Appendix C: List of Tools

All the tools in this list are open-source or freely available.

* **Betamax**: An HTTP stub server that can be used to ‘stub out’ backend systems that are too complex/expensive to install in a ‘Modest-sized Tuning Environment’ (Chapter 2).  
  http://betamax.software/
* **Eclipse MAT**: A heapdump viewer that is used for locating cause of memory leaks and for other heap analysis. Used for the ‘h’ (heap) in the P.A.t.h. Checklist.  
  http://www.eclipse.org/mat/
* **EHCache**: A mature, easy-to-use Java API for caching JDBC results and just about anything else. Used to cache results for data that rarely changes, perhaps less than a few times per hour. This API is used in jpt test 03a discussed in Chapter 9 on the P in P.A.t.h. Checklist.  
  http://www.ehcache.org/
* **GarbageCat**: A command-line verbose GC analysis tool. Helpful, but not a plug-it-in-now tool because it requires JVM parameters to enable verbose GC and post-processing of the verbose GC log file. For a plug-it-in-now alternative consider getting your GC metrics from the JDK’s jstat.  
  https://github.com/mgm3746/garbagecat
* **GCEasy**: An internet-based verbose GC analysis tool with nice graphing. Like GarbageCat above, this is helpful, but not a plug-it-in-now tool because it requires JVM verbose GC parameters and post-processing of the verbose GC log file. Another downside is that you have to either upload your GC log files to their server or pay a license fee to run inside your organization. For a plug-it-in-now alternative consider getting your GC metrics from the JDK’s jstat.  
  http://gceasy.io/
* **GCViewer**: A Java Swing-based verbose GC analysis tool with graphing. Like GarbageCat above, this is helpful, but not a plug-it-in-now tool because it requires JVM verbose GC parameters and post-processing of the verbose GC log file. For a plug-it-in-now alternative consider getting your GC metrics from the JDK’s jstat.  
  https://github.com/chewiebug/GCViewer
* **Glowroot**: A great APM tool used in many chapters in this book. Provides response time and throughput data for both JDBC requests and all servlet requests. It graphs JMX data, captures thread dumps and jmap -histo output. It does all of this with very low overhead. The javaagent parameter is easy to configure and the browser interface has decent usability.  
  http://glowroot.org
* **Grizzly**: A Java API for creating programs with a lot of networking code. If your backend systems talk something other than HTTP, Grizzly would be great for coding a custom stub server that uses a proprietary network protocol. This is helpful for creating a Modest-sized Tuning environment (Chapter 2) where you have full access to the system and can therefore get many fix-test performance tuning cycles in a single day.  
  https://grizzly.java.net
* **H2**: A pure Java database that is fast, mature and stable. Used in the jpt examples that come with this book.  
  http://h2database.com
* **heapSpank**: Using data from the JDK's jmap -histo, heapSpank shows the percentage of time that byte counts are on the rise for the 15 classes most likely to be leaking. This is a plug-it-in-now monitoring tool, code by the author.  
  http://heapSpank.org
* **Hikari**: A high-performance JDBC connection pool implementation. If you deploy to multiple containers (like one client with WebSphere and another with JBoss), you might have to learn/support two different connection pool implementations, one for each container (one for WebSphere, the other for JBoss). Using Hikari would provide the exact same connection pool in either JBoss or WebSphere.  
  https://github.com/brettwooldridge/HikariCP
* **Hoverfly**: Like Betamax and Wiremock, Hoverfly is an HTTP stub server program that can be used to ‘stub out’ backend systems that are too complex/expensive to install in a ‘Modest-sized Tuning Environment’ (Chapter 2).
* **JAD**: jad.exe is a command line Java decompiler for MS-Windows. When a stacktrace points to code you don’t have the source for, use a decompiler like JAD to generate the source code from a .jar/class file. See also: JD-GUI  
  https://sourceforge.net/projects/originaljavadecompiler/
* **JavaMelody**: Similar to glowroot.org, JavaMelody is an open-source APM tool.  
  https://github.com/javamelody/javamelody
* **jcmd**: a command line tool that displays all running JVMs on a machine and their pids (process ids). Comes with the JDK but not the JRE. The JDK’s jps does similar work, but jcmd’s output more succinctly displays the Java class name being run. Similar to “ps -ef | grep java” or “tasklist | findstr java”
* **JD-GUI**: This is a Java decompiler with a small user interface that runs on Windows, Mac and Linux. When a stacktrace points to code you don’t have the source for, use a decompiler like JD-GUI to generate the source code from a .jar/class file. See also: JAD  
  http://jd.benow.ca/
* **JDBC-Performance-Logger**: Provides great live response time and throughput metrics for all JDBC requests using a small GUI. There are no plug-it-in-now tools for SQL metrics, but this monitoring tool is so easy to use, it is the next best thing.  
  https://github.com/sylvainlaurent/JDBC-Performance-Logger
* **jhiccup**: An easy-to-use Java program used for identifying when a guest on a VM (like VMWare/ESX) is being paused, keeping an application from running. Load tests are invalid when VM guests are unhealthy like this. See Chapter 2 for a way to use jhiccup that is nearly “plug-it-in-now.” However, generating graphs requires post-processing of an output file which is definitely not plug-it-in-now.  
  https://github.com/giltene/jHiccup
* **jmap**: jmap is a plug-it-in-now, command line heap analysis tool that comes with the JDK and not with the JRE. Not available in the IBM J9. The stdout of “jmap -histo <myPid>” provides counts of all instances of all classes in the JVM process id specified by <myPid>. It incurs significantly less overhead than capturing a heap dump, but it does sometimes cause multi-second JVM pauses.
* **JMeter**: Featured in Chapter 7, JMeter is a full-featured network load generator used to apply load to any network-enabled software application. JMeter can also apply load directly to Java APIs (google for JMeter Java Sampler). A large variety of high-quality plugins like those from jmeter-plugins.org make JMeter an invaluable tool.  
  http://jmeter.apache.org/
* **JMeter Plugins**: Don’t use JMeter without these plugins. Without JMeter Plugins, JMeter is lacking both functionality and usability.  
  http://jmeter-plugins.org
* **jpt (javaPerformanceTroubleshooting)**: A set of code examples written for this book that demonstrate the most commonly found performance defects, in the author’s experience. See Chapter 8 for details. Important because it shows that performance defects can be demonstrated on both small and large hardware environments, so why not load test on small hardware as detailed in Chapter 2?ß
* **jstack**: A plug-it-in-now, threaddump capture program that comes with the JDK. The Manual Threaddump Profiling (MTDP) technique in Chapter 11 shows how to use jstack to troubleshoot the same kinds of problems as a standard Java profiler like jprofiler or YourKit, but it does so in environments where Java profilers are generally not acceptable.
* **jstat**: A plug-it-in-now command line GC metric capture program that comes with the JDK but not the JRE. Chapter 12 showed how to use jstat to do a quick and simple red-yellow-green GC performance assessment on both the old gen and the young gen. This assessment can be done in just minutes with any running JVM, not just those configured with special verbose GC parameters.
* **littleMock**: littleMock is the second set of performance/load examples that come with this book. Like jpt (above), littleMock can simulate good/bad performance that is great for demonstrating how to identify and fix performance defects. A simple web page is used to configure whether littleMock performs fast or slow. Changes in the web page configuration are reflected almost immediately in littleMock performance.  
  github.com/eostermueller/littleMock
* **mock-server**: An HTTP stub server, similar to Betamax (above) and wiremock.org  
  http://www.mock-server.com/
* **Netty**: Similar to Grizzly (above).  
  [https://netty.io](https://netty.io/)
* Pinpoint: A Java APM tool similar to JavaMelody, Glowroot.org.  
  https://github.com/naver/pinpoint
* **Spring Boot**: A container-like deployment option for Java server-side applications like web and SOA servers. Includes either Jetty or Tomcat. Packages all artifacts into an executable jar file, which is much more convenient for deploying than traditional containers like JBoss and WebSphere.  
  https://projects.spring.io/spring-boot/
* **StageMonitor**: An APM tool similar to JavaMelody and Glowroot.org.  
  http://www.stagemonitor.org/
* **tcpdump**: A Linux, Mac and unix based network capture program. In Chapter 10, this was used to inspect payloads to determine whether the message was in clear text and thus needed to be compressed for smaller payload and faster performance. Comes with Linux/Unix. See windump.exe for an MS-Windows variant.
* **windump.exe**: An MS-Windows command line network capture program. See tcpdump for details.
* Wireshark: An MS-Widows network capture and analysis program, can be used like tcpdump, above.