TigerCheck  
Architectural Spike Report

By

Ethan Darby

Zachary White

Nick Bean

Ross Spears

Submitted in partial fulfillment of the requirements for  
COMP 4710 Senior Design  
to the Department of Computer Science and Software Engineering,  
Samuel Ginn College of Engineering, Auburn University

Auburn, Alabama

September 30, 2011

# Table of Contents

Table of Contents ii

2 Executive Summary (System Metaphor) 1

3 Project Introduction 2

3.1 Previous Development 2

3.2 Intent This Cycle 2

3.3 Future Work 2

4 Requirements / User Stories 3

4.1 User Stories 3

4.2 User Stories 3

4.2.1 Off-Screen Ghost Markers 3

5 Design Documentation 5

6 Management Plan 6

6.1 Task Assignments 6

6.2 Development Schedule 6

6.3 Planned Code / Feature Freeze 6

7 Risk Mitigation 7

8 Test Plan and Test Procedures 8

8.1 Test Plan 8

8.2 Test Procedures 9

8.2.1 Procedure 1: Sample Test Procedure 9

9 Lessons Learned 10

References 11

Appendix A 12

A.1 Status Reports 12

A.2 Meeting Minutes 12

A.3 Size Estimation Documentation 12

A.4 Problem Reports / Change Requests 12

A.5 Correspondence 12

A.6 Source Code 12

[ DELETE THIS SECTION FROM YOUR FINAL REPORT ]

General Instructions for your Cycle Report

Basically, any written material you use or generate should be included in your cycle binder. This may include:

* Copies of all email correspondence related to the project.
* Design and process documents such as user stories, use cases, UML diagrams, database diagrams, user interface storyboards, requirements documents, etc.
* Copies of all status reports for the cycle.
* Your system metaphor, management plan, project schedule, etc.
* Your cycle Powerpoint presentation.
* All source code, database or XML schema, scripts, etc.

## Electronic Submission

All content in your cycle binder must be submitted electronically as a single PDF. Paper documents may be scanned using the department’s copy machine in 3101 Shelby Center. No account password is needed to scan.

Several free tools are available, such as CutePDF (through OIT software install page) and others. If you have it, Adobe Acrobat (not Acrobat Reader) is the best product available. You should optimize your final PDF to reduce file size prior to submittal. Un-optimized files (especially scanned documents) can be hundreds of megabytes in size.

Your PDF must be uploaded to Blackboard. Do NOT email them to us – these will exceed our mailbox size limits.

### Printouts

Bring TWO (2) printed copies of your cycle binder to your presentation. We will write grading comments on these and return them to you for your review. You may omit lengthy non-process sections of the cycle binder (such as source code) from the printed copy.

Bring TWO (2) printed copies of your PDF slides for the instructors.

# Executive Summary (System Metaphor)

Describe your project here in layman’s terms. This is the “elevator speech” designed to get someone quickly acquainted with your project. Assume the reader has no prior knowledge or technical skill. This section should be at least three but no more than five paragraphs in length.

# Project Introduction

This is the introduction to your project. It should be written in plain language and you should assume your reader knows absolutely nothing about this project. Start at the beginning, and give a full summary of the project, the customer, the reason this project is needed, and the goals your project this semester.

## Previous Development

Here, you can summarize any previous development efforts, both in the architectural spike and in previous cycles. You should be detailed and discuss goals for each previous cycle, milestones achieved, and work that was not completed.

## Intent This Cycle

Here you will discuss what your goals for this cycle are. Be specific and detailed. The cycle intent is a description of the envisioned product at the end of the cycle. It should capture the rationale for the particular set of features to be implemented in the cycle’s product.

The cycle intent:

* Is NOT a list of the features that will be in the product.
* Should expand on and fit with the system metaphor.

## Future Work

Discuss things you plan to do in the following cycle(s) or tasks that will be left for a future senior design team to implement.

# Requirements / User Stories

Include any requirements documents you have generated here. This section should at least include:

* Customer requirements documents (if any)
* User stories and use cases (if any)

## User Stories

List your user stories, in detail. Consider using a table to make the formatting easier.

A User Story is a description of a feature written in the Customer’s vernacular. It is a form of requirements specification and is used to create time estimates and development plans.

A User Story document should contain the following:

* Title/Name
* Summary: brief one-sentence description of the feature.
* Description: Full explanation of the feature in the customer’s vernacular
* Planned hours, actual hours, coder name(s), tester name(s), reviewers names, story status
* Relative priority, as compared to other user storied.

## User Stories

### Off-Screen Ghost Markers

Summary: A marker system to indicate to the user the location and velocity of any ghosts that do not fit on the screen

Description: The phone screen is much smaller than the arcade game screen. To preserve the character of the original game, the large playing area is needed, which means only a fraction can be displayed at any time. The player must have some awareness of where each ghost is on the playfield as well as whether the ghost is in “kill” or “run away” mode. The screen real estate available to the application should not decrease as result of the marker system. [and more, omitted in interest of brevity]

Hours: Total Planned: 20  
Planned this cycle: 20  
Total Actual: 30  
Actual this cycle: 30

Coder: Susan H., Joe W.

Tester: Bill S.

Reviewer: Entire team

Status: collaborative development

# Design Documentation

You should include any design documents you have generated in this section.

The design document tells how your product works, what decisions you made, why you made them. It should give a follow-on team or a new team member who comes in mid-cycle what they need to know to start working on the project:

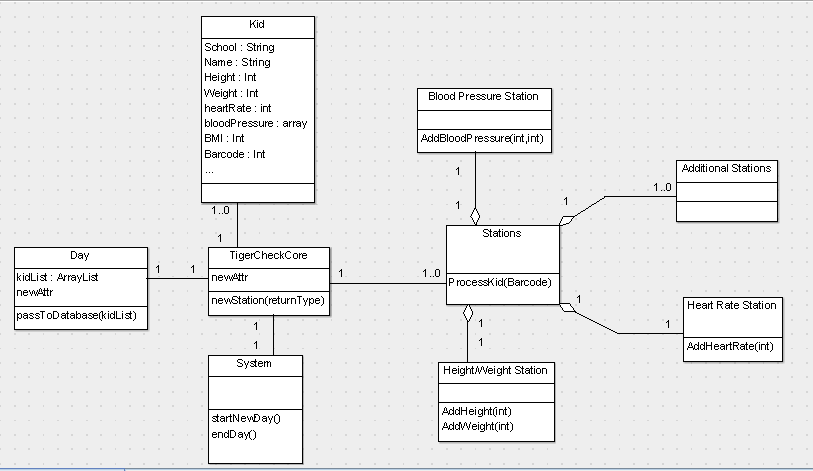
* Architecture – What is the overall picture of how your application works? Diagrams can be helpful here (e.g. UML), but prose description is essential.
* Structure – How do the pieces of code you have written interact with each other and with any framework code or libraries? Use prose, not just lists or pictures.
* Interfaces -- any interfaces to or interactions with other components in the application. Define these in detail, describing parameters, data rates, protocols, global shared variables any other important features.
* Assumptions & Dependencies – any assumptions made about or dependencies on other features or elements of the application.

Don’t rewrite the BREW, J2ME, Symbian, etc. manual -- describe the code you planned to write and the code you actually did write.

Types of design documents you that likely have already generated as part of your project design process include:

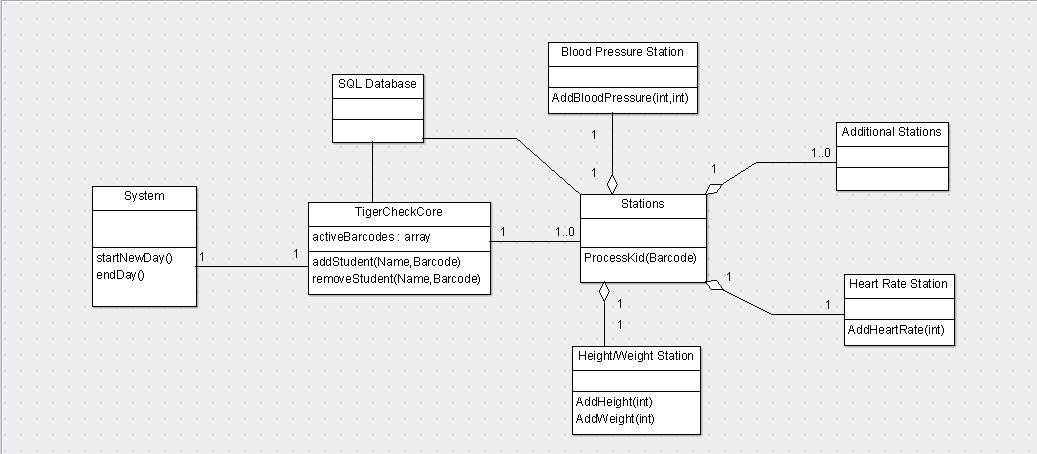
* Database and XML schema, scripts, object diagrams, etc.
* UML diagrams
* UI storyboards
* Other related documents or artifacts

After looking over the current source code of TigerCheck, our group decided rather than attempting to refactor live code, we’re planning on adding very little new functionality. We were approached by the customer to make a couple metrics visible, as well as add some quality of life changes. Due to the massive amount of refactoring that would be required to add some functionality desired by the developer, we’ve decided to rework TigerCheck with code readability and modularity as leading factors. Another big change we’re planning on changing is that we’re going to be compiling the data gathered on an SQL server for ease of manipulation of data. Due to the shortness of notice on this change, we’ve focused more on getting a very basic prototype working over planning. Despite this, we do have some basic design diagrams.



The above shows our class diagram for our original plans. We had 5 major classes and a series of subclasses. Our 4 classes were: Kid, Day, TigerCheckCore, System, and Stations. Kid was a class with a lot of variables and very few functions besides getters and setters. Day was our current plan on holding the Kid objects while the program was running. We decided on this method of holding Kids so we are not hitting the SQL Server more than once or twice per session. We’re currently planning on using TigerCheckCore as a switching station of sorts for the data. We decided on this usage of the core in order to attempt to increase modularity. System will contain any functions used to run the program, and will be the primary interface between the user and program. Stations was a superclass for all of the \_\_station classes, Stations will hold a majority of the functions of the subclasses (getKid, processKid, etc.)with each subclass adding information only needed for that station.

After further discussion, we decided to scrap our original plan in favor of the following:



We have decided to completely remove the kid class as well as the day class. This decision came about after discussions about whether frequently hitting the SQL server was as costly as originally expected. We decided to instead of having a class for the variables and a class to hold the objects for those variables, we would hit the SQL database whenever we needed data. This method will work better in our opinion because using this method allows for more compartmentalization as well as more modularity.

1) Commenting

Every new file created, for example a new asp page, new class, or anything that is a

separate file in the project, should have the following at the top

//FileName.extension

//Date Created: date

//Date Last Modified: date

//

//Function:

//

//

//Important Notes:

//

//

\*Please leave gaps between the Date, Function, and Important notes to

improve readability

2) Every method should have the following comment at the top, right before the

function:

/\*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FunctionNameHere

Date Last Modified: date

Name: nameOfPersonModifyingOrCreating

Functionality:

Parameters:

Returns:

Important notes:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*/

3) Function length

The Spring 2014 group agreed upon that no function should be more than 30 lines, unless the team agreed it was necessary to do so.

4) Naming Convention

Class Level Variables and Global Variables:

-underscore notates class or global variable

\_variablename for example

Function Level Variables:

-camel-case

-thisIsAnExapmle

Counter/Loop Variables:

- i, j, k are reserved for loops

- If you use names, make sure it’s descriptive

GUI-Variables:

-camel-case

-nameButton

-nameLabel

# Management Plan

The management plan is a high-level schedule indicating tasks and task assignments. It should include the following:

* User Story or tasks under development in this cycle.
* Team member assignments.
* Planned start and end dates for each user story and/or task this cycle. A Gantt chart should be included as a supporting figure.
* Planned code/feature freeze date.
* Any other key dates.

## Task Assignments

T.B.D.

## Development Schedule

Gantt chart here.

## Planned Code / Feature Freeze

T.B.D.

# Risk Mitigation

Include your risk identification and risk mitigation plan here.

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Impact on Project | Probability of Occurrence | Plan to prevent |
| 1. Data entered incorrectly | High | Low |  |
| 1. Kid names not deleted when sent to state kid check. | High | Low | Create temporary database to store kids’ names and unique ids. Use the ids to connect kids to their data for parent letters. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Test Plan and Test Procedures

Some good references for developing a test plan and test procedures are:

* 829-2008 IEEE Standard for Software Test Documentation
* 1008-1987 IEEE Standard for Software Unit Testing
* 1012-1986 IEEE Standard for Software Verification & Validation Plans
* 1059-1993 IEEE Guide for Software Verification & Validation Plans

The test documentation is a description of the steps used to test a feature and logs of when these tests were conducted.

The test documentation should contain the following:

* Acceptance Tests (each User Story must have at least one)
  + User Story name, Test name/ID, Test description, Required programs/files, Sequence of steps required to conduct the test.
* Unit Tests
  + User Story name, Test name/ID, Test description, Required programs/files, Sequence of steps required to conduct the test (manual) -or- Unit test code and test harness. (automated)
* Test Logs
  + User Story name, Test name/ID, Tester, Test date, Test Result, Comments

## Test Plan

A software test plan is a document that describes the objectives, scope, approach, and focus of a software testing effort. The process of preparing a test plan is a useful way to think through the efforts needed to validate the acceptability of a software product. The completed document will help people outside the test group understand the 'why' and 'how' of product validation.

It should:

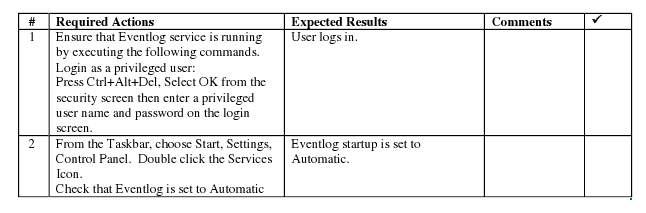
* be thorough enough to be useful but not so thorough that no one outside the test group will read it.
* State what the items to be tested are, at what level they will be tested, what sequence they are to be tested in, how the test strategy will be applied to the testing of each item, and describes the test environment.

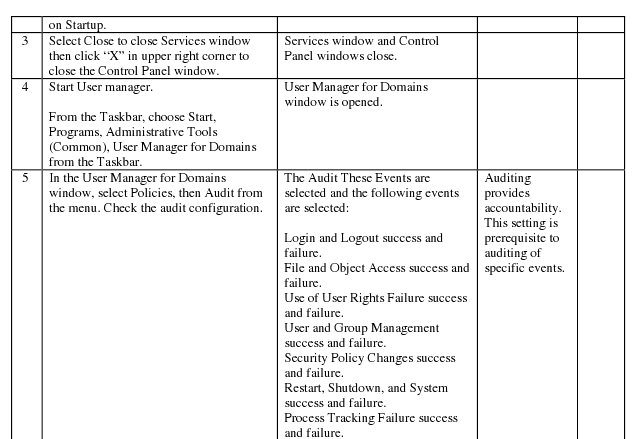
The objective of each test plan is to provide a plan for verification, by testing the software, the software produced fulfills the functional or design statements of the appropriate software specification. This generally means it should be traceable to the requirements specification.

## Test Procedures

### Procedure 1: Sample Test Procedure

This is an example of how you might organize a test procedure:





# Lessons Learned

Lessons learned is an invaluable section of your report for teams that come after you. Anything you tried that didn’t work, any technologies or solutions you considered or attempted and then abandoned, any problems with parts, components, vendors, software APIs, etc., should be documented here.

We are not interested in lessons such as “we learned how to better communicate as a team” or “we learned how to set up a database using Microsoft SQL Server.” That is pointless – you’re expected to develop team skills in this course and you’re expected to develop or improve your skills with new tools.

Instead, this section, which is arguably one of the most important in your report, should serve as a roadmap for future work and help future engineers avoid some of the problems or roadblocks you encountered.

# References

[1] Cinneide, M. Ó. and Tynan, R. A problem-based approach to teaching design patterns. *SIGCSE Bull.*, 36, 4 2004), 80-82.

[2] Alexander, C., Ishikawa, S. and Silverstein, M. *A Pattern Language: Towns, Buildings, Construction* Oxford University Press, 1977.

[3] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

# Appendix A Supporting Documents

## Status Reports

XXXXXX

## Meeting Minutes

XXXX

## Size Estimation Documentation

XXX

## Problem Reports / Change Requests

XXXX

## Correspondence

Include all correspondence, such as email, chat logs, message boards, etc., between:

* The team and the customer.
* The team and the instructor(s)/manager(s).
* Individual team members.

## Source Code

Finally, you should include source code in your PDF. This includes:

* All source code
* Database create scripts, stored procedures, etc.
* Administration info (IP addresses, server/machine names, user names, passwords, gmail lists, dropbox or sharepoint accounts, etc.)
* Version information (e.g. README.TXT)
  + A Version Description is the “README” for the delivered product.
  + A Version Description should contain the following:
    - Version number
    - Description of the application
    - Key features
    - Known bugs/issues

You DO NOT and SHOULD NOT print this lengthy portion of your report and hand it in to the instructors. Save a tree, and perhaps the planet.