

Product Development Lifecycle

Revised: May 18, 2012

By: Nelson Lin

Table of Contents

1. PRODUCT DEVELOPMENT LIFECYCLE	4
2. PRODUCT DEVELOPMENT PROCESS	6
2.1 REQUIREMENTS GATHERING	8
2.2 TEST CASES CONSTRUCTION	8
2.3 DATA MODELLING	9
2.4 CODING	10
2.5 TESTING	11
2.6 RELEASE	11
2. OPERATIONS AND MAINTENANCE	13
3. PROJECT MANAGEMENT	16
3.1 BACKGROUND THEORY	16
3.1.1 <i>Transparency</i>	16
3.1.2 <i>Inspection</i>	16
3.1.3 <i>Adaptation</i>	16
3.2 OUR SCRUM TEAM	17
3.2.1 <i>Product Owner</i>	17
3.2.2 <i>Development Team</i>	18
3.2.3 <i>Scrum Master</i>	18
3.3 SCRUM EVENTS	19
3.3.1 <i>The Sprint</i>	19
3.3.2 <i>Cancelling a Sprint</i>	20
3.3.3 <i>Sprint Planning Meeting</i>	20
3.3.4 <i>Sprint Goal</i>	21
3.3.5 <i>Weekly Scrum</i>	22
3.3.6 <i>Sprint Review</i>	22

3.3.7 <i>Sprint Retrospective</i>	23
3.4 SCRUM ARTEFACTS	23
3.4.1 <i>Product Backlog</i>	24
3.4.2 <i>Sprint Backlog</i>	25
3.4.3 <i>Increment</i>	26
3.5 DEFINITION OF “COMPLETED”	27

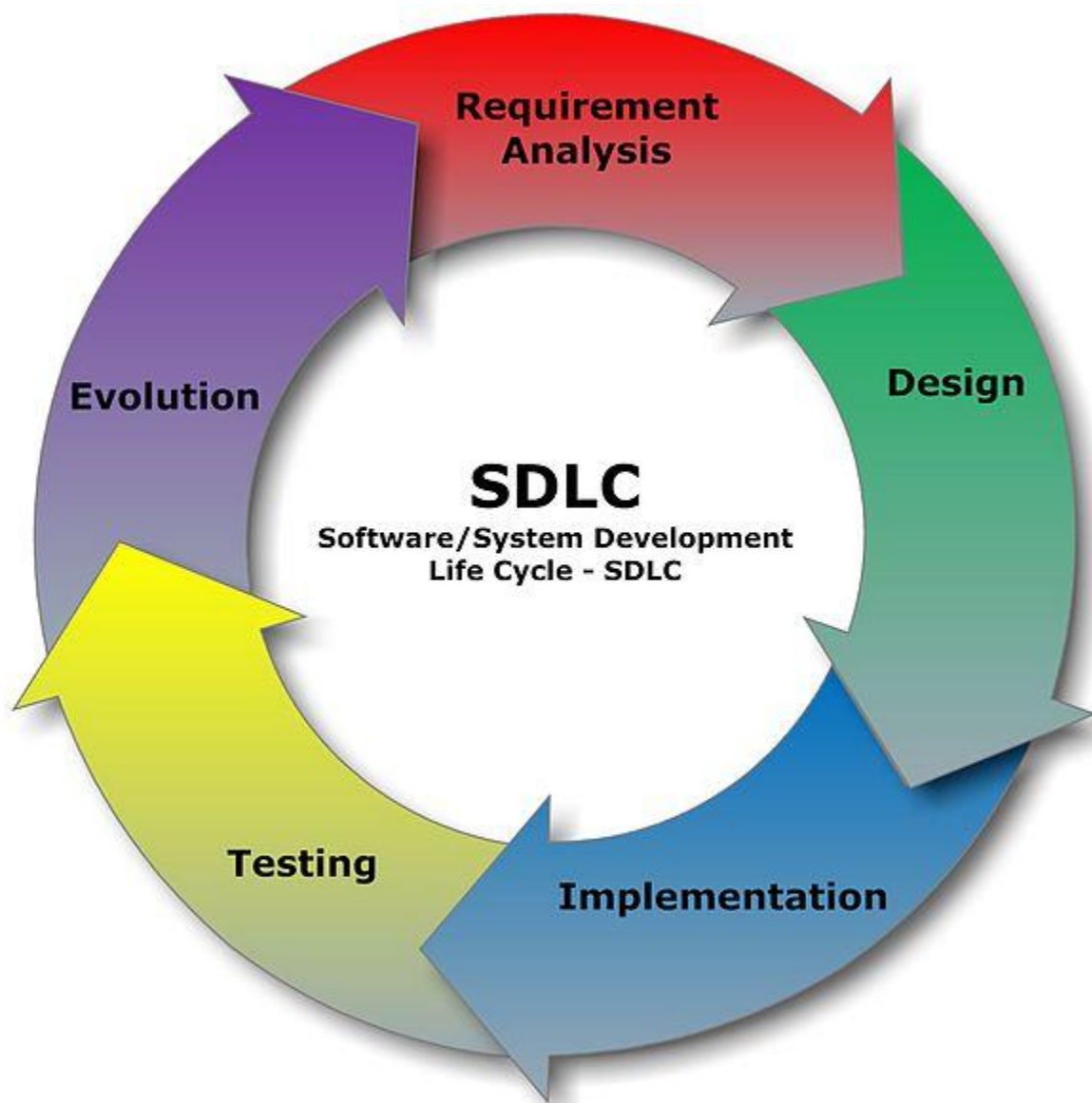
1. Product Development Lifecycle

Many development methodologies have been tried and tested in Robocoder Corporation. For the last ten years, we have found the combination of Zachman Framework, Scrum Methodology, and our own Rintagi automation to be most effective in creating, changing, and maintaining software information systems for internal use and for our customers.

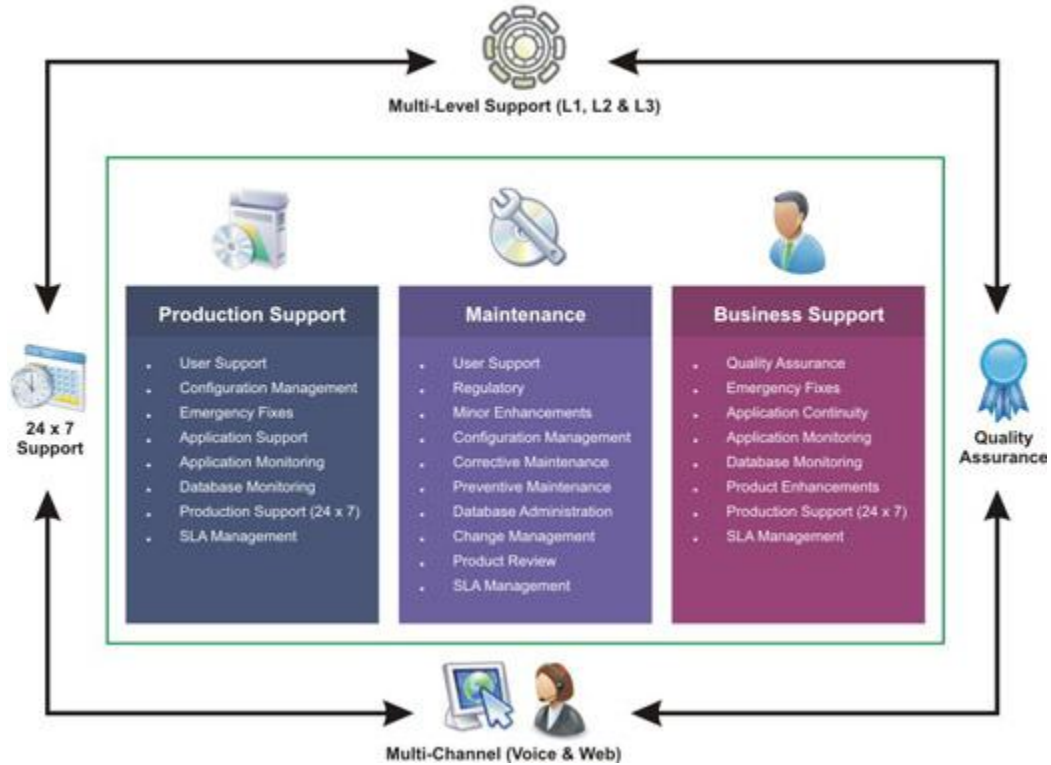
Our objective is to produce quality enterprise software demanded by large corporations in a fraction of the time and thus at a fraction of the cost. That is, custom software that has the characteristics of both integrity and agility.

There are three main aspects in the software development lifecycle:

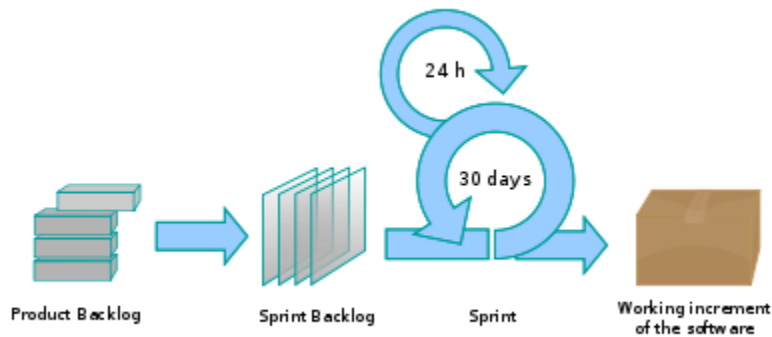
1. Product development process (Requirements, design, coding , testing, release);



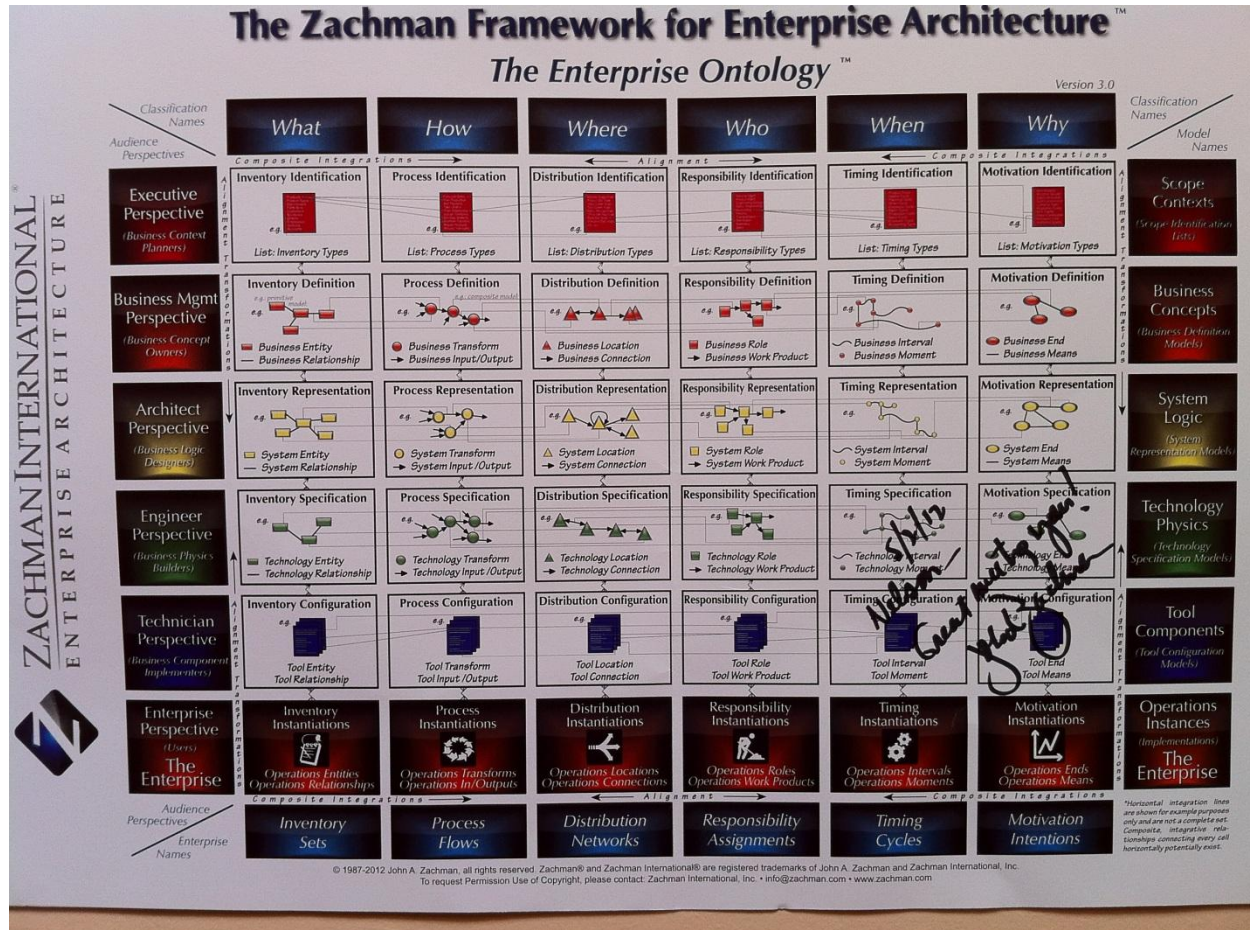
2. Operations and maintenance (deployment, change management, support, monitor);



3. Project Management (agile methodology);



Our product development process begins with architecture. We employ a variation of Zachman Framework as illustrated below:



This framework is a classification scheme for descriptive representations of an enterprise. It identifies a total, comprehensive set of models that are relevant for describing the enterprise.

It is the intersection between two historical classifications that have been in use for literally thousands of years. The first is the fundamentals of communication found in the primitive interrogatives: What, How, When, Who, Where, and Why. It is the integration of answers to these questions that enables the comprehensive, composite description of complex ideas. The second is derived from reification, the transformation of an abstract idea into an instantiation that was initially postulated by ancient Greek philosophers: Identification, Definition, Representation, Specification, Configuration and Instantiation.

Our emphasis is ontology. It is a theory of the existence of a structured set of essential components of an enterprise for which explicit expressions is necessary and perhaps even mandatory for creating, operating, and changing the enterprise. While many methodologies rely on complex modelling techniques that only few could understand, we focus on what represent the real business needs.

The key take away from this architecture is three layers:

1. Conceptual – the representation of the business requirements;
2. Logical – the representation of computing languages;
3. Physical – the representation on database and hardware;

Conceptual modelling has two purposes in the development of business processes and information systems. Firstly, it is a method for clarifying the major concepts used to conduct the business of the enterprise, thus facilitates communication among different parties. It helps to establish a common frame of reference within the organization. This is represented in the Zachman Framework in column 1 row 1.

Secondly, a Conceptual model is a high-level starting point for design and construction activities leading to implemented information systems that fulfil important business needs.

The logical data model can be a general descriptive model but is most often constructed as a normalized data model at a low or implementation level. This is done to ease the translation of the logical data model into a physical database model aimed at implementation. Logical models are represented in the Zachman Framework in column 1 row 2 and if fully normalized are in row 3.

Physical data models are representations of models that specify database or file structure implementations. Due to technology limitations and constraints, many unanticipated changes can be introduced in this layer of models. These models express physical characteristics such as physical nomenclature, storage, transport, and distribution of the actual data resources. They also express characteristics related to the access, navigation, and retrieval mechanisms for the stored data.

Physical models can sometimes be compromised to resemble, if not identical, to logical models in order to achieve database performance or ease of development. The artefacts they share are essentially the same although the terminology can be different. For example, instead of entities and attributes of the logical model, they are referred as tables and columns in physical model.

The physical database design is largely dictated by the database management system (DBMS) chosen for implementation. Relational implementations are most common in our development, although sometimes we would design dimensional model for retrieval of large amount of data such as data warehouse or data marts for decision support systems. Physical models are represented in the Zachman Framework in column 1 row 4 and 5.

This framework is a useful analytical tool to help us determine the following trade-offs, set ours' and customers' expectations, and devise strategies to mitigate the effects of our short and long term choices.

The trade-offs are:

1. Short term vs. long term options;
2. Implementation vs. integration;
3. Point in time solutions vs. infrastructure solutions;
4. Expense based approaches vs. asset based approaches;

5. Implementation optimization at the expense of the Enterprise vs. enterprise optimization at the expense of the implementation;
6. Etc.

2.1 Requirements Gathering

The objective here is to build a conceptual model to represent the business process. In this phase we found storyboarding to be most effective. Storyboarding was first used by Walt Disney Studio back in 1930s. The business process is sketched on paper and business rules written on the side.



There are four kinds of business rules:

1. Constraint rules that prevent inappropriate data being captured; eg. Error prompted
2. Inference rules specify what should happen if one or more conditions are met; eg. If A then B;
3. Computation rules specify the arithmetic operations on data; eg. $A + B = C$;
4. Action Enabler rules take specific actions with or without a condition; eg. Call an external system to retrieve a piece of data;

2.2 Test Cases Construction

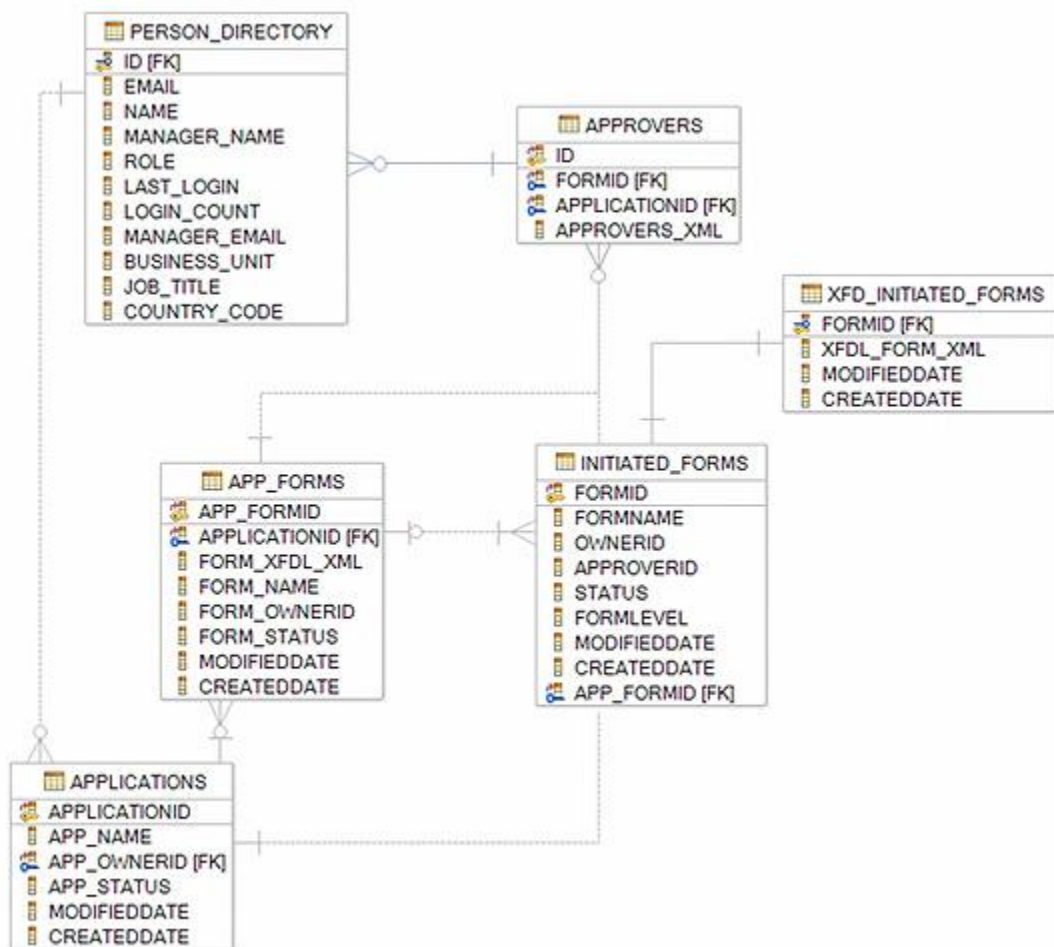
We found targeted programming more effective than un-targeted programming. The analogy is writing an exam. We often scores much higher if we know the questions ahead of time.

In this phase we create a checklist of test cases based on the information gathered above and validate them with relevant users briefly. This is a high level representation of what needs to be verified without detail information on how and the expected result. Therefore, this can significantly speed up the testing documentation. Occasional we specify expected value to be

one of the items to be checked if it is critical. A checklist system is being used here to keep track of each test taken with timestamp and person-stamped. These checklists are being updated all the time and are shared between developers and users. Both users and developers may update the template so each additional test is more comprehensive and accurate than the one before.

2.3 Data Modelling

With the test cases in mind, logical data model is being constructed. In 1976, Dr. Peter Chen at MIT developed the principles and practices around data modelling called Entity Relationship (ER) modelling technique.



This entity relationship model not only serves as database design, it also works well as a communication tool for discussing database implementations. The model consists of diagrams with quantitative and textual information attached to the objects represented. The objects that are diagrammed are entities, which are the things about which the business cares about, which are the various properties that the entities can have; and the relationships that exist between

entities. Additionally, metadata (data about the data) is captured that describes both the business meaning and use of the various attributes as well as their physical characteristics.

2.4 Coding

This is where we are making a difference. Instead of coding everything from scratch every time, we have developed software application called Rintagi to generate the targeted software application.

First a prototype is developed to validate the user interface, the behaviour and functionality for a typical screen and report. Once confirmed, all the possible combinations of changes to that screen or report are conceived and all the variations are developed into a prototype of its own. Now develop an application that can generate each variation when given the command in metadata (data on data). This is how Rintagi was built, organically from scratch.

Each time a change or addition is requested on the user interface provided by Rintagi, it would rebuild the entire module from scratch and discard the previous version. Because the entire automated code generation process has been tested, there is no need to test extensively for the requested change. Thus a lot of testings can be eliminated.

The analogy is like adding a sunroof to your house after the roof has been fully constructed. The ordinary process would be to cut a hole on the existing roof, build a support frame around it, have an engineer certify that it is structurally safe, add the sunroof, seal and test for leakage. In Rintagi, the “sunroof” is a dropdown selection with relative x and y coordinate to be specified on the “roof”. The “roof” is automatically tear down and thrown away replaced by brand new “roof” with a “sunroof” prebuilt on it. While it is not possible in the physical world, Rintagi codes live in a virtual world, making this an absolute reality. The entire process takes less than a second and because this has been tested and done many times, the “sunroof” is guaranteed safe, leak-proof and no need to be tested except to verify if it is in the best location; If not, change the x y coordinates and reconstruct the sunroof as many times as you like until your heart contents.

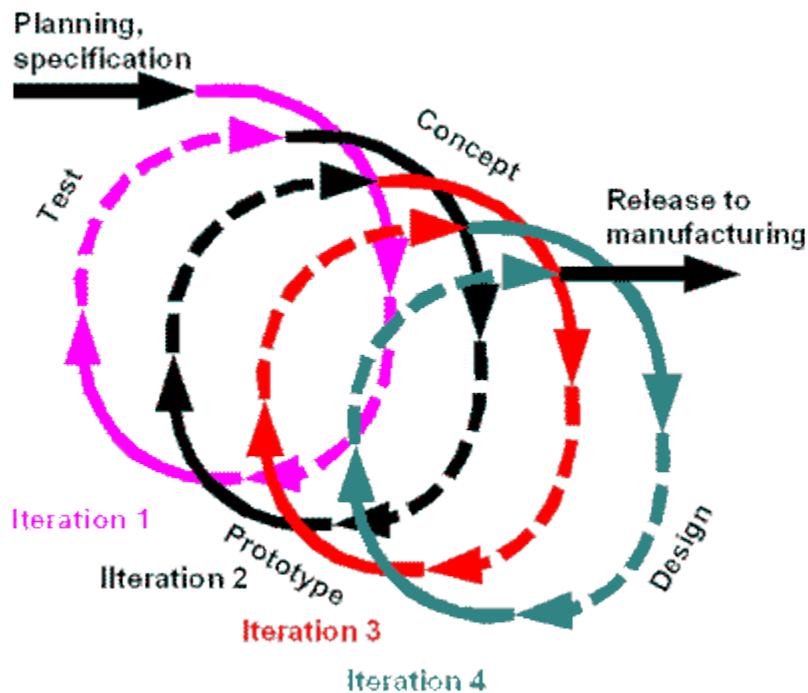
While this is absolutely the best way to develop an application, we have come across many occasions that the style, shape, size or functionality of the “sunroof” is different from our preconceived variations. Some variations can be handled by business rules. These business rules can be applied in three different tiers, namely, client tier on the browser, web tier on the web server, or server tier on the database server. Some variations have forced us to go back to add more combinations to the code generation process. This is where the Zachman Framework trade-off comes in handy. We have to determine if the requested “sunroof” is unique in its design that there will be no more requests like that in the future. If the answer is yes, we would choose to make it a custom program and never be regenerated again.

In custom programs we code with the test cases in mind. Then test against all the conceived test cases before deploying the tested codes from the development server to the test server.

2.5 Testing

The development server is designed for unit testing. The dedicated testing server is for user acceptance testing. The deployment process has been automated to minimize errors due to incomplete deployment. Every time a deployment is requested, the developer would compile an installation package and hand it over to the systems engineer to carry out the installation process. The systems engineer has his own check list to make sure all hardware and operating systems and accessories are in place to ensure a successful deployment.

Users would then use the checklist template created by the developer to further their testings. These checklist templates will be enhanced over time, errors report back to developers to be fixed and attempt to check again. This process is iterative and incremental, similar to the following:




2.6 Release

When users completed marking all the items on their checklists, it is time for release. A release date (most of the time is today's date) is chosen. On Rintagi, the developer would also increment the minor build number or major version number depending on the volume and significance of the enhancement to the current release. Then a deployment package similar to the one created for testing will be created and handed over to the system engineer to deploy to the production environment.

This deployment process has been automated. An interface is provided to the systems engineer. Upon successful credential verification and backup, the installation will preserve the data in the production system and only update the required programs.

RC Installer


Rintagi Installer

R6.43.20420
V2.4.11006
Administration
Common App

☐ New db Path: C:\SQLData
☒ Developer ☐ Production

Namespace from: RO to: Web Server: localhost

Client Tier
C:\inetpub\wwwroot\RO\Web Browse...
☐ DotNet 2.0 ☒ DotNet 3.5

Regular Web Service Tier
C:\inetpub\wwwroot\ROWS

XLS Import Web Service Tier
C:\inetpub\wwwroot\WsXls

Rule Tier
C:\Rintagi\RO Browse...
☐ DotNet 2.0 ☒ DotNet 3.5

Data Tier
Server Instance:
☐ MsSql 2005
Sys User: App User: ☒ MsSql 2008
Sys Pwd: App Pwd: ☐ Sybase 12.5

Client: \\ComputerName\C\$ Server: \\ComputerName\C\$ Backup Cancel Install

V3.0.10221 ©1999-2012 robocoder corporation. All rights reserved.

2. Operations and Maintenance

Since deployment has been automated, the focus here is on support. We have an internal policy that if there is a maintenance issue, we would drop all development if we have to, to resolve the maintenance issue first. It is our absolute priority to put our support before everything else.

These are our values and beliefs:

- To Deliver*** Delivery is #1 Key Success Factor in this industry. So follow through and be on time.
- To Serve*** We may turn down a prospect but we would not let our customer down. If and when there is a major quality issue, fix it before attempting other development.
- Be Committed*** You may refuse a given task at any time. However, once committed, follow through! Never let your customers or team members down.
- Earn Credibility*** Our value-added to our customer is intangible, they cannot see it and they cannot touch it; Keep your promises all the time, big or small, and earn the credibility.
- Be Respectful*** Always show respect while communicating with any internal or external people. Anything less is intolerable.
- Keep Confidentiality*** Doctors look after physical health, we look after financial health. Doctors do not share patient information among patients; we do not share information among clients.
- Be Discrete*** As a general rule, do not do any survey unless a written permission is obtained; It can be difficult to prove that the surveying company is not hired by our competition;
- Never Misrepresent*** Never misrepresent our products, services or value to anyone, internal or public, especially to our customers. There are ripple effects and the damage is usually one of the hardest to remedy.
- To Prioritize*** Prioritize according to customers' need. Never judge the importance of the task by your own standard, ask and you shall receive.
- To Learn*** Make everyday a learning experience. Take the opportunity to improve from past experience. Never make the same mistake twice.
- Take Total-Responsibility*** Take total responsibility all of the time. Never blame anyone or anything.
- Make Total-Solution*** Make sure solutions are well thought out in advance, all angles are covered, and genuine requirements are met.
- To Decide*** If you know it is your area of expertise, please do it; If not, pass it on to other team members who you know may do a better job. If you cannot decide, seek help from other team members.
- To Teamwork*** Think big! Think about others and help each other out. Use other people's experience!
- Earn Trust*** Never misinterpret information to team members. Earn their trust.
- Be Positive*** Do not justify what has been done in the past unless it helps to improve the future. We never have enough time to look back.
- To Focus*** Do more of it; do not wonder off somewhere else.
- Be Timely*** Quality of Life has a direct relationship with how effective time is spent. So use it wisely! Unless it is necessary, use 20% effort to achieve 80% result.

<i>To Backup</i>	The importance of backup cannot be over-emphasized! There is a saying in the IT industry that the three key success factors are Backup, Backup and Backup.
<i>Be Happy</i>	We can achieve a lot more when we are having fun! Happily achieving is way better than achieved happily!
<i>Keep Simple</i>	Whenever there is a choice, provide simple solutions; They are easier to create, change and maintain.
<i>Be Effective</i>	When communicating with others, get your points across in thirty seconds. Then take your time to describe in more details when asked.
<i>Be Resourceful</i>	Do not go by instinct, when you find a solution to a problem, find all the mutually exclusive alternatives, then pick the best from it.
<i>To Empathize</i>	Interpret customer or team member requirements correctly and go one step beyond what they currently need.
<i>To Aim</i>	Aim at the "ends" rather than the "means", unless you know for sure the "means" will take you to the "ends".
<i>To Anticipate</i>	Practice defensive Driving... Never drive in the blind-spot of others. Never let users drive you into failure. Anticipate their blind-spot, help them succeed!
<i>Be Proactive</i>	Think three steps beyond before implementing the first so that effort is not wasted, and more importantly save time.
<i>To Respect</i>	Make positive remark and criticize constructively when advising others. Be extra pleasant when advising clients. Do not let them lose face.
<i>Be Helpful</i>	Do not just deliver problems and never give excuses. Always present alternative solutions with any problem. When in doubt, ask for advice.
<i>To Listen</i>	God gave us two ears and a mouth, we should listen more than talk.
<i>To Respond</i>	Never ignore customer and team member questions and concerns.
<i>To Inform</i>	Keep your customers and team members well informed of your progress, especially when the job is completed. Having someone polling for answer is both inefficient and frustrating.
<i>Be Efficient</i>	Consult team members on any problem or issue that cannot be resolved within a reasonable timeframe. Do not reinvent the wheel, using existing and proven objects.
<i>Be Careful</i>	Do not take on unnecessary risks. Big mistake can cost millions of dollars and lots of humiliation. Words travel quickly, good or bad.
<i>To Share</i>	When dealing with something risky such as production data, do it together with another team member. Share both the glory and the blame.
<i>To Validate</i>	Before doing anything, ask how you would know if you have done it right. Prepare all the test cases so that validation can be performed easily when done.
<i>To Confirm</i>	Never change users' specifications without confirming with the corresponding users, unless you are given the Power-of-Attorney.
<i>Be Assertive</i>	Although it is a must to ask open questions when interviewing users for a preliminary overview, you should always make it easy for the users to just answer yes or no by stating your position or recommendation when interviewing for detail specifications.
<i>Never Assume</i>	Do not make assumption, always VERIFY! Assume is spelled ASS_U & _ME. Always present assertive statements and ask for a simple Yes or No answer.

***Be
Consistent***

Say “I don't know but I'll find out for you”, unless you are absolutely sure about the statement you are making. Turning the unknown into a question is the best strategy.

***Be
Organized
To***

Never attempt to memorize "outstanding" things to do. Write them down so you can focus on fulfilling the list instead.

Improvise

Use today's tools and technology to solve today's problems. Tomorrow's tools cannot help you today.

***Add
Value***

Always give the most for the least! Go an extra mile! That's value to the customer!

3. Project Management

Scrum is an agile framework for developing and sustaining complex software products. In Robocoder Corporation, we use a variation of Scrum to make clear the relative efficacy of our product management and development practices so that we can improve. It is not a process or a technique for building products; rather, it is a framework within which we can employ various processes and techniques.

The Scrum framework consists of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to the success and usage.

The rules of Scrum bind together the events, roles, and artifacts, governing the relationships and interaction between them.

3.1 Background Theory

Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience *and* making decisions based on what is known. Scrum employs an iterative, incremental approach to optimize predictability and control risk.

Three pillars uphold every implementation of empirical process control: transparency, inspection, and adaptation.

3.1.1 Transparency

Significant aspects of the process are visible to those responsible for the outcome.

Transparency requires those aspects be defined by a common standard so observers share a common understanding of what is being seen. We use shared checklists to communicate what is meant by “completed”.

3.1.2 Inspection

We frequently inspect Scrum artifacts and progress toward a goal to detect undesirable variances. These inspections are infrequent so that inspection does not get in the way of the work. We found inspections are most beneficial when diligently performed by buddy developer at the point of work. We do code-walking and we encourage buddy-system. Aside from catching each other's mistakes, the other benefit of buddy-system is having a back-up on everything we do.

3.1.3 Adaptation

If our buddy determines that one or more aspects of a process deviate outside acceptable limits, and that the resulting product will be unacceptable, the process or the material being processed is being adjusted as soon as possible to minimize further deviation.

Scrum prescribes four formal opportunities for inspection and adaptation:

- Sprint Planning Meeting
- Weekly Scrum
- Sprint Review
- Sprint Retrospective

3.2 Our Scrum Team

Our Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.

Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of “completed” product ensure a potentially useful version of working product is always available.

3.2.1 Product Owner

The Product Owner is responsible for maximizing the value of the product and the work of the Development Team. How this is done may vary widely depending on the nature of the project.

The Product Owner is the sole person responsible for managing the Product Backlog. Product Backlog management includes:

- Clearly expressing Product Backlog items;
- Ordering the items in the Product Backlog to best achieve goals and missions;
- Ensuring the value of the work the Development Team performs;
- Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next; and,
- Ensuring the Development Team understands items in the Product Backlog to the level needed.

The Product Owner may do the above work, or have the Development Team do it. However, the Product Owner remains accountable.

The Product Owner is one person. Those wanting to change a backlog item’s priority must convince the Product Owner.

For the Product Owner to succeed, our entire organization respects his or her decisions. The Product Owner’s decisions are visible in the content and ordering of the Product Backlog. No one is allowed to tell the Development Team to work from a different set of requirements, and the Development Team isn’t allowed to act on what anyone else says.

3.2.2 Development Team

The Development Team consists of professionals who do the work of delivering a potentially releasable Increment of “completed” product at the end of each Sprint. Only members of the Development Team create the Increment.

Development Teams are structured and empowered to organize and manage their own work. The resulting synergy optimizes the Development Team’s overall efficiency and effectiveness. Development Teams have the following characteristics:

- They are self-organizing. No one (not even the Scrum Master) tells the Development Team how to turn Product Backlog into Increments of potentially releasable functionality;
- Development Teams are cross-functional, with all of the skills as a team necessary to create a product Increment;
- Scrum recognizes no titles for Development Team members other than Developer, regardless of the work being performed by the person; there are no exceptions to this rule;
- Individual Development Team members may have specialized skills and areas of focus, but accountability belongs to the Development Team as a whole;
- Development Teams do not contain sub-teams dedicated to particular domains like testing or business analysis.

Optimal Development Team size is small enough to remain nimble and large enough to complete significant work. Fewer than two or three Development Team members decreases interaction and results in smaller productivity gains. Smaller Development Teams may encounter skill constraints during the Sprint, causing the Development Team to be unable to deliver a potentially releasable Increment. Having more than nine members requires too much coordination. Large Development Teams generate too much complexity for an empirical process to manage. The Product Owner and Scrum Master roles are not included in this count unless they are also executing the work of the Sprint Backlog.

3.2.3 Scrum Master

The Scrum Master is responsible for ensuring Scrum is understood and enacted. Scrum Masters do this by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules. The Scrum Master is a servant-leader for the Scrum Team. Robocoder Corporation has certified Scrum Master on staff.

The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which aren’t. The Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team.

The Scrum Master serves the Product Owner in several ways, including:

- Finding techniques for effective Product Backlog management;
- Clearly communicating vision, goals, and Product Backlog items to the Development Team;
- Teaching the Scrum Team to create clear and concise Product Backlog items;
- Understanding long-term product planning in an empirical environment;

- Understanding and practicing agility; and,
- Facilitating Scrum events as requested or needed.

The Scrum Master serves the Development Team in several ways, including:

- Coaching the Development Team in self-organization and cross-functionality;
- Teaching and leading the Development Team to create high-value products;
- Removing impediments to the Development Team's progress;
- Facilitating Scrum events as requested or needed; and,
- Coaching the Development Team in organizational environments in which Scrum is not yet fully adopted and understood.

The Scrum Master serves the organization in several ways, including:

- Leading and coaching the organization in its Scrum adoption;
- Planning Scrum implementations within the organization;
- Helping employees and stakeholders understand and enact Scrum and empirical product development;
- Causing change that increases the productivity of the Scrum Team; and,
- Working with other Scrum Masters to increase the effectiveness of the application of Scrum in the organization.

3.3 Scrum Events

Prescribed events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum. Scrum uses time-boxed events, such that every event has a maximum duration. This ensures an appropriate amount of time is spent planning without allowing waste in the planning process.

Other than the Sprint itself, which is a container for all other events, each event in Scrum is a formal opportunity to inspect and adapt something. These events are specifically designed to enable critical transparency and inspection. Failure to include any of these events results in reduced transparency and is a lost opportunity to inspect and adapt.

3.3.1 The Sprint

The heart of Scrum is a Sprint, a time-box of one month or less during which a “completed”, useable, and potentially releasable product Increment is created. Sprints have consistent durations throughout a development effort. A new Sprint starts immediately after the conclusion of the previous Sprint.

Sprints contain and consist of the Sprint Planning Meeting, Weekly Scrums, the development work, the Sprint Review, and the Sprint Retrospective.

During the Sprint:

- No changes are made that would affect the Sprint Goal;
- Development Team composition remains constant;
- Quality goals do not decrease; and,

- Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.

Each Sprint may be considered a project with no more than a one-month horizon. Like projects, Sprints are used to accomplish something. Each Sprint has a definition of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product.

Sprints are limited to one calendar month. When a Sprint's horizon is too long the definition of what is being built may change, complexity may rise, and risk may increase. Sprints enable predictability by ensuring inspection and adaptation of progress toward a goal at least every calendar month. Sprints also limit risk to one calendar month of cost.

3.3.2 Cancelling a Sprint

A Sprint can be cancelled before the Sprint time-box is over. Only the Product Owner has the authority to cancel the Sprint, although he or she may do so under influence from the stakeholders, the Development Team, or the Scrum Master.

A Sprint would be cancelled if the Sprint Goal becomes obsolete. This might occur if the company changes direction or if market or technology conditions change. In general, a Sprint should be cancelled if it no longer makes sense given the circumstances. But, due to the short duration of Sprints, cancellation rarely makes sense.

When a Sprint is cancelled, any completed and “completed” Product Backlog Items are reviewed. If part of the work is potentially releasable, the Product Owner typically accepts it. All incomplete Product Backlog Items are re-estimated and put back on the Product Backlog. The work done on them depreciates quickly and must be frequently re-estimated.

Sprint cancellations consume resources, since everyone has to regroup in another Sprint Planning Meeting to start another Sprint. Sprint cancellations are often traumatic to the Scrum Team, and are very uncommon.

3.3.3 Sprint Planning Meeting

The work to be performed in the Sprint is planned at the Sprint Planning Meeting. This plan is created by the collaborative work of the entire Scrum Team.

The Sprint Planning Meeting is time-boxed to eight hours for a one-month Sprint. For shorter Sprints, the event is proportionately shorter. For example, two-week Sprints have four-hour Sprint Planning Meetings.

The Sprint Planning Meeting consists of two parts, each one being a time-box of one half of the Sprint Planning Meeting duration. The two parts of the Sprint Planning Meeting answer the following questions, respectively:

- What will be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

1. What will be done this Sprint?

In this part, the Development Team works to forecast the functionality that will be developed during the Sprint. The Product Owner presents ordered Product Backlog items to the Development Team and the entire Scrum Team collaborates on understanding the work of the Sprint.

The input to this meeting is the Product Backlog, the latest product Increment, projected capacity of the Development Team during the Sprint, and past performance of the Development Team. The number of items selected from the Product Backlog for the Sprint is solely up to the Development Team. Only the Development Team can assess what it can accomplish over the upcoming Sprint.

After the Development Team forecasts the Product Backlog items it will deliver in the Sprint, the Scrum Team crafts a Sprint Goal. The Sprint Goal is an objective that will be met within the Sprint through the implementation of the Product Backlog, and it provides guidance to the Development Team on why it is building the Increment.

II. How will the chosen work get done?

Having selected the work of the Sprint, the Development Team decides how it will build this functionality into a “completed” product Increment during the Sprint. The Product Backlog items selected for this Sprint plus the plan for delivering them is called the Sprint Backlog.

The Development Team usually starts by designing the system and the work needed to convert the Product Backlog into a working product Increment. Work may be of varying size, or estimated effort. However, enough work is planned during the Sprint Planning Meeting for the Development Team to forecast what it believes it can do in the upcoming Sprint. Work planned for the Sprint by the Development Team is decomposed to units of one week or less by the end of this meeting. The Development Team self-organizes to undertake the work in the Sprint Backlog, both during the Sprint Planning Meeting and as needed throughout the Sprint.

The Product Owner may be present during the second part of the Sprint Planning Meeting to clarify the selected Product Backlog items and to help make trade-offs. If the Development Team determines it has too much or too little work, it may renegotiate the Sprint Backlog items with the Product Owner. The Development Team may also invite other people to attend in order to provide technical or domain advice.

By the end of the Sprint Planning Meeting, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment.

3.3.4 Sprint Goal

The Sprint Goal gives the Development Team some flexibility regarding the functionality implemented within the Sprint.

As the Development Team works, it keeps this goal in mind. In order to satisfy the Sprint Goal, it implements the functionality and technology. If the work turns out to be different than the Development Team expected, then they collaborate with the Product Owner to negotiate the scope of Sprint Backlog within the Sprint.

The Sprint Goal may be a milestone in the larger purpose of the product roadmap.

3.3.5 Weekly Scrum

The Weekly Scrum is a 15-minute time-boxed event for each Development Team to synchronize activities and create a plan for the next week. This is done by inspecting the work since the last Weekly Scrum and forecasting the work that could be done before the next one.

The Weekly Scrum is held around the same time at the head office each week to reduce complexity. During the meeting, each Development Team member explains:

- What has been accomplished since the last meeting?
- What will be done before the next meeting?
- What obstacles are in the way?

The Development Team uses the Weekly Scrum to assess progress toward the Sprint Goal and to assess how progress is trending toward completing the work in the Sprint Backlog. The Weekly Scrum optimizes the probability that the Development Team will meet the Sprint Goal. The Development Team often meets immediately after the Weekly Scrum to re-plan the rest of the Sprint's work. Every week, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work together as a self-organizing team to accomplish the goal and create the anticipated Increment in the remainder of the Sprint.

The Scrum Master ensures that the Development Team has the meeting, but the Development Team is responsible for conducting the Weekly Scrum. The Scrum Master teaches the Development Team to keep the Weekly Scrum within the 15-minute time-box.

The Scrum Master enforces the rule that only Development Team members participate in the Weekly Scrum. The Weekly Scrum is not intended to be a status meeting, and is for the people transforming the Product Backlog items into an Increment.

Weekly Scrums improve communications, eliminate other meetings, identify and remove impediments to development, highlight and promote quick decision-making, and improve the Development Team's level of project knowledge. This is a key inspect and adapt meeting.

3.3.6 Sprint Review

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done. This is an informal meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

This is a four-hour time-boxed meeting for one-month Sprints. Proportionately less time is allocated for shorter Sprints. For example, two week Sprints have two-hour Sprint Reviews.

The Sprint Review includes the following elements:

- The Product Owner identifies what has been “completed” and what has not been “completed”;
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved;

- The Development Team demonstrates the work that it has “completed” and answers questions about the Increment;
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely completion dates based on progress to date; and,
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning Meetings.

The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.

3.3.7 Sprint Retrospective

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning Meeting. This is a three-hour time-boxed meeting for one-month Sprints. Proportionately less time is allocated for shorter Sprints.

The purpose of the Sprint Retrospective is to:

- Inspect how the last Sprint went with regards to people, relationships, process, and tools;
- Identify and order the major items that went well and potential improvements; and,
- Create a plan for implementing improvements to the way the Scrum Team does its work.

The Scrum Master encourages the Scrum Team to improve, within the Scrum process framework, its development process and practices to make it more effective and enjoyable for the next Sprint. During each Sprint Retrospective, the Scrum Team plans ways to increase product quality by adapting the Definition of “completed” as appropriate.

By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a formal opportunity to focus on inspection and adaptation.

3.4 Scrum Artefacts

Scrum’s artefacts represent work or value in various ways that are useful in providing transparency and opportunities for inspection and adaptation. Artefacts defined by Scrum are specifically designed to maximize transparency of key information needed to ensure Scrum Teams are successful in delivering a “completed” Increment.

3.4.1 Product Backlog

The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

A Product Backlog is never complete. The earliest development of it only lays out the initially known and best-understood requirements. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic; it constantly changes to identify what the product needs to be appropriate, competitive, and useful. As long as a product exists, its Product Backlog also exists.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, and estimate.

The Product Backlog is often ordered by value, risk, priority, and necessity. Top-ordered Product Backlog items drive immediate development activities. The higher the order, the more a Product Backlog item has been considered, and the more consensus exists regarding it and its value.

Higher ordered Product Backlog items are clearer and more detailed than lower ordered ones. More precise estimates are made based on the greater clarity and increased detail; the lower the order, the less detail. Product Backlog items that will occupy the Development Team for the upcoming Sprint are fine-grained, having been decomposed so that any one item can be “completed” within the Sprint time-box. Product Backlog items that can be “completed” by the Development Team within one Sprint are deemed “ready” or “actionable” for selection in a Sprint Planning Meeting.

As a product is used and gains value, and the marketplace provides feedback, the Product Backlog becomes a larger and more exhaustive list. Requirements never stop changing, so a Product Backlog is a living artifact. Changes in business requirements, market conditions, or technology may cause changes in the Product Backlog.

Multiple Scrum Teams often work together on the same product. One Product Backlog is used to describe the upcoming work on the product. A Product Backlog attribute that groups items is then employed.

Product Backlog grooming is the act of adding detail, estimates, and order to items in the Product Backlog. This is an ongoing process in which the Product Owner and the Development Team collaborate on the details of Product Backlog items. During Product Backlog grooming, items are reviewed and revised. However, they can be updated at any time by the Product Owner or at the Product Owner’s discretion.

Grooming is a part-time activity during a Sprint between the Product Owner and the Development Team. Often the Development Team has the domain knowledge to perform grooming itself. How and when grooming is done is decided by the Scrum Team. Grooming usually consumes no more than 10% of the capacity of the Development Team.

The Development Team is responsible for all estimates. The Product Owner may influence the Development Team by helping understand and select trade-offs, but the people who will perform the work make the final estimate.

At any point in time, the total work remaining to reach a goal can be summed. The Product Owner tracks this total work remaining at least for every Sprint Review. The Product Owner compares this amount with work remaining at previous Sprint Reviews to assess progress toward completing projected work by the desired time for the goal. This information is made transparent to all stakeholders.

Various trend burndown, burnup and other projective practices have been used to forecast progress. These have proven useful. However, these do not replace the importance of empiricism. In complex environments, what will happen is unknown. Only what has happened may be used for forward-looking decision-making.

3.4.2 Sprint Backlog

The Sprint Backlog is the set of Product Backlog items selected for the Sprint plus a plan for delivering the product Increment and realizing the Sprint Goal. The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality.

The Sprint Backlog defines the work the Development Team will perform to turn Product Backlog items into a “completed” Increment. The Sprint Backlog makes visible all of the work that the Development Team identifies as necessary to meet the Sprint Goal.

The Sprint Backlog is a plan with enough detail that changes in progress can be understood in the Weekly Scrum. The Development Team modifies Sprint Backlog throughout the Sprint, and the Sprint Backlog emerges during the Sprint. This emergence occurs as the Development Team works through the plan and learns more about the work needed to achieve the Sprint Goal.

As new work is required, the Development Team adds it to the Sprint Backlog. As work is performed or completed, the estimated remaining work is updated. When elements of the plan are deemed unnecessary, they are removed. Only the Development Team can change its Sprint Backlog during a Sprint. The Sprint Backlog is a highly visible, real-time picture of the work that the Development Team plans to accomplish during the Sprint, and it belongs solely to the Development Team.

At any point in time in a Sprint, the total work remaining in the Sprint Backlog items can be summed. The Development Team tracks this total work remaining at least for every Weekly Scrum. The Development Team tracks these sums weekly and projects the likelihood of achieving the Sprint Goal. By tracking the remaining work throughout the Sprint, the Development Team can manage its progress.

Scrum does not consider the time spent working on Sprint Backlog Items. The work remaining and date are the only variables of interest.

3.4.3 Increment

The Increment is the sum of all the Product Backlog items completed during a Sprint and all previous Sprints. At the end of a Sprint, the new Increment must be “completed” which means it must be in useable condition and meet the Scrum Team’s Definition of “completed”. It must be in useable condition regardless of whether the Product Owner decides to actually release it.

3.5 Definition of “Completed”

At Robocoder Corporation, everyone understands what “completed” means. We achieve this via shared checklists. When checked, the work is completed on the product Increment.

The same definition guides the Development Team in knowing how many Product Backlog items it can select during a Sprint Planning Meeting. The purpose of each Sprint is to deliver Increments of potentially releasable functionality that adhere to the Scrum Team’s current Definition of “completed”.

Development Teams deliver an Increment of product functionality every Sprint. This Increment is useable, so a Product Owner may choose to immediately release it. Each Increment is additive to all prior Increments and thoroughly tested, ensuring that all Increments work together.

