Industrial Year Report

IBM Level 3 CICS Service Engineer

Author:
Alexander Brown

September 10, 2012

Contents

Chapter 1

Organisational Environment

International Business Machines Corporation (IBM) is an American multinational technology and consulting corporation, with it's headquaters in New York, America. IBM sell a wide range of technical products, both hardware and software, in areas ranging from mainframe computing to nanotechnology. As well as technical products IBM also offer a range of consulting, hosting and infastructure services.

IBM are also know for innovation; holding the largest number of United States of America (US) patents, building new technologies such as IBM Watson and pushing corporate initatives like Smart Planet.

IBM is split into several different business areas which are listed below:

- Global Business Servicess (GBS)
- Global Technology Services (GTS)
- Software Group (SWG)
- Systems and Technologies Group (STG)
- Sales & Distribution
- Intergrated Technology Delivery (ITD)
- Intergrated Managed Business Process Services
- IBM Global Financing

I was employed under the Industrial Trainee (IT) scheme in SWGUnited Kingdom. Based at the Hursley site in Hampshire, the main SWG site in the United Kingdom (UK).

SWG[1] is split into five brands: DB2, Lotus, Tivoli, WebSphere and Rational. SWG in Hursley is largely WebSphere-based.

Customer Information Control System (CICS) Transaction Server[2] is a part of the WebSphere brand. Like many products in SWG CICS has several different deparments: development, test (Functional Verification (FV) test, system test, etc.) and service.

All service departments in IBM are split into three distinct levels:

[1]IBM is such a large company, it would take over 5,000 words to explain the whole structure, so I shall focus on my specific working areas.

[2]CICS Transaction Server is typically shortened to CICS internally

Level 1 Service Are the first point of contact for customers. They deal with basic problems with the product and have a general understanding of the product. If the problem can't be solved by Level 1, it is elevated to Level 2. All problems reported to Level 1 are raised as a Problem Management Request (PMR) and are tracked by Remoth Technical Assistance Information Network (RETAIN).

Level 2 Service Have a good working knowledge of the product and are typically able to diagnose and solve customer problems. If Level 2 are unable to diagnose the problem the PMR is elevated to Level 3 or, if the diagnosis reveals a problem with the product an Authorized Program Analysis Report (APAR) is raised against Level 3.

Level 3 Service Have a very good knowledge of diagnosing problems with the product and of the internals of the product and are authorised to make changes to the source code of the product to fix problems raised by APARs. The majority of Level 3 work involves handling APARs, however some specialist members of the team handle PMRs.

Due to the specialist knowledge required to work in the CICS Level 3 Service team I was not expected to deal with either APARs or PMRs[3]. My main role in the team was to maintain existing tooling and to develop new tooling which would benefit the team.

At the start of the year my main responsibilities were to maintain a tool which would gather statistics on PMRs and APARs in RETAIN and a system named "SPA", a z/OS based working environment specific to CICS Level 3 service. However, due to a process change about a year before I joined IBM, a new working environment; Rational Team Concert (RTC) was being used for all new releases of CICS.

I was initially tasked with intergrating this environment into the statistics tooling, or vice versa as RTC could potentially provide management and statisics gathering by default.

After some changes to the team I was asked to change my focus to maintaining a large tool which pulled APARs from RETAIN into RTC. This tool was also designed to perform other functionality such as delivering fixes for the Eclipse-based suite of tools for CICS to update sites and IBM's central fix management site - Fix Central.

Due to consistent issues and a lack of knowledge with this tooling, it was eventually decided to switch from this tooling to APAR Polling Tool (APT) a tool built and maintained by the IBM WebSphere MQ (MQ) Level 3 Service team and which was being considered being supported by the lab-wide build team.

This left a hole for the automation of delivering fixes for the suite of Eclipse-based tools CICS has. I was asked to develop a solution, Explorer Delivery Tool (EDT)[4], which had to be resilient to the problems which had plagued the old tooling.

Developing this tooling also increased my exposure to the team's RTC environment and lead to me being partly responsible for maintaining the structure of work items (representations of APARs and other associated tasks the team required to follow the service process).

Towards the end of my year I was also picked as part of a small team to plan and run the inductions for the SWG-based ITs for 2012-13. As well as this I was a member of the main team for Smart Cursor, a side project to continue the Extreme Blue project of the same name (BBC News Article).

[3]It can take graduates up to a year and a half to work without constant supervision.

[4]Explorer is a shortening of CICS Explorer; the main Eclipse-based tool for CICS

Chapter 2

Technical and Application Environments

Most systems I worked with in IBM were either mainframe (z/OS or z/Linux systems), RETAIN and SPA were both being z/OS application[1]; or Linux servers, typically Red Hat Enterprise Linux (RHEL).

Because CICS is almost entirely mainframe-based the department has access to several of the on-site mainframes. However, with the introduction of RTC the department has also required the use of Linux servers to host the RTC server on and to run build engines that hook into RTC and perform useful tasks, such as building CICS or polling APARs into RTC Work Items. Some of these were virtualised machines maintained by the Infastructure team in Hursley, whilst others were physical machines the department maintained.

The statsitics tools I was responsible for were hosted on two different systems; the Java-based tool ran on a RHEL CentOS 5 server running IBM WebSphere Application Server (WAS), MQ and IBM DB2 (DB2). Whilst the other was a Windows XP machine running Lotus Notes 8.5 and IBM Personal Communications (PComm) sessions into RETAIN and SPA.

For RTC build engines, the department had access to several virtualised servers; one RHEL CentOS 4 server (winlnx0u.hursley.ibm.com) which was removed towards the end of my year due to CentOS 4 going out of IBM support and was replaced with two RHEL CentOS 6 servers (cicspoller1.hursley.ibm.com and cicspoller2.hursley.ibm.com). All these servers ran RTC 3.1.0.1 Build Engines which accessed the main departmental RTC servers (jazz104.hursley.ibm.com and jazz114.hursley.ibm.com). Another machine, local to the department (hsm.hursley.ibm.com) was used as a back-up Build Engine machine, in case the server running the virtualised servers should go down. HSM was also used as a testing environment for new releases of RTC.

[1]Interesting point: RETAIN is a CICS application

Chapter 3

My role in IBM

When I first join I started with a two day induction in the North Harbour office in Portsmouth which covered the general environment I would be working, Health and Safety and the Business Conduct Guidelines. As well as some information about what I could do during the year. I received my work laptop and met with my Personal Development Manager (PDM).

On the third day I started properly in the Hursley office and joined the CICS Level 3 Service team. The team was formed of around 20-25 members including several Graduates and a another IT coming to the end of his year at IBM

The current IT, Abul, assisted me with the set-up of my laptop and with settling into the role of maintaining the team's Java-based statistics tool which run on WAS. This tool would poll RETAIN for PMRs and APARs on an hourly basis and also connected to SPA using MQ. This information was collected into a DB2 database.

Every day a report would be generated for the previous day's reciepts of PMRs APARs from the data in the database which was sent to the team leaders and the appropriate 1st and 2nd Line Managers. It would also use the database to create a similar report monthly, as well as performing a backup of data.

Finally, using Java Platform, Enterprise Edition (Java EE) the tool provided a JavaSever Pages (JSP) website which could be accessed by any member of the CICS Level 3 Service BlueGroup[1].

I started out fixing some minor issues with this tool and learning the RTC Client Application Programming Interface (API) in the effort to pull APAR data from the DB2 database across to a view in the RTC eclipse client.

Soon after Abul left I was also given charge of a Lotus Notes-based statistics tool which collected similar information as the aforementioned statistics tool. This one, however, was more manual and required the maintainer to run a scan at least once per day. This tool directly interfaced with four PComm sessions, two of which connected to RETAIN and two to UKCPSG.

The RETAIN sessions handled PMR and APAR polling. Collecting full information on PMR data, and basic data on APAR data. One of the UKCPSG sessions connected into SPA and gathered data on the current progress of APARs, whilst the other ran an internal routine called QA which collated APAR information from different SPA sub-sections for a more full report on the APAR in question.

A few months into this I was asked to help maintain the team's APAR Poller; a large tool

[1]Effectively an Lightweight Directory Access Protocol (LDAP) Group

which polled APARs in RETAIN, creating them as a work item in RTC if the APAR didn't already exist in the RTC server or updating the existing work item with any changes which had been made in RETAIN. This was originally designed to be a two-way bridge (changes in the RTC APAR work item could update the RETAIN APAR. However as permissions for the RTC work item were less secure as the ones in RETAIN so this bridge had been disabled).

The APAR Poller ran on a RTC Build Engine; this build engine would connect into a RTC server, which would hand the engine tasks to perform depending on build definitions on the RTC server.

My main role in maintaining this was to keep it working as some of the internal libraries it used updated regularly and, contrary to guidelines, sometimes broke backwards-compatability. This proved to be difficult with the sheer size of the application and the way in which it was invoked.

The application was invoked using Apache Ant; an open source tool similar to GNU Make, but designed for Java applications. Whilst this in itself was a good design descision, the readability of the Ant scripts was less than desirable. To add to this some of the variables were taken from the Build Engine Definition on the RTC server, some from the Build, some from flat files and some hardcoded into the Ant scripts. This made it nearly impossible to debug.

Mark Richards, a fellow member of the CICS Level 3 Service team also charged with maintaining the poller, and I both raised our concerns for the maintainability of the Poller in the long run. With these concerns aboard a number of meetings were set up with the /glsmq Level 3 Service team to discuss their solution, APT.

APT performed very similar function as the APAR Poller did; however it was a lot more simplistic. Despite this it was being considered by the Hursley Build Team as a lab-wide solution which they could maintain easily. Due to this the CICS Level 3 Service team had a vested interest in this solution and, after a fair few meetings, came up with a list of requirements we would need to be added to APT for us to replace the poller with.

With this came another problem. In the CICS Explorer and associated Eclipse-based tools branch of the poller code was a mechanism to deliver APAR fixes to the IBM update-sites and to Fix Central; the centralised location for the majority of fix packs. Due to the nature of CICS this site was only used for the eclipse-based tools[2].

This functionality was a useful step for the few memebers of the team who worked with the Eclipse-based tools in the delivery of fixes. Mark Richards was the one to request that I rewrite a more maintainable version of this code using Open Services Gateway Initative framework (OSGi) and following good practices for Java, OSGi and the IBM guidelines. I worked with him closely whilst designing the system and to help fix problems once I had started to write the code.

During this time I was also given another piece of work which reguarded the documentation of trace messages in one of the CICS domains. I was set-up with read-only access to the CICS 3.1, 3.2 and 4.1 source code on SPA and tasked with checking the difference between the source code definitions of trace codes and the documented versions on the IBM information centre to check for inconsistencies (i.e. added or removed entries and entires with incorrect parameters) and note down the correct version to be sent to UTD, the department which handles documentation.

Almost immediately after completing this work I was asked to create a small eclipse-plugin to print the team's Unit History Log (UHL) RTC work items in a similar form to the old version done by SPA. The UHL is a report of all the work which goes into diagnosing and fixing

[2]A fix for CICS are released as a Program Temporary Fix (PTF); a common mechanism for z/OS applications.

an APAR and are used heavily in the APAR reviews. They are also used if an APAR goes Programming Error (PE), where the fix causes a problem, to identify why the APAR went PE and how the process can be changed to prevent such action happening again.

This UHL Printer Plug-in used the Standard Widget Toolkit (SWT) Printer libraries, along with the RTC Plain-client libraries, to gather information from the currently selected UHL RTC work item, build it into a document tree and finally parse this tree and print it straight to the printer. From this I then generated an Update-Site and associated elements and hosted it on one of the departmental servers for the team to easily access.

After completing this I had a lot of time to work on EDT. To start with I had a lot of discussions with Mark about the best structure for the application. From there we decided OSGi was the best way forward due to it's focus on modularity; if one section broke due to updating libraries it would remain isolated to that section of the application without affecting any other features.

With this decision in place we then began coming up with the a decent structure for the application which would leverage OSGi. We also discussed how we would handle auditing and object serialisation in a readable format. To this end we decided to use Jackson, an Open Source library under the Apache License produced by Codehaus, which serialised Java Objects into JavaScript Object Notation (JSON) files through the use of Java annotations.

The first part of the process was to generate documentation for the fix to be delivered. To do this we turned to another Open Source library; Apache Velocity. This library acts as a templating engine allowing Java values to be passed into a predefined template with relative ease. Again this is licensed under the Apache License.

To do this a set of Plain Old Java Object (POJO)s were also defined, these would be the objects which Jackson would serialises and deserialise to JSON files.

With this complete I demonstrated the workings to both my Task Manager and Team Leader. With their approval I continued the project and ended up with weekly progress meetings with them.

The next part was adding functionality to store files to a Source Code Management (SCM) system. This would allow auditing on all items in the application. Because of the heavy reliance on RTC in the whole department and due to some issues with connectors to other SCM systems, it was decided to use RTC as the SCM system.

At this time I also decided to write the connector to the work item side of RTC. This worked slightly different from the code in the UHL Printer so I chose to write it from scratch to make it cleaner and more maintainable. This work item module read certain types of work item on the server and would populate the correct POJO with information from this work item.

Up until this point I hadn't had any problems intergrating libraries into the application, both Jackson and Velocity were contained in single jar files. The RTC libraries were large, but due to the Eclipse-based nature of RTC they were already designed to be intergrated into OSGi.

However, when it came to intergrate Fix Central's libraries into EDT, I started running into dependency problems. The problem had occurred as the Fix Central Library had dependencies on certain external libraries which contained different versions of fairly common libraries.