Software Factory: What it is and How it Helps Improve Software Quality

Software Testing TechDay – November 19, 2019
Agenda

- **Why a Software Factory**
- Life of a Software Project
- Defining Principles
- Ultimate Software Factory
- Assembly Lines
How Deming Changed Manufacturing

Who was W. Edwards Deming?

- American engineer, statistician, and management consultant
- Deming's message: improving quality reduces cost while increasing productivity and market share
  - Improved quality results in increased customer loyalty
- Japanese manufacturers applied Deming’s techniques in the 1950s
- Result: experienced unheard-of levels of quality and productivity, which created new international demand for Japanese products

Manufacturing before Deming

- Independent processes for Design, Manufacture, and Test
- Inspect finished products for defects
- Re-work defective products to resolve issues

Manufacturing after Deming

- Integrated processes that build in quality
- Measure quality at each step in the manufacturing process
- Determine the root cause of failures and improve processes
Software is Pervasive and Growing

Why a Software Factory

Software Size (million Lines of Code)

- Modern High-end Car
- Facebook
- Windows Vista
- Large Hadron Collider
- Boeing 787
- Android
- Google Chrome
- Linux Kernel 2.6.0
- Mars Curiosity Rover
- Hubble Space Telescope
- F-22 Raptor
- Space Shuttle
Software Development on the Space Shuttle

- **Requirements Definition**
  - Started in 1973; 2000+ requirements changes were made between 1975 and 1981
  - Rigorous maintenance of requirements documents was a key factor in project success

- **Collaboration**
  - Software contract awarded to IBM, who had an office next to Johnson Space Center
  - Led to more synergy and free flow of ideas

- **Verification**
  - Established independent testing organization to be the “conscience” of the project
  - 35% of budget; just under 50% of the software team members
  - Verification started during the requirements definition phase
    - Test procedures included actual inputs to be used and expected outputs
  - First flight software had 1,020 tests

- **Change Management**
  - Average of 2 years for a new requirement to get implemented, tested and released

Despite the well-planned and well-manned verification effort, software bugs exist. Part of the reason is the complexity of the real-time system, and part is because “we didn’t do it up front enough” – “it” being thinking through the program logic and verification schemes.

*(Interview with Stanley Mann, Johnson Space Center, June 8, 1983)*
Software Development on a Modern Car

- **Requirements Definition**
  - High level requirements created by the customer/OEM
  - Project teams need to break those down into lower level requirements
    - This does not always happen

- **Collaboration**
  - Software development teams are often global
    - Time zones and native language differences make communication more challenging

- **Verification**
  - Projects are performing both high level/system tests as well as lower level software unit and integration testing
    - Number of tests executed is exploding; full test runs can take weeks or months to execute

- **Change Management**
  - Regression errors are a large source of quality issues in released products
  - Adoption of Agile means that new requirements can be implemented, tested and released in weeks or months – on some projects

Software and software development processes are much more complex than they were in the 1970s
Applying Deming's Principles to Software

Common Software Development Workflow

- Designers, Developers and Testers work in isolation
  - Independent Processes
- Periodic Testing
  - Inspect finished products for defects
- Bug Fixing
  - Re-work defective products to resolve issues

The ‘Deming’ Development Workflow

- Integrated Testing Process
  - Well defined process and goals that are shared
- Continuous Testing
  - Measure quality at each step in the manufacturing process
- Bug Prevention
  - Determine the root cause of failures, and improve processes
Why a Software Factory

The Cost of Periodic Testing

Bugs found late in the release cycle
- When many tests are run for the first time

Bug fixing is costly
- Context switching by Developers
- Long feedback loops before they know about failures

New features get released slowly
- Features get stuck in QA
- Hard to solve regressions

The following graph courtesy the NIST helps in visualizing how the effort in detecting and fixing defects increases as the software moves through the five broad phases of software development.
Catching Defects When Introduced

Source: Applied Software Measurement, Capers Jones, 1996
Cambridge University research estimates the global cost of debugging software to be $312 Billion annually.

*Source: Cambridge University, January 8, 2013*

Short Term Maintenance costs are generally double the original development and testing costs.

According to Gartner, after 5 years the original development cost is 8% of the total lifetime cost of ownership. 92% is maintenance.

Automotive SPICE®
Process Reference Model
Process Assessment Model

ISO 26262
Road vehicles - Functional safety -
Part 6:
Product development at the software level

1 Part, 128 Pages

10 Parts, 486 Pages
Agenda

Why a Software Factory

- Life of a Software Project
  - Defining Principles
  - Ultimate Software Factory
  - Assembly Lines
Software Development Plan vs Reality

Life of a Software Project

<table>
<thead>
<tr>
<th>Concept and Planning</th>
<th>Requirements and Design</th>
<th>Coding</th>
<th>Test</th>
<th>Release and Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Requirements and Design  
Coding  
Test  
Release date

Requirements and Design  
Coding  
Test  
Release date

Plan

Reality 1

Reality 2

Start development

Release date

Commercial and reputational risk: delay or bugs
Life of a Software Project

V1

System or Safe Requirements

System Test Cases

Architecture Design

Integration Test Cases

Unit Design

Unit Test Cases

Coding

Requirements

System Test Cases

Architecture Design

Integration Test Cases

Unit Test Cases

Time Taken
Life of a Software Project

V2

- **System or Safe Requirements**
- **System Test Cases**
- **Architecture Design**
- **Integration Test Cases**
- **Unit Design and Implementation**
- **Unit Test Cases**
- **Coding**

- **Requirements**
- **System Test Cases**
- **Architecture Design**
- **Integration Test Cases**
- **Unit Design**
- **Unit Test Cases**

- **Time Taken**

- =New files/cases
The Cost of Independent Processes
Agenda

Why a Software Factory
Life of a Software Project

- **Defining Principles**
  - Ultimate Software Factory
  - Assembly Lines
<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Leverage tried and tested manufacturing standards</td>
</tr>
<tr>
<td>2</td>
<td>Understand and track the requirements</td>
</tr>
<tr>
<td>3</td>
<td>Work smarter not harder</td>
</tr>
<tr>
<td>4</td>
<td>Create a continuous testing environment</td>
</tr>
<tr>
<td>5</td>
<td>Build a scalable constant integration engine</td>
</tr>
</tbody>
</table>
#1 Leverage Tried and Tested Manufacturing Standards

- Understand the System
- Just-In-Time delivery
- Find bottlenecks
- Boost communication and collaboration
- Stop the production if there is an issue
- Everyone helps if there is a major issue

An example of a Kanban board applied to Software development process
Managing the ‘work in progress’
#2 Understand and Track the Requirements

## Defining Principles

- **Basis for SW Development**
- **Basis for SW Tests (functional)**
- **Allows Traceability**
  - Requirement <-> Test Case
  - Requirement <-> Source Code
- **Allows Change Impact Analysis**
  - When source code changes:
    - -> what requirement might be affected
  - When a requirement changes:
    - -> what source code might be affected
    - -> what test case might be affected
Defining Principles

#3 Work Smarter not Harder

- Know your end game?
- How do you create the minimal number of tests for your requirements?
- How do you know when you are done?

- If only we had a way to measure our ‘doneness’...

- We do! - Accelerate your test development by measuring and tracking coverage
## #4 Create a Continuous Testing Environment

### Defining Principles

<table>
<thead>
<tr>
<th>Concept and Planning</th>
<th>Design and Requirements</th>
<th>Coding</th>
<th>Test</th>
<th>Release and Maintenance</th>
</tr>
</thead>
</table>

- **Test as early as possible**
  - Each code change
  - Use Simulation
- Define test cases with the requirements
- TDD, Scrum, Agile, Kanban
- Regression testing on each code change
  - Change-based testing
  - "Baseline test cases" for legacy code

![Project Life Cycle Diagram](image)

**Waterfall vs. SW factory**
#5 Build a Scalable Constant Integration Engine

- Continuous integration in an assembly line process
- A fully tested source code base should always be available to produce a product
- Each stakeholder can run all tests
  - Developer
  - Tester
  - CI Server
Agenda

Why a Software Factory
Life of a Software Project
Defining Principles

» **Ultimate Software Factory**
Assembly Lines
Testing: Early, Plenty, Thoroughly and Continuously

- Static Analysis
- Unit Testing
- Integration Testing
- System Testing
- Requirements Based Testing
- Code Coverage Based Testing
- Robustness Testing
- Regression Testing
- Continuous Integration: "Clean base"
  Testing of each code change. Test Cases "in sync" with code
- Parallel Testing
- Change Based Testing
Change Impact Analysis

What can change?

- Requirement
  - Linked test cases automatically invalidated
  - Source code traceability can give a clue what code needs to be changed

- Test Case
  - Re-run the changed test case

- Source Code
  - A real challenge
  - Do we need to execute all test cases?
  - Tools can help reducing the test execution times with an intelligent test case Selection
Deploying Change-Based Testing on Code Changes

- **System Requirements** → **System Test Cases**
- **High-Level Requirements** → **Integration Test Cases**
- **Low-Level Requirements** → **Unit Test Cases**

Test Cases

Source Code

Code Change
Parallel Testing

Independent Test Environments
Ultimate Software Factory

Software Quality and Testing Process Dashboard
Complete Portfolio

Benefits
- Full support in the development process
- Uniform test management, test automation (CI), result analysis and traceability
Agenda

Why a Software Factory
Life of a Software Project
Defining Principles
Ultimate Software Factory

» Assembly Lines
vVIRTUALtarget Pro generates the RTE and BSW Emulation for your AUTOSAR project enabling early design phase testing with CANoe's simulation power and vTESTstudio's test design environment.
VectorCAST/C++ provides a natural way to define low-level software unit test cases, eliminates the need to write and maintain test driver code, and makes those tests available to all the developers on the team as a reusable asset to validate their work for each code check-in.
**Software Integration Testing**

Combine multiple software units into software components to perform integration testing that focuses on how the components interact with each other from a control and data flow perspective using VectorCAST/C++. 

- **CANoe, vTESTstudio, vVIRTUALtarget, VectorCAST/QA**

- **VectorCAST/C++**
Develop automated system test cases on a virtual ECU without any hardware present, and continuously run your automated system test cases. Use Change Base Testing to quickly run the impacted system tests on each code change.
Test ECUs thoroughly by connecting the communication networks and I/O interfaces to the test system using CANoe and Vector’s VT System Hardware.
Squore can assist in visualizing, aggregating, justifying, and delivering high quality products to your customers. With different stakeholder views, Squore can produce easily interpreted visualization of KPIs.
For more information about Vector and our products please visit

www.vector.com