Motivation and Explanation for New IoC Testing Framework Feature sequenceGrade

This page contains notes transferred from the meeting notepad for the GPII Technical Face to Face held at UMD on December 5th, 2016, from a session in which we explored the motivation and purpose of the new IoC testing framework feature `sequenceGrade` which has been delivered under FLUID-5903 for the Infusion 2.0 release. Documentation for this feature is held at amb26/infusion-docs/blob/FLUID-5903/IoCTestingFramework.md (TODO: update this link once these docs are merged into infusion-docs trunk).

- Intro to the IoC testing framework:
    - System of configuration that has been developed over the last 4+ years. 2 goals: (1) make it easier to test async conversational interactions (applications or systems that involve mutliple back-to-back async interactions) (2) Facilitate testing substantially sized fragments of applications.. Before it was either unit tests OR full system tests. The IoC testing framework allows us to express test-fixtures as arbitrary shaped chunks of components (ie. mocking the edges around it)
- A sort of realistic old-fashioned test fixture: http://docs.fluidproject.org/infusion/development/IoCTestingFramework.html#a-more-complex-example-using-sequence-
- An actually realistic but very small example from the real architecture:
    - https://github.com/GPII/universal/blob/master/tests/shared/DevelopmentTestDefs.js
    - What has happened here is what always happens - people take advantage of the ability to fabricate configuration in an automated way - since this is one of the primary advantages of infrastructure expressed as (JSON) configuration
    - So this usage of the framework does not correspond absolutely to what is in its docs
    - But here is an element which is recognisable: https://github.com/GPII/universal/blob/master/tests/shared/DevelopmentTestDefs.js#L52
    - FOR EXAMPLE the base level of assembly is here in Kettle: https://github.com/fluid-project/kettle/blob/master/lib/test/KettleTestUtils.js
        - This deals with the basic tasks of setting up and tearing down a set of fixtures which contain a Kettle Server
        - There is similar infrastructure in gpii-express, etc.
- The next level of aggregation up is the much more involved set of definitions in the main GPII: https://github.com/GPII/universal/blob/master/gpii/node_modules/testing/src/Integration.js
    - Core machine for running simple Integration/Acceptance tests at https://github.com/GPII/universal/blob/master/gpii/node_modules/testing/src/Integration.js#L42
    - This derives from a common testCaseHolder at https://github.com/GPII/universal/blob/master/gpii/node_modules/testing/src/Testing.js
        - This shows an architectural pattern which is in general faulty (although in this particular instance is mostly OK) - representing "points in time" by "firings of events"
        - Where the events do indeed represent architecture-wide sequence points, this makes sense
        - Where they simply represent the completion of some generic workflow, for example a request to a Couch Database, this is faulty
    - This is expressed in this way because of limitations in the architecture of the IoC testing framework
        - Here are the supported sequence fixture types http://docs.fluidproject.org/infusion/development/IoCTestingFramework.html#supported-fixture-records - there are fixtures corresponding to events and their firings
        - The new IoC testing framework is in trunk, but its docs are not - here they are
            - https://github.com/fluid-project/infusion-docs/pull/103
            - https://github.com/amb26/infusion-docs/blob/FLUID-5903/src/documents/IoCTestingFramework.md#supported-fixture-records
    - Here is a more complex and involved example of the kind of "test rot" that occurs when the requirements for reuse become more ambitious
        - https://github.com/GPII/universal/blob/master/tests/ContextIntegrationTests.js#L69
        - This SHOULD be able to reuse the common sequence elements and `testCaseHolder` from the core integration tests, but it can't (couldn't)
            - Core rot is here: https://github.com/GPII/universal/blob/master/tests/ContextIntegrationTests.js#L222
            - This is able to use the "tiny sop of reuse" which allows us to build up arrays hierarchically rather than having to author a single flat array
                - This allows us to drop in previously canned sequence arrays which are identical - or else to write functions which build these
                - This doesn't allow us to usefully parameterise these elements where they vary only in some small detail (for example, a particular context name)
                - And as we know, function calls are the root of all authorial evil (they cannot be easily re-authored or reconfigured by someone trying to reuse the overall set of fixtures)
            - Similar example of the same rot only worse is here: https://github.com/GPII/universal/blob/master/tests/JournalIntegrationTests.js
        - Here is another mad example of a struggle to obtain reuse:
            - TEST CASE DISRUPTOR used to derive various kinds of faulty OAuth 2 conversations from the one correct one:
                - https://github.com/GPII/universal/blob/master/gpii/node_modules/testing/src/CloudBasedOAuth2.js#L584
            - Reveals that fact that VARIATION OVER SEQUENCE is the other main axis of parameterisation we can't achieve
            - Here is an example of a user of this system: https://github.com/GPII/universal/blob/master/tests/platform/cloud/AcceptanceTests_chrome_oauth2.js#L102
- All of these failures of reuse point to the fact that we need a generic system that allows us to express parameterisation of test case sequences along BOTH of these axes simultaneously (and others too, preferably)
    - That is, the ability to parameterise context names where they occur within sequences AND to build up sequences collaboratively without fragile references to exact sequence numbers (which are not stable under disruption)
    - The solution involves use of some modern framework features which have grown up since the IoC Testing Framework was first written
        - And also, use of the general idiom of splitting up complex JSON configuration into Infusion grades
    - The most important of which is the use of Priorities: http://docs.fluidproject.org/infusion/development/Priorities.html
- Docs for the new "grade and priority" system are here: https://github.com/amb26/infusion-docs/blob/FLUID-5903/src/documents/IoCTestingFramework.md#using-sequencegrade-to-build-up-complex-reusable-test-sequences
    - Described by https://issues.fluidproject.org/browse/FLUID-5903
    - There is now a new supported fixture element named **sequenceGrade**

- The example in the docs shows how to achieve the "other" important form of parameterisation of test cases - variation over SEQUENCE - the ability for a derived test case to freely reassort the sequence elements written by another, and insert their own elements at an arbitrary point in the sequence - by writing priority-based "constraints" in the `fluid.test.sequence` blocks
- This example from the test cases shows how to achieve the other important form of parameterisation
    - https://github.com/fluid-project/infusion/blob/master/tests/test-core/testTests/js/IoCTestingTests.js#L514
    - The use of this parameterised grade appears here: https://github.com/fluid-project/infusion/blob/master/tests/test-core/testTests/js/IoCTestingTests.js#L537
    - This uses a fundamental new framework facility that is described in FLUID-5903: https://issues.fluidproject.org/browse/FLUID-5903
    - This wasn't possible before, but as soon as something like this becomes possible, all kinds of uses seem to crop up, e.g.:
        - https://github.com/amb26/fluid-authoring/blob/FLUID-4884/src/js/DynamicComponentIndexer.js#L23
        - This uses the "compact syntax" which is documented here: http://docs.fluidproject.org/infusion/development/Invokers.html#compact-format-for-invokers
            - (this syntax is not necessarily a best practice, but it is more concise, and the kind of loss of reuse value it represents doesn't seem to be significant in practice - people have never, it seems, in real life wanted to override a function name without overriding its arguments, or vice versa)
- NOTE: An important aspect of the improvement is that the framework automatically takes charge of constructing actual instances of "fluid.test.sequence" components in memory - this can be used to hold sequence-specific state, or even further subcomponents coordinating sequence-specific activities
    - Cindy noted that this will allow us to carry "fixture-centric components" such as Kettle request components along with the sequence grades rather than having to clumsily allocate them with increasing numbers at top level (in a manner reminiscent of a C programmer)