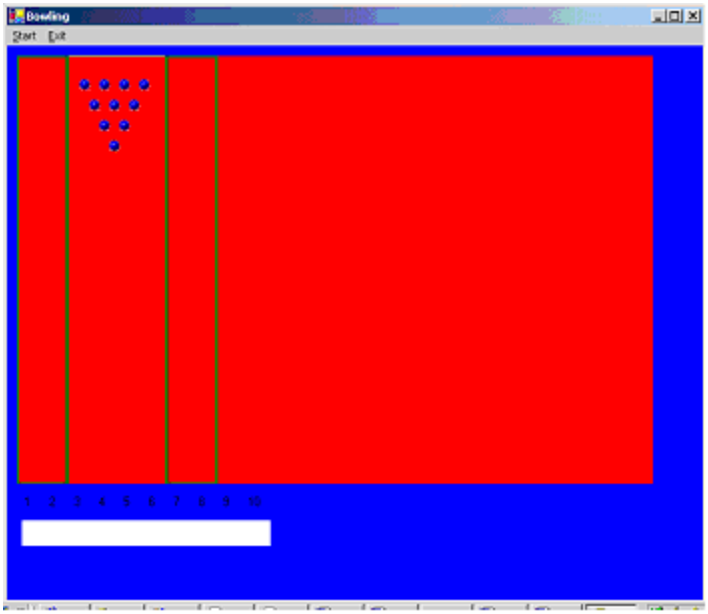

Figure 68: Initialized Bowling Display

Modes of Operation

Bowling has three modes of operation:

1. Initialized: The executable is loaded and the display is initialized. To enter this mode, double-click the Bowling icon.

2. Playing: Animation begins (elements move around the screen). To enter this mode, position the cursor at the point from which you want to launch the bowling ball. To release the ball, left-click.
3. Paused: Animation stops (all elements are stationary). To enter this mode, hold down the left mouse button. To resume play, release the button.

Rules for Bowling

This version of Bowling is for one player. It has a few simple rules:

- The player attempts to knock down as many pins as possible in 10 frames.
- The player throws two balls in each frame.
- Knocking down all pins in a frame with the first ball is termed a strike.
- Knocking down all pins in a frame with both balls is termed a spare.
- The score for a frame is the total number of pins knocked down by the two balls.
- The score for a frame in which a spare is made is 10 plus the number of pins knocked down by the first ball of the next frame.
- The score for a frame in which a strike is made is 10 plus the number of pins knocked down by both balls in the next frame.
- If a strike or spare is achieved in the 10th frame, a third ball is thrown, and the number of pins knocked down is added to the score for that frame.

Test Bowling

Test Bowling as follows:

1. If you closed the game, double-click on the Bowling icon.
2. Position the cursor at the point from which you want to launch the bowling ball and left-click.
   Animation begins.
3. Let the ball collide into the pins.
   Based on the rules, the score is determined.
4. Hold down the left mouse button.
   The game is paused.
5. Release the left mouse button.
   The game is resumed.
6. On the toolbar, click Help.
   Online help is displayed.
7. On the toolbar, click File > Exit.
   The game exits.

Advanced Features

No advanced features are implemented in this version.

Appendix A: Acronym List and Glossary

ACB	Architecture Control Board
AGM	Arcade Game Maker
application engineering	discipline of constructing a product using the product line architecture and selecting from the inventory of available product line components based on the production plan
ATAM	Architecture Tradeoff Analysis Method®
CCB	Configuration Change Board
CEO	chief executive officer
CLR	common language runtime
core asset	reusable artifacts and resources that form the basis for the software product line
CM	configuration management
DHR	director of human resources
domain engineering	discipline of creating, acquiring, and managing the resources needed to build a set of products
FODA	feature-oriented domain analysis
IDE	integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
MVC	model-view-controller
MIDP	Mobile Information Device Profile
PLM	product line manager
product	any saleable or usable output produced from the product line core assets
RUP	Rational Unified Process
software product line	group of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission and that are developed from a common set of core assets in a prescribed way
TE	test effectiveness
UML	Unified Modeling Language
VC	version control
VPBA	vice president for business affairs
VPPD	vice president for product development
VPPP	vice president for product planning
WAP	wireless access protocol

Appendix B: Memos

Arcade Game Maker Memo 04-01

To: For the record

From: Vice President for Product Development

CC: Vice President for Product Planning

Date: March 3, 2004

Re: Production Strategy

Strategy development is not an exact science. To ensure clear communication, this memo captures the rationale behind our product production strategy.

The strategy is derived by considering the goals of the product line and its identified qualities and available technologies. One of the inputs to this process is Porter's Five Forces Model, shown in Figure 69. This model allowed us to structure our reasoning about strategy.

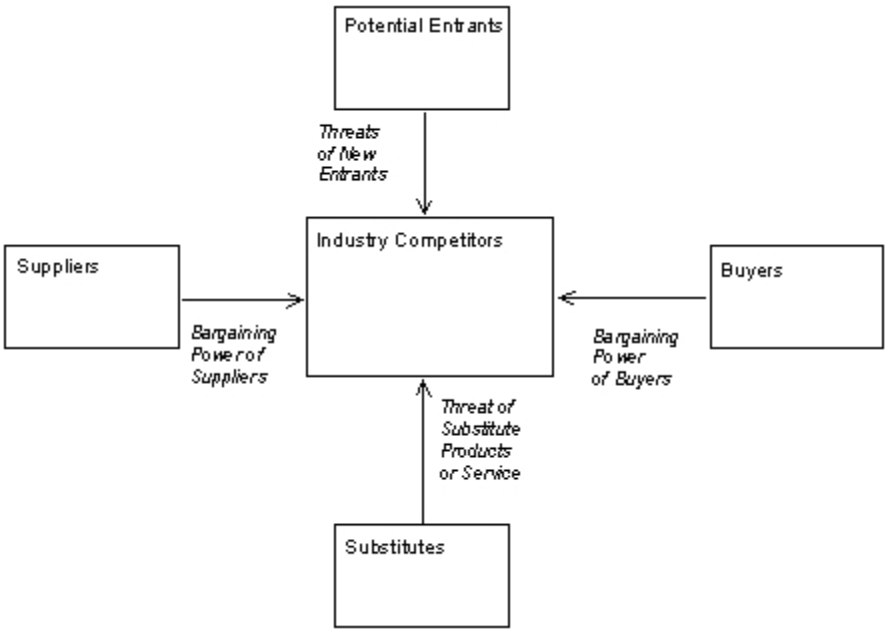

Figure 69: Porter's Five Forces Model

Each box in the graphic represents a force on the production strategy that a company must resolve at a strategic level. Each force may vary across products or within a product across production increments.

Goals

Goals shown in Figure 70 were identified in the product line business case. They were listed in the corporate strategic plan and identified by considering the Industry Competitors, Substitutes, and Possible Entrants forces.

Qualities

Quality attributes shown in this figure are influenced primarily by game buyers. Cell phone companies purchasing the second iteration of products require low resource consumption. Qualities were identified by considering the Industry Competitors and Potential Entrants forces. To stay competitive, we must be able to update our products quickly.

Technologies

Technologies used to produce the products are influenced by what is available from suppliers. For example, several game engines (essentially platforms upon which games can be built) are available. Suppliers provide tools like code generators (e.g., Eclipse Foundation's Java Emitter Templates [JET]) that allow products to be generated from models. For wireless devices, buyers drive the technology choice because our products will be integrated into their devices.

Goals	Qualities	Technologies
• Improve market position	• Modifiable	• ~~Game engine, custom additions~~
• Improve time to market	• Improved performance	• Generative programming
• Increase productivity	• Low cost	• Programming environments such as the .Net suite
• Enable mass customization of products	• Low resource consumption	

Figure 70: Strategy Components

Strategy Formation

The production strategy is the high-level notion of how products will be developed. It determines some core asset features. The strategy is formed by considering each element in Figure 69 and ensuring that the columns are mutually supportive and consistent.

Tradeoffs were made to reach a consistent state. For example, the use of game engines is not consistent with low resource consumption, so the game engines choice is deleted from the technologies list. The resulting lists, shown in Figure 70, form the basis for the strategy.

We considered the fit among table elements, especially the technologies. Generative programming requires a basic set of components from which products can be composed. The use of a traditional programming environment to produce software assets that then became the set of components shows a reasonable fit between traditional development and generative programming.

Our final production strategy can be summarized as follows:

We will position ourselves as the leading provider of rapidly customized, high-performance, low-cost games by producing products that are easily modified, have better performance than our competitors' products, are sufficiently low cost to deter potential entrants from entering the market, and require sufficiently few resources to allow their use on any embedded computer. We will produce the initial products using a traditional iterative, incremental development process and a standard programming language, IDE, and available libraries. We will create domain-based assets (including a product line architecture and software components) for the initial products in a manner that will support a migration to automatic generation of second and third increment products.

Arcade Game Maker Memo 04-02

To: For the record

From: Vice President for Product Development

CC:

Date: May 12, 2004

Re: Homogeneity of the AGM Product Line

The intent of a product line is to achieve strategic levels of reuse. This is possible if the products in the product line share sufficient commonality. As we reviewed our product line planning, we found a metric[15] for computing homogeneity among products in a product line:

$$\text{homogeneity} = 1 - \frac{\sum_{i=1}^{\text{Number of products}} |R_{U_i}|}{|R_T|}$$

where R_U are the requirements unique to the *i*th product and R_T is any product line requirement

This metric is computed based on the use case model found in the Requirements section. With 12 total use cases and 3 unique use cases, the metric is computed as follows:

$$\text{homogeneity} = 1 - \frac{3}{12} = .75$$

---

15 Clements, Paul; McGregor, John D.; & Cohon, Sholom G. The Structured Intuitive Model for Product Line Economics (CMU/SEI-2005-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

This computation of .75 shows that commonality is high for the product line. However, we have been treating our product line as three sets of three products. No use cases show the differences among the three sets. If we add three functional sub-use cases, one for each platform on which the games will run, the computation can be reframed as follows:

$$\text{homogeneity} = 1 - \frac{3}{15} = .8$$

Arcade Game Maker Memo 04-03

To: For the record

From: AGM Product Line Manager

CC: Vice President for Product Development

Date: May 22, 2004

Re: Business Model Update and Decision Support

Since we developed our original business case, a seminal paper[16] was published on a business model that includes the equation below. We used this model to reanalyze our business case and quantify some difficult business decisions.

$$C_{org}() + C_{cab}() + \sum_{i=1}^{n} (C_{unique}(product_i) + C_{reuse}(product_i))$$

In this memo, we analyze and apply the equation to illustrate how it supports a fundamental decision made by our product line organization.

Equation's Fixed Costs

C_{org} is the cost of readying the organization for the product line strategy. It appears that our original estimate of $30,000 was accurate.

C_{cab} is the cost of the core assets. Because the products are being produced in three groups of three products, any core assets that apply to the second and third sets do not have to be available until that

---

16 Bockle, G.; Clements, P.; McGregor, J. D.; Muthig, D.; & Schmid, K. "Calculating ROI for Software Product Lines." IEEE Software 21, 3 (May/June 2004): 23-31.

increment of the product line is ready to be produced. Therefore, we have two choices: we can produce all of the core assets at once up front, or we can produce them incrementally, just in time. In this memo's Development Decision section, we explain how this model helped us make that decision.

Equation's Per Product Costs

C_{unique} is the cost of constructing the unique portion of each product. In Arcade Game Maker Memo 04-02, we used a new, unpublished metric to estimate a homogeneity value of .8 for our products.[17] Based on our original estimates of .3, .6, and 1.0 for our three product groups, we can now estimate savings of $1,250,000 as opposed to $1,000,000. This estimate is based on a raw homogeneity value that has not been calibrated to our development environment.

C_{reuse} is a function of the quality of the core assets: it is the cost of reusing the core assets in the context of the product. Because the cost of building a reusable asset (C_{reuse}) is often estimated at 1.5 times the cost of building for a specific use (C_{cab}), C_{reuse} decreases to some extent as C_{cab} increases. Further investigation of this relationship would allow us to determine what level of development for reuse produces the best return.

Development Decision

As stated above, we could build all of the core assets at one time before building any products, or we could build only those assets required for the next increment.

If we build all assets at once, some remain unused until the specific increment is built. Therefore, the value of C_{cab} must include an opportunity cost, which is based on the value of having resources used to create the core assets available for other work during that time (see Figure 71). In other words, are some developers doing work that will have to wait if they are diverted to core asset development, and if so, what is the value of that work?

The model allows us to select the implementation of C_{cab}, giving us the opportunity to represent the two approaches and compare the costs. We can assume that it will cost the same to implement the assets at either point. Thus, if the opportunity costs are greater than zero, the choice is obvious–build just in time. If opportunity costs are zero, it doesn't matter.

Our opportunity costs are greater than zero. We don't have enough developers to construct all the core assets up front. But no more developers will be available later, so we will build assets continuously rather than just before the next increment.

---

[17] The accuracy of this estimate depends on how similar the use cases are in terms of the amount of effort required to implement them (i.e., as the differences in use cases increase, the accuracy of the estimate decreases).

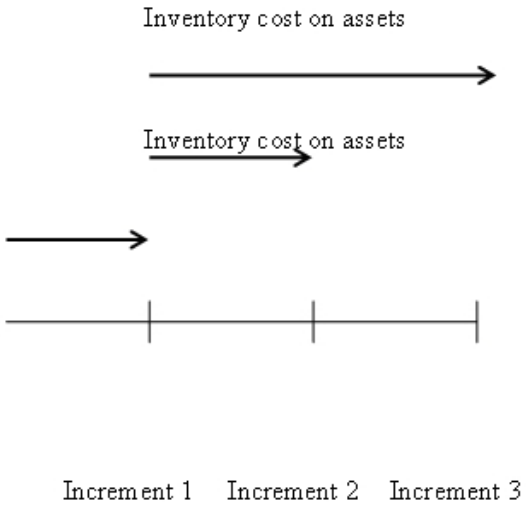

Figure 71: Cost of Inventory

Arcade Game Maker Memo 04-04

To: CEO

From: Vice President for Product Development

CC: Vice President for Product Planning

Date: August 8, 2004

Re: Variation Point Difficulty

We recently discovered a problem with the AGM product line. A variation point that was not identified in our original analysis requires the reworking of several core assets, and the schedule has slipped. Fortunately, we still have six products to produce. If the usual figures of payoff after three products hold, we will still be much better off having used the product line approach than a one-off approach. We can revise the business case if you want to clarify this issue. Details for the new variation point are included below.

History

Originally, the variation analysis was conducted assuming that products would be manufactured using the Java language. Later, we switched to C# but did not revisit the variation analysis to determine if there were any implications. There were some, but they were not immediately obvious. Construction of the freeware tier of products proceeded with no difficulties. But when construction of the wireless products began, the implications became obvious.

Wireless Products

Our products will operate on wireless devices that use the Mobile Information Device Profile (MIDP). However, we discovered that the MIDP-defined API for products running on the device did not bind

to C#, so we switched back to Java. Switching languages required relatively simple revisions to code assets, but then we also discovered that not all devices that support MIDP support floating point arithmetic as well. This restricted MIDP API forced us to switch graphics libraries as well, which required even more extensive revisions to the code assets than we initially anticipated.

Lessons Learned

We have learned two important lessons. First, do not rely on the implementation language to encapsulate the variation in platforms. If the product mix includes devices with which the team is not familiar, there should be an abstraction layer that could be replaced during implementation but that conceptually separates the product functionality from platform concerns. Second, analyses of any type should be revisited, particularly when a decision (such as making a language change) is made that has far-reaching implications. Currently, once an analysis is completed and the report is delivered, it is not clear who has responsibility for the analysis and accompanying decisions or that there is anything to have responsibility for. We can address this confusion by treating an analysis as a working document and identifying an owner who would periodically revise it to reflect current conditions. Unless you object, I will make that policy official and assign owners to the analyses that have been conducted so far.

Arcade Game Maker Memo 04-05

To: Vice President for Product Development

From: Director, Human Resources

CC: Vice President for Product Planning

Date: September 9, 2004

Re: Training Plan

The newly funded product line organization will introduce several new ideas and techniques to AGM, and this memo describes how AGM personnel in the new organization will be trained. This training plan includes activities that are specifically related to the product line strategy: it does not include normal, ongoing training activities (e.g. tool-specific activities). It also describes courses, workshops, and conference tutorials that will be used to thoroughly acclimate personnel to the new strategy.

Courses

Training courses are modeled after the SEI Software Product Line Curriculum. Course descriptions are provided below, and Table 37 shows the courses that the individuals in each type of role should take.

Product Line Overview. This course provides a general overview of the principles and practices of the product line strategy.

Product Line Adoption. This course describes the actions necessary to take an existing organization and transition to the product line strategy.

Developing Software Product Lines. This course provides hands-on exposure to the practices necessary to develop core assets and products in a product line organization.

Project Management. This course provides the specifics of project management for managers who will be responsible for the unique projects in the product line organization. This course is for managers who will lead projects as well as those responsible for chartering projects.

Table 37: Courses and Roles

Course	Software Engineering		Technical Management	Organizational Management
	Team Lead	Team Member	Functional Team Leads, Product Line Manager	CEO, VPPD, VPPP, DHR
Product Line Overview	X	X	X	X
Product Line Adoption			X	X
Developing Software Product Lines	X	X		
Project Management	X		X	X

Workshops

After most courses are completed, we'll offer a series of full-day workshops. Each one will focus on a single topic, and most will be centered on a practice area in the SEI *Framework for Software Product Line Practice*.[18] Workshops will be led by consultants or staff members and will include practical hands-on activities in product line analysis or product production planning.

Conference Tutorials

If a few AGM personnel (workshop leaders, other individuals, or small groups) require specialized training, they will be sent to conferences. The training personnel in the office of the director of human resources will periodically search for these opportunities and advertise them to the staff.

Training personnel will also search for and advertise conferences such as the SEI Software Product Line Conference (SPLC) to a wider audience.

---

18 SM Framework for Software Product Line Practice is a service mark of Carnegie Mellon University.

Arcade Game Maker Memo 04-06

To: CEO, Vice President for Product Development

From: Vice President for Product Planning

CC:

Date: November 4, 2004

Re: Customer Interface Management

This memo describes implications of the product line strategy for our relationships with customers.

The first increment of products will be a set of freeware products available on our website. After customers provide minimal contact information for our marketing purposes, they may download the products individually. We will use the timing of the downloads as a rudimentary customer satisfaction indicator: those who download one product and then come back after at least 24 hours to download additional products will be considered satisfied; those who download one product and do not return will be considered not satisfied.

The second increment of products is fairly traditional, and our sales staff will contact ongoing customers in the telecommunications domain. The difference in this increment will be the approach. Our salespeople generally ask customers what they want, conduct a very preliminary requirements elicitation, and then write a contract. The new strategy will require salespeople to explain what products are possible given the variations that will be available in the second increment. We will provide extensive training for the salespeople to be certain they present the limited number of variations positively. Salespeople will listen to what customers want and then provide valuable input for the final specification of the third increment.

The third increment of products will target a much broader customer base than the second increment. The number and types of variations will be expanded from increment two. We will contact a new customer base: convention planners and convention marketing companies. We will replace the increment one products on the Web with demos of the convention products. However, the demos will not be available for download, because our goal is on-the-spot delivery (for most customers). The customer will provide the salesperson with the graphic images and bits of text that will show up in the games. The salesperson will then invoke the generator and build the product.

The biggest issue that we face with customers is the transition from “the customer can have anything they want” to “the customer can chose from the set of available options.” For the third increment, we expect mostly new customers who think that we can make software do anything they want. We hope that our pricing and rapid delivery will manage this expectation sufficiently.

Arcade Game Maker Memo 05-01

To: For the record

From: Vice President for Product Development

CC: Vice President for Product Planning

Date: January 30, 2005

Re: Acquisition Strategy

This memo documents the acquisition strategy for the AGM product line organization. The strategy covers all software used in AGM products but created outside of AGM. Such software comprises a large part of AGM products and affects our product liability insurance.

Our product development process will be standards based: that is, the architecture, design, and implementations of products will use all applicable standards (e.g., data interchange). Doing so will provide AGM with the maximum flexibility in locating and integrating externally acquired software components.

The five modes in AGM's acquisition strategy–buy, commission, mine, build, or adopt–are defined below:

1. **Buy.** Software may be purchased from a vendor just like any other commodity.
2. **Commission.** Software may be special ordered from a vendor.
3. **Mine.** Software may be extracted from previous AGM products.
4. **Build.** Software may be created by AGM developers.
5. **Adopt.** Freeware or shareware may be accepted into a product build.

AGM will acquire (and not create) software whenever it is cost efficient. All developers–regardless of whether they create software core assets or software that is unique to a product–will analyze how the asset might be provided. They should analyze the cost for all five acquisition modes and hold all software–regardless of how it was acquired or developed–to the same development and testing standards (i.e., every asset must be tested and meet all quality requirements). Each AGM product development team will maintain an architecture view that identifies each module's source.

AGM is committed to the use of open source licensed software, and we expect that at least 50% of each product will be open source. All software (including open source) will be tested before it is in-

cluded in any AGM product release. All externally acquired software will be tested by AGM personnel to the same level of coverage as internally developed software unless the software is from a "trusted" source.[19]

Commissioned and purchased assets will be acquired following AGM's standard Request for Bid and Sole Source Purchase procedures. The fact that the software is paid for will not diminish the need to test it and eventually reach trusted status. The costs of vendor interaction must be included in the analysis.

Appendix C: Adoption Plan

The Arcade Game Maker (AGM) product line will produce a series of arcade games. Each game is a one-player game in which the player controls, to some degree, the moving objects. The objective is to score points by hitting stationary obstacles. The games range from low complexity to high and will be available on a variety of platforms.

This is the product line adoption plan. Its purpose is to define the steps that AGM will follow to adopt the product line software development strategy.

Readership

The adoption plan provides some level of information to all the stakeholders in the AGM software product line. Managers will find information that supports scheduling and work assignments. Executives will find a tactical plan which maps to the organization's strategic plan. Engineers will find the list of tasks to which they will eventually be assigned.

Primary Goal

The primary goal of the AGM Product Line Adoption Plan is to provide a plan, including schedule and work assignments, for conducting the activities required to launch the AGM Product Line.

Rationale

There are a number of actions that must be carried out to launch the product line. The relationships among these actions are complex and variable. This plan describes the specific set of actions that will achieve AGM's goals and operationalizes these actions in sequences that will be most effective.

---

[19] A trusted source is identified by the core asset team as providing consistently high-quality software. Software from trusted sources will still be tested but only to a minimal, API-level of coverage.

Guiding Principles

- AGM will follow the 29 practices defined in the SEI Framework for Software Product Line Practice [Clements04].
- AGM is taking an incremental approach to initiating product line practices. Each scheduled activity should be scheduled just in time to support follow on activities and should only be carried out to the extent necessary for successful operation. Each iteration of the product line activities will introduce further refinements and extensions to the core asset base and the processes that support the product line.
- From its time as a manufacturer of game boxes, AGM has an overarching manufacturing process that guides the development of any product. Any product line development processes must fit within this process.
- AGM is a CMM Level 2 organization and as such has a number of processes, including a process definition process, which must be followed. Any new processes must be coordinated via AGM's existing process roadmap.
- AGM's risk mitigation strategy relies on following the Launching and Institutionalizing Product Line practice area described in [Clements02].

Prerequisites

The VPPD has already conducted sufficient analysis to determine that the product line strategy holds promise for AGM. This is sufficient cause to move forward with launching the product line.

Owner

The AGM VPPD is the owner of this adoption plan.

Objectives of the Adoption Effort

- AGM will produce the first iteration of products, freeware implementations of three games, from the first release of a core asset base within six months of the initiation of this plan.
- AGM will produce successive three-product iterations at intervals of 3 months after the initial iteration.

Strategies

- AGM will follow an iterative, incremental approach to product line development.
- AGM will exploit existing process, technical, and organizational strengths in moving to the product line approach.
- AGM will follow the IDEAL model of technology change introduction.

Approach

Initiating

The VPPD will establish the initial product line governance structure and management controls.

Activity 1	Establish product line oversight	
	Owner	VPPD & VPPP
	Deliverables	Governance document; oversight committee assignments
	Major risks	The oversight committee members may not fully understand the goals for the product line. Needed actions may not be taken. The governance document may not be sufficiently comprehensive, requiring personnel to spend unnecessary time resolving issues.
	Communi-cations	The VPPD must fully and clearly communicate the goals of the product line.
	Reviews	The VPPD & VPPP will each review and must approve the document.
	Resources	Executives/managers as assigned

Activity 2	Create Product Line Organization	
	Owner	VPPD
	Deliverables	Organizational structure: assignments and chart Concept of Operations (CONOPS) Product Line manager designation
	Major risks	1. The roles needed in the organization may not align with existing skills requiring extensive training. 2. The new organizational structure and CONOPS may present a cultural change that will make implementation difficult.
	Communica-tions	The new organizational structure and new operational procedures must be clearly communi-cated to the staff.
	Reviews	AGM's CEO must review and approve the new structure and the CONOPS before each be-comes effective.
	Resources	VPPD and staff

Diagnosing

The VPPD decided he needed an outside opinion about his organization before committing to the product line strategy. He hired the SEI to conduct an SEI Product Line Technical Probe (PLTP). The results of the PLTP are not included with this example. This activity is intended to diagnose the strengths of AGM with respect to the practices needed for a successful product line and to identify ar-eas where it will be challenged.

Activity 3	Conduct Product Line Technical Probe (PLTP)	
	Owner	Product Line Manager
	Deliverables	PLTP Report
	Major risks	AGM members will not be candid with the PLTP team, which will skew the results. The PLTP results will fail to identify the significant challenges.
	Communica-tions	The results of the PLTP must be communicated to executives and staff.
	Reviews	The VPPD and VPPP will review before communicating to the staff.
	Resources	SEI product line experts and executive staff
Activity 4	Remediation Planning	
	Owner	Product Line Manager
	Deliverables	Action Plan that describes remediating deficiencies found in the PLTP.
	Major risks	Remedial efforts will not be effective, leaving AGM without some of the skills necessary to be successful.
	Communica-tions	The plan will be communicated to all managers who will be responsible for the improve-ments.
	Reviews	The VPPD must approve the plan prior to execution.
	Resources	The entire product line organization.

Establishing

The Product Line Manager will charter the projects necessary to carry out the initial product line activities. Each project will be staffed by personnel from the appropriate functional teams. He will charter:

- A core asset development project that will produce sufficient core assets to populate the initial three products. This project includes
  - an architecture team
  - a process definition team
  - a component development team
- An organizational management project that will include
  - a monitoring activity
  - an advisory activity
- Three project teams that will produce the three products that will constitute the pilot product line.

Activity 5	Establish specific teams for product line operation	
	Owner	Product Line Manager
	Deliverables	Detailed staff assignments to roles in the product line organization
	Major risks	The product line manager may not select the best-qualified candidate for each job and thereby jeopardize success.
	Communica-tions	The product line manager communicates needs to functional managers. Functional managers designate their team members for tasks within each project.
	Reviews	The executive team reviews assignments.
	Resources	Product line manager and functional managers
Activity 6	Establish the tool infrastructure	
	Owner	Team consisting of representatives from the functional teams.
	Deliverables	A comprehensive tool plan
	Major risks	There may not be adequate tools for every task in the product line, resulting in too much error-prone handwork. 2. Managers may not be willing to make the investment in tools, which requires hard dollars instead of soft money within the company.
	Communica-tions	Designated tool team communicates with vendors about requirements.
	Reviews	Functional teams review the tool plan for their areas of responsibility.
	Resources	Tool team members
Activity 7	Establish the process roadmap and constituent processes	
	Owner	Team consisting of representatives from the functional teams.
	Deliverables	A comprehensive process roadmap and an initial set of processes.
	Major risks	The definitions of the new processes may not adequately reflect the product line approach and may lead to the need for rework of many assets. The new product line processes may be sufficiently different from current processes that extensive training is required.
	Communica-tions	Designated process development team interacts with the functional teams.
	Reviews	Functional teams review the process map for their areas of responsibility.
	Resources	Functional team members

Acting

The early scoping activity indicated that a software product line repays investment sufficiently fast so that the set of products AGM intends to produce can be divided into three increments of three products

each. The VPPD has decided that the first “increment” in which the first three products are to be developed will be a pilot for the product line strategy. This activity will be under the guidance of the newly appointed product line manager.

Activity 8	Core asset development	
	Owner	Core asset team lead
	Deliverables	Requirements document with commonality/variability analysis Architecture definition Components and test cases Variable build script
	Major risks	The core asset development team may not be able to strike the proper balance between sufficient depth to explore problems and sufficient speed to complete the pilot in a timely manner.
	Communications	The lead must communicate with the product team leads and the product line manager. The functional team members assigned to core asset development must communicate with counterparts on the product development teams.
	Reviews	The product line manager will review all team actions during the pilot.
	Resources	Core asset development team
Activity 9	Product development	
	Owner	Product team lead(s)
	Deliverables	Unique components and test cases Product implementations
	Major risks	The product development team may not use the core assets according to the production plan, which will undermine the product line efficiencies.
	Communications	The lead must communicate with the core asset team leads and the product line manager. The functional team members assigned to product development must communicate with counterparts on the core asset development team.
	Reviews	The product line manager will review all team actions during the pilot.
	Resources	Product development team(s)
Activity 10	Conduct training	
	Owner	Director, HR
	Deliverables	Training courses covering the structures, processes, and techniques used in the product line organization.
	Major risks	The training may be delivered so early that staff will have forgotten by the time they have the opportunity to put the ideas into practice. The training may be delivered too late to effect necessary change in behavior and approach.

		The training may become out of date very rapidly as the organization adjusts to lessons learned in practice.
	Communications	Training developers must communicate with process definition teams and managers to ensure that training is scheduled correctly and that courses are updated on a timely manner.
	Reviews	Product line manager and team leads
	Resources	Training staff and managers

Learning

For the pilot product line this phase will have particular importance. The monitoring function of the organizational management team will

Activity 11	Data collection and evaluation	
	Owner	VPPD
	Deliverables	Goal-driven evaluation report developed by the oversight committee
	Major risks	Management may continue to collect the same measures as before the product line and miss key indications that the product line is not performing as expected. Management may collect new measures but continue to interpret them in the old context and miss key indications that the product line is not performing as expected.
	Communications	The results of the evaluation will be widely communicated from the CEO on down.
	Reviews	CEO will review
	Resources	Oversight committee members and assigned staff

Activity 12	Apply lessons to artifacts	
	Owner	VPPD
	Deliverables	Revised versions of all structures, processes, and techniques
	Major risks	Unwarranted changes may be made if lessons are not carefully analyzed. Staff may become confused if artifacts are incrementally (and continually) changed, reducing the efficiency of the product line organization.
	Communications	All changes must be clearly communicated through “town hall meetings” and training courses.
	Reviews	VPPD will review
	Resources	Product line organization

Activities vs. Objectives[20] Validation Matrix				
	Objectives			
Activities	**Improve Market position**	**Reduce time to market**	**Increase productivity**	**Enable mass customization**
1. Establish product line oversight		high	medium	
2. Create Product Line Organization	high			
3. Conduct Product Line Technical Probe (PLTP)		high		high
4. Remediation Planning	low	medium	medium	medium
5. Establish specific teams for product line operation		medium		medium
6. Establish the tool infrastructure		high		medium
7. Establish the process roadmap and constituent processes				
8. Core asset development		high		
9. Product development	high			
10. Conduct training			high	
11. Data collection & evaluation	medium		high	
12. Apply lessons to artifacts	high			

Key	
Contributes greatly	high
Contributes somewhat	medium
Slightly affects	low

Responsibility Matrix								
	Team members							
Activities	**VPPD**	**VPPP**	**PL Manager**	**Core Asset Team**	**Product Team**	**SEI**	**Director HR**	**Ad hoc Team**
1. Establish product line oversight	o	o						
2. Create Product Line Organization	o	p	p					
3. Conduct Product Line Technical Probe (PLTP)	p	p	o	p	p	p	p	

[20] These are the product line objectives, not the objectives of the adoption effort.

4. Remediation Planning	p	p	o				p	
5. Establish specific teams for product line operation	p		o				p	
6. Establish the tool infrastructure			p					o
7. Core asset development			p	o				
8. Product development			p		o			
9. Conduct training			p	p	p		o	
10. Data collection & evaluation	o	p	p				p	
11. Apply lessons to artifacts	o		p	p	p			

Key	
owner	o
participant	p

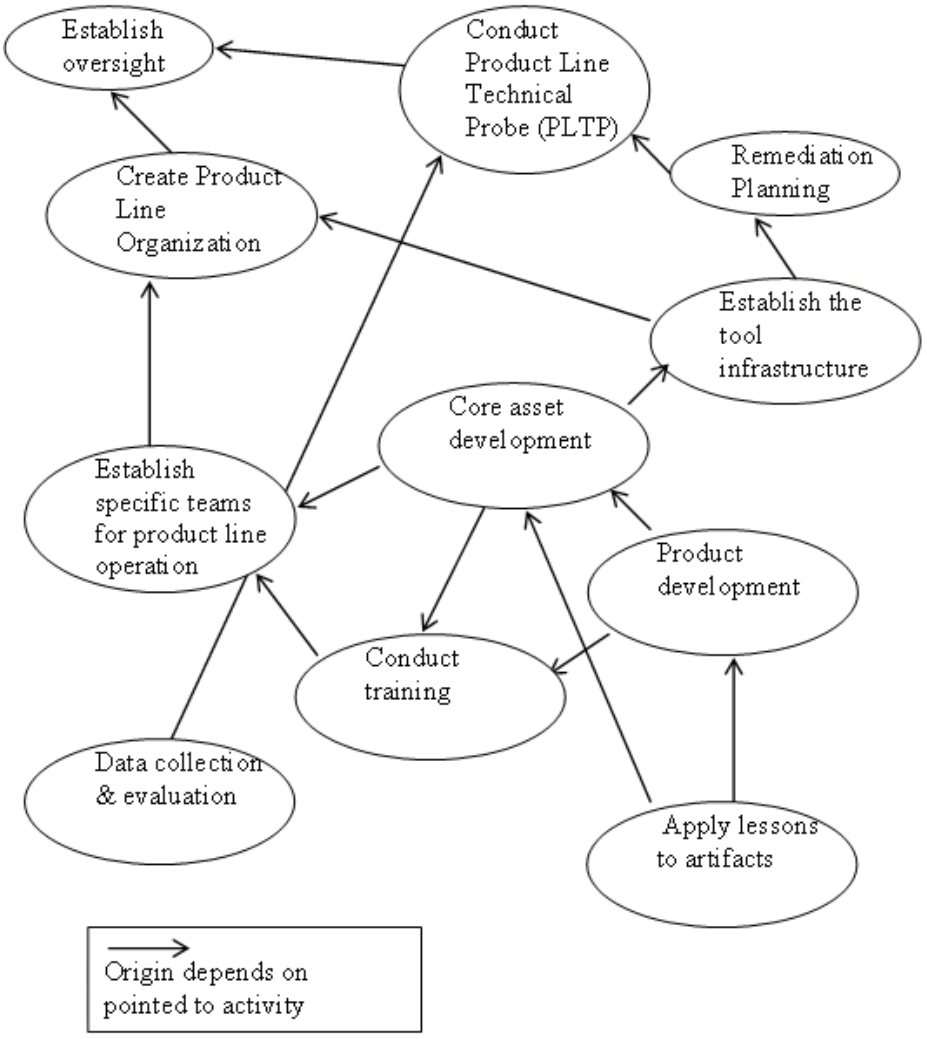

Figure 72: Activity Relationship Diagram

Major Risks and Mitigations

- There is a high probability some personnel will fail to perform their roles in the new operational structure resulting in missing artifacts and actions. This risk will be mitigated by developing a detailed Concept of Operations that describes the product line development process including each of the roles in the process. Training on the process will be required for each executive, manager, and engineer.
- There is a medium probability that, after the delivery of the first iteration of products, the product line strategy will be seen as taking too many resources for the benefits it provides. This risk will be mitigated by clearly modeling the costs and benefits in the business case.
- There is a medium probability the second iteration of products will require as much effort as the first iteration, reducing the benefits to AGM. This risk will be mitigated by defining and evaluating a software architecture and production plan for the products.

Appendix D: Marketing and Product Plan

The Arcade Game Maker (AGM) product line organization will produce a series of arcade games ranging from low to high obstacle count with a range of interaction effects. Each game is for one player who controls, to some degree, the moving objects. The objective is to score points by hitting stationary obstacles. The games range from low obstacle count to high and will be available on a variety of platforms.

The marketing and product plan communicates the marketing strategy that provides the rationale for which products are to be built and when.

Reusable Assets

The marketing and product plan establishes the high-level context for product line decisions. In a product line, assets are designed as reusable within the product line's context. That is, no attempt is made to make an asset "as general as possible," so each design decision is made with regard to the products in the product line. This context is formalized in the scope and refined in the architecture.

Readership

The marketing and product plan is intended primarily for stakeholders who are responsible for scoping and business case development. Managers can use the information that supports product planning, architects can use the information that supports commonality and variability analysis, and product developers will find the rationale for including each product in the product line.

Use in Product Planning and Production

The marketing and product plan is used in the earliest stages of product line planning. It is input into the business case for the software product line and provides information for the product line scope and schedule.

Product Context

AGM has been producing large-scale computer games for many years. Our latest analysis points to new markets that we have not yet served. Competitors are serving those markets already, but we believe that our experience and reputation will allow us to capture a sizable market share, approximately 30%, in a very short time.

Table 38 identifies three product types that we will analyze. The freeware games are simple implementations of basic games that are used to attract potential customers to our website. While the implementations can be simple, the quality must be high-these games represent AGM. The commercial PC games will run on a standard PC and be the "professional" versions of the freeware (i.e., they'll have more features than basic freeware games). Finally, the commercial wireless games will be sold to wireless carriers who will make the games available to their subscribers by download. The wireless games will be created by porting the professional PC games to a variety of wireless platforms.

Table 38: Example Attribute/Product Matrix

Feature	Freeware Arcade Games	Commercial PC Games	Commercial Wireless Games
Display	Standard bitmaps	Vector graphics	Wireless access protocol (WAP) bitmap
Interaction	Mouse/buttons	Pointing devices/buttons	Buttons
Game pieces	Simple icons	Dynamic icons	Dynamic icons
Scoring	High scores stored locally	Local storage/scores shared with subscription	Scores stored on network/subscription required

Feature Model

The feature model for the AGM product line can be found in the *Arcade Game Maker Pedagogical Product Line* Scope section.

Analysis and Projections

Hook

We have identified a possible hook around which we could build a marketing campaign: a “nostalgia” series in which we release Brickles, Pong, and maybe Bowling. The newest generation of players generally doesn’t know these games, but their parents do. We believe that releasing these games for new platforms and even old ones would be successful.

Niche Opportunity

We have identified an underserved opportunity that might be profitable if we can meet the necessary price point. Companies are always looking for new items to give away at conventions and conferences, and one exciting option is simple computer games that are customized to include company-specific icons and product presentations.

Projections

Table 39 shows our projected sales by product type over three release dates. Sales figures were developed based on the information presented earlier in the Marketing and Product Plan.

Table 39: Projected Sales Figures

	December 2004	May 2005	September 2005
Freeware PC games	300,000*	300,000*	300,000*
Wireless device games	400,000	750,000	200,000
Customized convention giveaways	500,000	500,000	4,500,000

* Number of downloads versus dollars.

The freeware games will generate moderate attention when they are deployed. They won't generate any revenue, but we estimate that sales for other releases would be 25% less if they are released before the freeware versions attract players to our website.

The wireless games will be built for the Mobile Information Device Profile (MIDP) 1.0 standard that will be widely available on wireless devices about April 2005. MIDP 1.0 will have a short life cycle: MIDP 2.0 will be released fall 2005.

The convention giveaways sales figures reflect increasing attendance as the economy recovers. We do not expect this business to be profitable until fall 2005.

Recommendations

We recommend that the product types described in Table 39 be put into production as soon as possible. Note that the figures in the body of the table provide a possible production sequence based on optimizing the sales figures. We recommend a nostalgic game series. Brickles, Pong, and Bowling tested positively with clients we interviewed. These games should be released in each of the three forms shown in Table 38, the Example Attribute/Product Matrix.

Maintaining the Marketing and Product Plan[21]

Periodically, the marketing group will scan the business environment and update the marketing and product plan. A scan will be initiated by the following events:

- a recent major change in economic forecasts
- a change in the company's strategic goals
- a year's passing since the last scan

As shown in Figure 73, a scan begins with data collection and is followed by data analysis and information dissemination to upstream and downstream consumers.

---

21 This section is the "attached process" described by Clements and Northrop [Clements 02]. For the product line marketing and product plan, the process focuses mainly on tracking industry and consumer changes.

Figure 73: Process Flow for Maintaining the Marketing and Product Plan

Bibliography

[Bachmann 00]	Bachmann, F.; Bass, L.; Chastek, G.; Donohoe, P.; & Peruzzi, F. *The Architecture Based Design Method* (CMU/SEI-2000-TR-001, ADA375851). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[Bass 99]	Bass, L. & Kazman, R. *Architecture-Based Development* (CMU/SEI-1999-TR-007, ADA366100). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Bass 03]	Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice*, Second Edition. Boston, MA: Addison-Wesley, 2003.
[Chastek 01]	Chastek, G.; Donohoe, P.; Kang, K. C.; & Theil, S. *Product Line Analysis: A Practical Introduction* (CMU/SEI-2001-TR-001, ADA396137). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[Chastek 02a]	Chastek, G. & McGregor, J. D. *Guidelines for Developing a Product Line Production Plan* (CMU/SEI-2002-TR-006, ADA407772). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
[Chastek 02b]	Chastek, G.; Donohoe, P.; & McGregor, J. D. *Product Line Production Planning for the Home Integration System Example* (CMU/SEI-2002-TN-029, ADA405846). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Clements 01]	Clements, P.; Cohen, S.; Donohoe, P.; & Northrop, L. *Control Channel Toolkit: A Software Product Line Case Study* (CMU/SEI-2001-TR-030, ADA396286). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[Clements 02]	Clements, Paul & Northrop, Linda. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.
[Clements 03]	Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2003.
[Clements 04]	Clements, P. & Northrop, L. *Framework for Software Product Line Practice Version 4.2*. (2004).
[Cohen 99]	Cohen, Sholom. *Guidelines for Developing a Product Line Concept of Operations* (CMU/SEI-1999-TR-008, ADA367714). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Cohen 01]	Cohen, Sholom. *Case Study: Building and Communicating a Business Case for a DoD Product Line* (CMU/SEI-2001-TN-020, ADA395155). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[Cohen 03]	Cohen, Sholom. *Predicting When Product Line Investment Pays* (CMU/SEI-2003-TN-017, ADA418466). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[DeBaud 99]	DeBaud, J. & Schmid, K. “A Systematic Approach to Derive the Scope of Software Product Lines,” 34-43. *Proceedings of the 21st ICSE*. Los Angeles, CA, May 16-22, 1999. Los Alamitos, CA: IEEE Computer Society, 1999.
[Heie 02]	Heie, Anders. Global Software Product Lines and Infinite Diversity (keynote from the Second Software Product Line Conference [SPLC2]). (2002).
[Kazman 00]	Kazman, R.; Klein, M.; & Clements, P. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004, ADA382629). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[McGregor 92]	McGregor, John D. & Sykes, David A. *Object-Oriented Software Development: Engineering Software for Reuse*. New York, NY: Van Nostrand Reinhold, 1992.
[McGregor 01a]	McGregor, John D. & Sykes, David A. *A Practical Guide to Testing Object-Oriented Software*. Boston, MA: Addison-Wesley, 2001.
[McGregor 01b]	McGregor, John D. *Testing a Software Product Line* (CMU/SEI-2001-TR-022, ADA401736). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[McGregor 03]	McGregor, John D. *The Evolution of Product Line Assets* (CMU/SEI-2003-TR-005, ADA408419). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[Toft 00]	Toft, Peter; Coleman, Derek; & Ohta, Joni. “A Cooperative Model for Cross-Divisional Product Development for a Software Product Line,” 111-132. *Software Product Lines: Experience and Research Directions: Proceedings of the First Software Product Lines Conference* (SPLC1). Denver, CO, August 28-31, 2000. Boston, MA: Kluwer Academic, 2000.

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu | www.cert.org
Email: info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0004419