Chapter 6
Designing and Implementing Software and Computer Skills Simulations

Renee Barr and Christopher Coughlin

Most employers intuitively look to simulations to assess computer skills. Employers looking to assess 'computer skills' are typically looking for one of two types of assessments—general assessments of computer literacy or specific assessments focused on a particular piece of software. In addition, there is a wide range of skill level assessed, from basic (*Do you know how to save a file to a folder?*) to highly advanced (*How do you design secure networks for a global company?*). However, the bulk of the market for simulated computer assessments is in measuring basic computer literacy and knowledge of common software programs for high volume, relatively low skilled hiring. Accordingly, this chapter will examine how simulations assessing general computer literacy skills and specific skills using common software packages (e.g., Microsoft® Office) are developed, validated, and implemented (Fig. 6.1).

Simulations that focus on basic computer literacy or functioning largely involve assessing the candidate's ability to manage computer files and folders, access and use different software applications, navigate through an operating system, utilize the Internet, understand computer terminology, format and print documents, enter data, and perform other common work functions. Unlike simulations developed to assess a candidate's basic computer literacy, simulations for software programs are designed to assess a narrower skill set with a very specific program. Most of the common software programs used by the average worker come in a packaged group or office suite in which the different applications complement each other. In these office suites, there are usually a few programs that are widely used. These established programs include word processing software, database management systems, spreadsheet applications, email clients, and presentation programs. Other programs included in these productivity suites are not utilized as often by the average user, but still appeal significantly to certain workplace communities (e.g., graphics editors, collaborative software).

R. Barr (✉) · C. Coughlin
CEB, Alpharetta, GA, USA
e-mail: Renee.Barr@shl.com

C. Coughlin
e-mail: Chris.Coughlin@shl.com

M. Fetzer, K. Tuzinski (eds.), *Simulations for Personnel Selection,*
DOI 10.1007/978-1-4614-7681-8_6,

Fig. 6.1 Sample item from an MS Word® simulation

The most prevalent productivity suite, Microsoft® (MS) Office, has been on the market for almost 25 years. In the summer of 2012, Microsoft® announced that their Office suite had over a billion installations on computers across the globe (Sawers 2012). Also, the Office suite dominates the productivity software industry by controlling 90–95 % of the market share (Team 2013). According to Microsoft®, a consumer buys a new copy of the Office 2010 suite every second somewhere on earth (Arghire 2012). As indicated by its expansive reach, MS Office is commonly used by a variety of businesses because of its reasonable cost of volume licensing fees (i.e., five or more users), copious learning resources for the software itself, and the ease of working with other individuals and businesses who also utilize the Office software. Businesses use Office irrespective of their size or revenue. For example, a small "mom-and-pop" shop is just as likely to use Word, PowerPoint®, or Excel® as is a publicly traded conglomerate. Thus, it is not a surprise that simulation developers target the Office suite when building assessments for learning, training and development, and employee selection applications. In order to provide examples common to software simulations and readers, this chapter will utilize MS Office.

This chapter will review each step of the software simulation development and deployment process, taking care to note how the needs of each stakeholder (test provider, candidate, staffing clients, and corporate clients) influence the process. The development and deployment processes are broken down into four steps: design of the

simulation interface, design of the simulation content, validation of the assessment, and implementation of the assessment.

6.1 Designing Software Simulation Interfaces

When designing software simulations, the importance of replicating the 'real' experience is paramount and potentially more critical than other types of simulations. Software simulations are job knowledge tests at their core, designed to measure how well a candidate knows a particular program (windows, Internet Explorer®, Excel®, etc). As a result, accurate measurement depends on the test, faithfully simulating the actual program. Further, test users have a strong preference for software tests that look and feel like the actual program. However, the test must not only look like the software, it must also behave like the software, allowing for all the same functionalities as the actual software. Any instance where the test does not mirror the appearance or functionality of the actual software should be addressed in the instructions and tutorials at the beginning of the test.

Replicating the look and feel of the software in the test environment is critical for multiple reasons. Because the testing medium and the tool used to demonstrate software knowledge is one and the same, there is an intuitive expectation of a high fidelity test on the part of test users. This correspondence between test medium and the knowledge being assessed also means there are measurement implications that result from using low fidelity tests. When the actual functioning of the software cannot be replicated in the test environment, the test instructions provide the best opportunity to inoculate test users' expectations and minimize measurement error.

When evaluating a MS Windows® simulation, both test users and candidates expect the test to look and function just like MS Windows® does. Users of most types of simulations would like them to be as realistic as possible, but in the end, strict high fidelity does not end up being the most important criteria. While users of a customer contact simulation may wish, it looked and functioned just like the proprietary software system they use on the job, they can usually be convinced that replicating their custom software would be cost prohibitive and that it is really most important to measure a candidate's ability to use that *type* of software effectively. A new hire in a customer contact position will most likely receive weeks of training on the proprietary software once hired, making the ability to learn a software program more important to successful selection. However, software simulations are actually intended to measure a candidate's knowledge of a particular program. Lack of fidelity to the actual software intuitively feels inadequate to test users and candidates, and there are real measurement concerns that support this impression.

Using a software program is a very visual experience. The location and appearance of menus, icons, and data entry fields are fundamental to how these programs are used. When a user hovers over a menu, clicks on an icon, or enters information into a field, the program reacts the same way every time. A simulation that does not accurately replicate these characteristics leads to increased measurement error and negative candidate reactions. During the test, they are preoccupied by worries regarding these

differences and may alter how they 'use' the system to answer questions as a result. After the test, they do not feel like they had a fair opportunity to demonstrate their knowledge of the program, and they are correct.

While maximizing face validity is an important goal, it is not feasible to develop a simulation that corresponds exactly to each candidate's experience with the software. In recent versions of office suites, individual users are allowed and often encouraged to customize the program with elements that will better facilitate their individual work goals. For example, an MS Word 2010 user is able to add buttons (e.g., E-mail, Quick Print) or remove the default buttons (e.g., Save, Undo) on the Quick Access Toolbar so they can easily locate and perform their most frequent activities. While this customization results in higher satisfaction with the application and allows for quicker completion of particular software tasks, it presents a problem for simulation developers trying to create an interface that mimics exactly what each candidate views when using their version of the program at home or work. This issue can be mitigated by establishing a simulation model that replicates the default program made by the software proprietors. While the customized interface features cannot always be replicated for every candidate, the default settings are usually established as such by the software developers to maximize the benefits to the general software user.

Elements of the display size (i.e., screen resolution, aspect ratio) are another set of factors impacting developers' ability to replicate each candidate's unique experience with the software. In their own experiences with the software, candidates have a number of different display settings they might use to view the programs in accordance with their personal preferences. If a user has their monitor set at a display resolution of 1024 $\times$ 768 (in pixels) and views Word 2010 with this monitor setting, then the software program adjusts the interface to that particular resolution. So, a user who views Word 2010 with a screen resolution set at 800 $\times$ 600 sees a different Word interface than a user who works in Word with a screen resolution of 1920 $\times$ 1080. For example as seen in Figs. 6.2 and 6.3, the number and configuration of the options available on the menu bars change depending on the display size. This creates a limitation when designing one common interface for the simulation. To address this issue, the simulation developer might construct the interface at or a size below the most commonly used resolution, as indicated by data on the current usage share of different screen resolutions.

Simulation developers not only need to consider candidate expectations regarding display size, they also must take into account how the simulation will be displayed when administered to candidates. The software simulations usually use another platform to create the simulated interface and item-specific content. These software applications, designed to deliver rich internet applications or animation, are used to mimic the component (e.g., Word 2007) precisely so the candidate sees it as the software application being measured. Whichever software (e.g., Adobe® Flash®, Microsoft® Silverlight®, HTML5) is used to create the simulated interface, the simulation developer must ensure that it is supported by all web browsers (e.g., Internet Explorer®, Mozilla Firefox) and, if the software isn't supported fully, then finding out if the limitations will impact the simulation is essential in assuring a quality simulation.

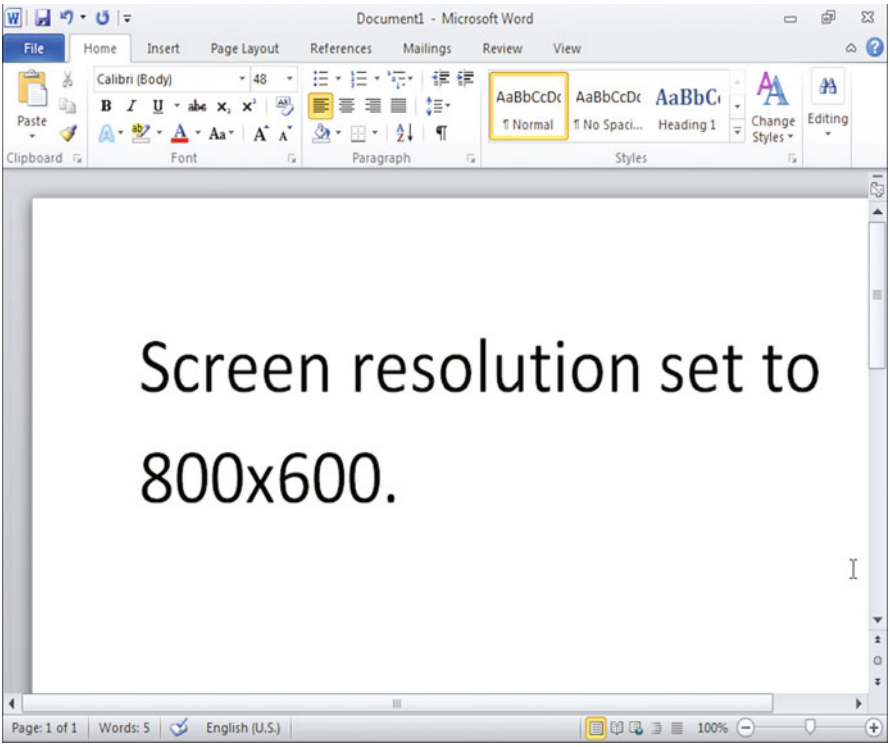

Fig. 6.2 View of MS® Word when screen resolution is set to 800 × 600. Note how many menu options are minimized or collapsed

Test users prefer software simulations to multiple-choice software tests because most people do not memorize the layout and functioning of a program by rote. A notable exception is syntax-based programs where writing code is the primary means of interacting with the software program. Most solid intermediate level users of MS Excel® do not have the menu bars and pop-up windows memorized. This makes writing multiple-choice items that accurately reflect the candidate's ability to use the software challenging. The test ends up being a better measure of who has memorized a book on how to use the program rather than who actually knows how to use the program itself. You may know how to conditionally format cells in MS Excel®, but could you write step-by-step instructions without looking at the program? Most test administrators are not looking to identify software experts or trainers using assessments, they are looking to identify functional users. Simulations that do not quite replicate the real look of a program encounter the same issues as a multiple-choice test: candidates answer some items incorrectly, not because they do not know how to perform those tasks in the program, but because the test is confusing and does not reflect how the program is used on the job.

Most of today's popular software programs offer multiple ways to perform an action (e.g., dropdown menus, shortcut keys, entering formula or code). To accurately

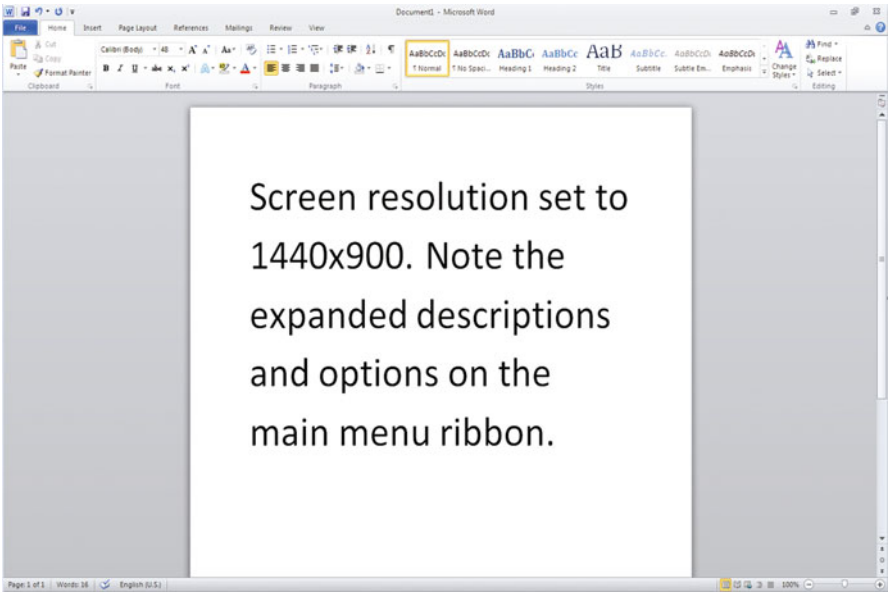

Fig. 6.3 View of MS® Word when screen resolution is set to 1440 × 900. Note the additional visible menu options

capture the look and feel of a software program in a simulated test, the simulation would ideally allow for items to be correctly answered in multiple ways. There is an added layer of complexity as each item essentially has multiple correct answers, but unless there is a job related reason for restricting the mode of correct answer, then allowing for these multiple correct responses is critical to accurate measurement of the ability to use the software on the job. For example, in MS Word 2010 there are multiple methods to bold a particular piece of text, as shown in Fig. 6.4. The three most common methods are probably to select the text and (1) use the shortcut keys 'CTL + B', (2) go to the home menu tab and click on the 'B' in the 'Font' section, or (3) right click and then left click on the 'B' in the mini toolbar above the shortcut menu. This is not an exhaustive list of options, and it would be difficult to assert than any one method is fundamentally better or a better indicator of knowledge of the program. The shortcut key is faster, but not so much as to impact the job performance of the average user. Given that most software simulations are not custom designed for a particular job or client, it would be challenging to determine the one best answer or most widely used method across test users. Finally, if a simulation does not allow for multiple methods of answering an item, then this would have to be explained in the test instructions or item stem.

Test tutorials and instructions do not make the simulation more realistic; instead, they can serve to minimize the negative impact of whatever aspects of the simulation that are not realistic. If there are limitations to how the simulation emulates the actual software, it is best to inform candidates of these discrepancies in the test instructions. This will prevent candidates from having to guess at the discrepancies and may

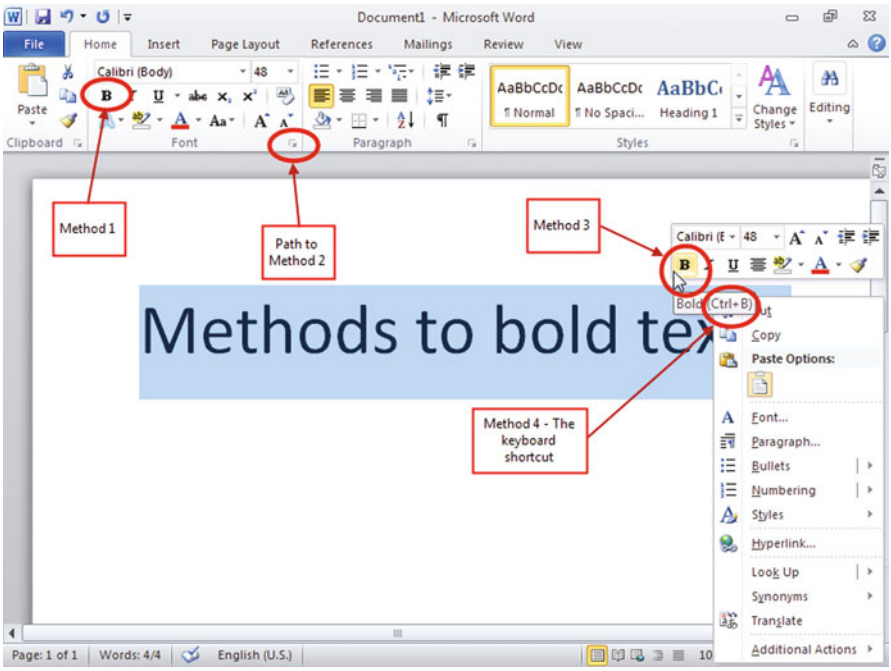

Fig. 6.4 Illustration of the multiple correct methods to bold text in MS® Word

even prevent any negative candidate reactions. Ideally, the forward instructions for a software simulation would include an interactive tutorial, allowing candidates to get a feel for the interface before being evaluated. If needed, the item stems themselves can include instructions that account for incidents where the simulation does not fully replicate the software's capabilities. For example, if the simulation cannot accommodate using shortcut keys to copy an object in PowerPoint®, the item stem could read 'Copy the smart art object on slide 2 and paste it on slide 3 without using shortcut keys.' Here the instructions start to bleed into the simulation's item content which is inexorably interrelated to the overall simulation interface. However, if an item stem requires numerous caveats in order to operate within the interface's limitations, then it would probably be best to find another item within the content domain that does not require caveats. Numerous caveats could make the instructions and/or item stems too confusing or verbose. Ideally, the interface and item stems complement each other to create an uncomplicated candidate experience.

6.2 Designing Software Simulation Content

When determining the content of a software simulation assessment, the ultimate goal is to identify and measure the knowledge that is most indicative of successfully using the software on the job. Achieving this aim requires simulation developers to balance

multiple considerations. How is the software typically used on the job? Are test administrators interested in identifying multiple skill levels? What limitations does the software interface place upon the items that can be created? What influence do past or future versions of the software have on the content of the assessment currently under construction? Managing these sometimes competing issues will play a large role in determining how successful the assessment will be. Do test administrators like what they see? Do candidates feel they had a fair chance to demonstrate their skill? Is the test valid for its intended use? Does the test ultimately predict who will perform well on the job?

The first step in designing the simulation content is identifying the software knowledge needed on-the-job and organizing it into meaningful knowledge domains. Assuming the software simulation is being designed for general use across a wide variety of users rather than being custom-developed for a particular user or job, a wide net needs to be cast in order to understand how the software is used on the job. Focus groups or interviews with software users representing the scope of jobs and industries that will be using the assessment allow the simulation developer to broadly define the universe of test content that will be needed to be created. In addition to focus groups and interviews, the simulation developer could acquire information from software trainers, master users, teachers, and application proprietors to better understand the most common and important tasks. Many software programs have a larger range of capabilities than are commonly used on the job, and understanding which capabilities are actually relevant to job performance is critical. Often understanding the relevant capabilities corresponds to understanding the range of skill levels that are of interest to test administrators. Many test administrators are much more concerned with identifying individuals with basic or intermediate skills than they are with identifying expert users. Test administrators are much more likely to want to know if a candidate can print a specific area of cells from a MS Excel® worksheet, than if the candidate can write a macro. The simulation developer should come away from the focus group, interview, and information review process with a confident understanding of the most common and important areas of knowledge for the software in question.

This broad understanding of the important knowledge areas is then organized into knowledge domains with multiple skill levels. The knowledge domains can be thought of as test scales even if they are not actually scored and reported as individual scales. They are a framework for organizing the test content that can play an important role in customizing the test, validating the test, designing the test report, and using the test to make hiring decisions. The knowledge domains organize similar software skills together and often contain skills that are related, but with different expertise levels (i.e., basic, intermediate, and advanced). For example, a simulation for MS Excel® might have knowledge domains for entering data, formatting, and using formulas. The entering data domain might contain multiple skill levels: basic tasks, such as typing text directly into a cell; intermediate tasks, such as automatically displaying data from another cell on the same worksheet; and advanced tasks, such as automatically displaying data from a cell on another worksheet or workbook. When compiling this information, there are a number of resources that a simulation

developer can locate and use to help organize information into knowledge domains and, subsequently, into subsets of expertise. Most software developers provide comprehensive training manuals, user forums, help guides, materials for certification testing, etc. that can direct this domain organization process. Also, if this is a new version of an already established software program, utilizing the content or topics of any previous software simulations could prove useful. Once the knowledge domains and skill ranges are created, they directly inform item development.

One of the most attractive benefits of using a simulated software assessment is the ability to ask a candidate to perform an action in the program. Simulated software assessment items become open-ended requests to demonstrate knowledge rather than closed multiple choice questions about the knowledge. From a measurement perspective, multiple choice questions have inherent guessing effects which are minimized in the simulation assessment format. Because of contextual software clues and the opportunity to attempt an item more than once, guessing effects are not eliminated in simulation-based assessments, but they are less problematic than an assessment where candidates can randomly select an answer from a group of four or five choices. Thus, from the candidate's perspective there are only item stems while the interface itself is one large distractor.

Writing good software simulation items is grounded in the same item writing fundamentals as any other type of item writing activity. Item stems should be clearly worded. They should be directly related to the knowledge domain they are supposed to measure and the targeted skill level. They should not be double-barreled unless you are intentionally asking the candidate to perform a multistep action. The primary constraint on software simulation item writing is the simulation interface and the technology behind it. The item must be answerable within the interface and the simulation must be able to correctly score the item response. If, in the interface design process, it was deemed too cost prohibitive to build certain aspects of the actual software program, then these areas should not be available to item writers.

The most important difference between traditional item writing and item writing for software simulation assessments is identifying the correct answer to an item. In traditional item writing, the item writer has the benefit of writing close-ended questions that define the universe of possible correct answers in the responses' options given to the candidate. The item writer should not have this kind of freedom when creating content for a robust software simulation. If the ideal simulation replicates the actual program's capabilities and the simulation developer wants to evaluate the candidate's ability to use the program in a natural manner, then an ideal item is open-ended with multiple correct responses. The item writer must identify every correct method of answering the item so that they can all be programmed as a correct response. If an item asks the candidate to cut and paste a piece of text, it is not enough for the item writer to identify a single method, or even the three most common methods, for performing this task. It is critical that the item writer identify each and every method for performing the task. Each method must then be programmed as a correct response. If the simulation interface cannot accommodate a correct method, then its robustness is lessened and the item stem must be written in a manner that tells the candidate not to take that action.

Identifying and implementing every plausible method ensures that the candidate can use atypical methods and still correctly complete the task given in the item stem. Most software developers will provide their preferred methods to complete a particular task, but, with a little research, the complete list can be identified for each task. Most productivity suites, when successful, will attempt to improve and expand on each version of the suite and create a bigger and better release in the future. Since these software applications have so many versions, a simulation developer must remember to incorporate legacy methods for certain tasks. Legacy methods, in this context, can be expressed as methods that worked in an earlier version of a software application, but have been abandoned by the proprietor to focus on the preferred, newer method to complete this task. However, these legacy methods usually work in every future version of a program. For example, a method that was emphasized for Microsoft® Word 97 might not be mentioned as a viable method to complete the same task in Word 2010, but, it will still usually work in the latter version. Discovering these legacy methods might be difficult for the simulation developer, but these are necessary for software users who developed the skills to complete the task years ago and allowed these skills to generalize to subsequent versions of the software.

After determining all the correct methods for each question, the simulation developers are also tasked with addressing which elements of the assessment should be scored as incorrect. Usually any individual action that is not advancing the simulation towards completing the task in the question stem is scored as incorrect. However, there are some actions in the software that will elicit a neutral response. These neutral responses (colloquially known as "dead elements") to certain candidate actions are established because some methods are not necessarily directly completing the task assigned in the item stem, but can be viewed as innocuous, ambiguous, or indirect to the task at hand. If these method uncertainties can not be resolved in the assessment instructions or item stem, then it is best to have nothing happen when the candidate attempts to perform this neutral action. For example, imagine that a candidate is asked to change the font of some selected text in Word 2010 and they opened up the Font dialog box to complete this task. However, they do not like the positioning of the dialog box within the simulation and they attempt to move it via a left click and drag of the title bar. Since this is not necessarily an incorrect action, the simulation developer can make a left click of the title bar of the dialog box a dead element when programming the item and nothing will appear to happen within the simulation if and when the candidate attempts to left click this title bar. Thus, candidates are not penalized for committing a neutral action.

Another item writing consideration is ensuring that the item content does not stray away from the targeted software application. It is increasingly common for productivity suites to interconnect its component programs, allowing users to seamlessly move between programs to complete tasks. For example, MS Word allows a document to be sent via email as an attachment. When completing this task, the action is initiated in Word, which delivers the user to an opened email in the user's email client (e.g., MS Outlook) with the document attached. From here, the user must know how to use the email client—an entirely different program—to complete sending the document. This task becomes problematic as the basis for an MS Word item because

it crosses the content domains of two separate programs. While MS Word and MS Office skills are likely to be positively correlated, this type of item makes for messy measurement and should be avoided. If this type of dual program task is essential to include in the assessment, then the item should only score the candidates' response where it resides inside the targeted program. Using the example of attaching a Word document to an email, only the candidate's action inside Word would be scored.

Item writing for the most popular software titles must often also take into consideration previous versions of the software. Programs such as MS Windows® and MS Office have had numerous versions since their inception in 1981 and 1989, respectively. Assessments have been built around those versions and organizations have conducted job analyses and validation work around those versions. It is advantageous to those organizations if the simulation built to assess the newest version of a program corresponds where appropriate to previous versions of the assessment. In practice, this typically means that much, but not all, of the new items directly or indirectly correspond to items in the simulation for the previous version. Updated versions of software programs are often produced to incorporate user feedback on common problems. Enhancements are carried forward in each subsequent version. When a software program has been updated many times, users' most common tasks were addressed long ago and the new enhancements rarely impact common business-oriented tasks. If the current version of the software program has completely new functionalities or no longer allows other functionalities, then the item content will need to reflect this. However, much of the time the same functionalities exist and are executed in a similar fashion as in previous version of a software program. Saving a file, opening a file, formatting, printing, and the like are important skills in each version of MS Office. It would not be unusual to have the same item stem across numerous versions. For example, the item stem "Save the open file in the 'Project XYZ' folder using the name 'Timeline"' would be applicable to any MS Office program released in the last decade. The interface, menu, and icons that the candidate would use to achieve this action might look a bit different across versions, but the item stem does not need to change. This correspondence of item content across versions helps to streamline the job analysis and validation work an organization must conduct when moving from older versions of the software simulation to a newer one.

Like any other assessment, once the items are written, a method of quality assurance (QA) should be applied to the written items and their corresponding answers or methods. The simulation developer can utilize software subject matter experts, super users, industrial and organizational (I-O) psychologists, etc. to certify that the items are appropriate for the simulation and fulfill the requirements set forth by the item writer (e.g., clearly worded, directly related to the knowledge domain). Confirming the items follow these standards and guidelines described earlier will allow the simulation developer to progress to the next steps in the development of the simulation, which is usually the tangible creation and programming of the simulation itself.

Unlike other assessments, once the simulation has been created, additional QA measures need to be applied to assure that the simulation works properly. Identifying that the response data for every item are captured accurately, the "right" methods and interface are programmed correctly, and the "wrong" or incorrect options and dead

elements are coded accurately, this is essential to guarantee that the simulation adequately mimics the real software application and works properly. If the candidate is taking the simulation through an Internet browser, the simulation developer must ensure the software (e.g., Adobe® Flash®, MS Silverlight®) used to mimic the software being measured is not causing any interactive problems with the browser. Different browsers can interact with the simulation software and absorb some of the simulated functions of particular keyboard shortcuts. For example, if HTML5 and JavaScript® are used to create a simulation for MS PowerPoint® 2010, then the simulation developer must certify that all key press combinations for every item work properly and do not interact or become absorbed by all supported browsers or operating systems. Given the ever expanding number of commonly used web browsers, this QA testing can be lengthy in terms of time and content covered. These additional QA measures should be comprehensive and allow the simulation developer to identify and resolve any simulation defects before the assessment is released to the market.

6.3 Validating Software Simulations

Validating software simulations is not fundamentally different than validating any other type of knowledge or skill assessment. Test administrators must demonstrate the job relatedness of the test content through job analysis and validation. Like many knowledge and skill assessments, software simulations are more likely to be content-validated than criterion-validated. As such, job analysis is often the critical step in implementing a software simulation assessment that will have utility for the test user and minimize legal risk.

Job analysis plays the primary role in the implementation of software simulation assessments because it is the step that ensures the correct knowledge domains, skill levels, and versions of the software program are being targeted. These are the basic characteristics that must be evaluated to demonstrate the job relatedness of software simulations. Organizations routinely seek to assess candidates' skills on common software programs across a wide range of positions. Even when the organization is interested in a single job title (e.g., Administrative Assistants), the individuals with this title can sit in a range of locations and/or business units. The level of MS Excel® and MS PowerPoint® skills required for an Administrative Assistant may vary considerably depending on, if they sit in the Finance, Marketing, or Human Resources (HR) departments. It would not be uncommon to find Administrative Assistants in a location that opened in the last couple years using the most recent version of MS Office, while the Administrative Assistants sitting in an older location associated with a regional company acquired by the multinational last year are using an older version of MS Office. Job analysis indicates if it is appropriate for the same MS Office simulation assessment to be used across these positions.

Determining which version of a software simulation assessment is appropriate for use in an organization's selection process can be surprisingly complex. Ideally, an organization is using the most recent version of a software program throughout

their organization, making the determination of which version of the assessment to use is straightforward and simple. Unfortunately, this is not the case for most test administrators. It is much more common for an organization to find that there are multiple versions of the software in use or that they are using an arguably out-of-date version of a software program. In the middle of 2012, many organizations were still using MS Office 2007 (Kelly 2012).

Both large and small time gaps between the most recent version of a software program and the version in use in an organization can make it difficult to determine which version of the assessment is most appropriate. When the gap is large, young or recently trained candidates may be particularly disadvantaged by continued use of the older version of the software assessment. These individuals are more likely to be experienced in the most recent version of the software program and less likely to have been exposed to the old version of the software program. A recent high-school graduate may score poorly on a MS Excel® 2007 assessment, not because they are inexperienced in using MS Excel® or unable to learn MS Excel®, but because they have only been exposed to the menu layout of MS Excel® 2010. However, recent software releases can disadvantage experienced candidates. For both monetary and logistical reasons, organizations are often slow to adopt new software releases. An experienced candidate may be currently employed at an organization using MS Excel® 2007. This candidate uses MS Excel® on a daily basis at an above average skill level. However, the position the candidate is testing for is at an organization that uses MS Excel® 2010 and this is the version of the simulated assessment the organization uses during the selection process. This experienced candidate is likely to score lower on the 2010 version of the assessment than she would have on the 2007 version. This is unfortunate for the hiring organization because, while this candidate might take a few weeks or months to adjust to the new version of MS Excel®, it is highly probable that she would again attain her previous skill level which is more indicative of her long term success in the position. Thus, when deciding which version of a software simulation assessment to implement, test administrators must consider the time gap between the most recently released version of the software program and the version currently in use, in addition to the job analysis results.

Once a version of the software simulation has been selected, it will ideally be validated. In practice, it is quite common for software simulations to be implemented with little or no job analysis and validation work. Perhaps because these assessments have tremendous face validity when they are well constructed, test users perceive there to be both minimal legal risk and minimal risk of implementing a test that has little or no utility. When software simulation assessments are validated, a content validation approach is almost always employed. Test administrators have very little appetite for criterion validation approaches to establishing the job relatedness of software simulation assessments. This likely arises from the perception that they are low risk assessments, but there is also a practical reason that criterion validation approaches are avoided. Concurrent criterion validation of software simulation assessments would be challenging in many organizations because the incumbent test scores would likely suffer from severe range restriction. When organizations seek to

use a software simulation assessment in the selection process for a position, it is typically because that program is used on a daily basis in the position. Few incumbents of any meaningful tenure are retained in these positions if they are not performing adequately with the software. Organizations are also generally uninterested in predictive criterion validation strategies for these positions. When the perceived risk of implementing without criterion validation is so low, why take the very real risk of hiring a candidate who scores poorly on the assessment? When the perceived risk is so low, why expend any additional resources or time on criterion validation at all?

Because software simulations are designed to assess mastery of a content domain, content validation is arguably the most appropriate approach to take. Content validating software simulation is a very straightforward process, especially if the test content is already organized into knowledge domains of varying skill levels. Subject matter experts (SMEs) can be asked to rate the job relatedness (i.e., importance, frequency, etc.) of each knowledge domain and the skill levels within each. Actual items can be used as concrete examples of each domain and skill level to help SMEs make accurate ratings. For an even more thorough content validation effort, SMEs can rate the job relatedness of each individual item. Angoff, or a similar type of expectancy rating, can be collected to help inform the process of setting a qualification standard for the assessment. Between the job analysis and content validation efforts, test users should have all the information necessary to make sound decisions regarding how to implement the software simulation in their selection processes.

6.4 Implementing Software Simulations

With their high face validity and familiar subject matter, software simulation implementations can appear very straightforward at the outset of a project. However, the ubiquity of many software programs throughout an organization can present some unusual implementation issues. Deciding which test to use, how to administer it, and where to set a passing standard while maintaining job relatedness and consistency is often a surprisingly complex assignment from an HR process perspective.

6.4.1 Choosing a Test

Once an organization decides to implement software simulations, how does it decide which tests to use for a particular opening? Even if the organization has conducted a job analysis and content validation effort around the testing, this can be a challenging decision. For example, XYZ Corp. needs to decide how to implement their MS Office testing for administrative positions. Should XYZ Corp. require all candidates to take PowerPoint®, Excel®, and Word simulations? Doing so would simplify the process and ensure consistency in the assessments' use; however, it would likely result in some candidates taking tests not actually related to the specific

administrative position for which they are applying. Administrative employees' use of PowerPoint®, Excel®, and Word likely varies by the department they sit in and the person they support. A similar issue can arise over which version of a test to use. Many organizations update their software in a piecemeal fashion, resulting in multiple versions being in use at the same time, which complicates ensuring both consistency and job relatedness when choosing a test. As a result, the person tasked with deciding which test to use for a particular opening can have several competing demands influencing what is the most job-related and consistent choice.

Determining who should be making the decision regarding which assessment to use is a key point of consideration. HR can make a centralized decision, facilitating consistency of process but potentially decreasing job relatedness. Alternatively, authority to decide which test to use can be given to hiring managers who are closer to the job. Allowing hiring managers to make the decision may increase job relatedness, but will almost certainly reduce consistency in the testing process. Finally, recruiters can be tasked with the decision in the hopes that they are in a position to better balance consistency and job relatedness. If given training and tools on how to make a testing decision, recruiters and hiring managers can be an attractive solution. However, decentralizing the decision in this manner creates a more complex process to manage and oversee.

The decision of which test to use and who gets to make that decision is not a "one size fits all" decision. In general, the more confidence an organization has in the judgment of those in the field and the less legal risk an organization faces, the more likely it is that these decisions will be decentralized and somewhat customizable to the requisition being filled. If these decisions will be decentralized, it is imperative that decision aids are created to help ensure their job relatedness and consistency. Some simple decision tree type documents outlining the factors to consider and summarizing the content and skill level of the available test options can make a large impact on how well these processes are executed upon implementation.

6.4.2 Administration Considerations

Once an organization has decided which tests to use, it then has to consider a variety of administration factors. How many chances does a candidate have to answer each question? Can the candidate take the test again? Will the test be timed? Will the test be administered remotely? Because the test in question asks candidates to demonstrate skills, the answers to these questions can be somewhat different than expected. While most selection assessments do not allow candidates multiple opportunities to answer a question, it is not uncommon for software skills assessment to do so, and it is not as great a threat to test measurement as it might seem. The bulk of the market for software simulation assessments consists of MS Office and basic computer literacy. In addition, most of the jobs in question do not require expert skill. Instead, most of the jobs in question require good working knowledge of the program. At this skill level, it is not uncommon for someone to click through a couple of menus or taskbars

to remember how to perform a task of low frequency or intermediate skill. Thus, it is not unreasonable to allow candidates two attempts to perform a task. When one considers that most software simulations consider a response to be incorrect if a candidate makes one click outside of a correct path, it is arguably more reflective of on-the-job characteristics to allow two attempts, than to allow only one. In addition, it is quite easy for candidates to make inadvertent clicks during a test session. Scoring these inadvertent clicks as incorrect responses would only add error into the skill measurement.

Test-retest policy decisions for software simulations most closely reflect best practices around general job skill and knowledge assessment. Organizations should avoid applying test-retest standards from cognitive and personality assessment to software simulations. Because a candidate can improve his or her MS Office and basic computer literacy skills with a short training course or self-study with an online tutorial or book, it is reasonable to think that candidates could improve their skill level in a short amount of time. The feasibility of quick skill improvement would support a relatively short retesting window, perhaps only 4–6 weeks. If the software simulation is adaptive and candidates do not see the same items each time they take the test, then it is easier for organizations to support a short retesting policy. However, if the software simulation is static and the same items are presented at each test session, then concern over candidates becoming too familiar with the test content becomes a legitimate issue for organizations.

The advisability of making a software simulation timed is determined by the nature of the timer and the evidence for implementing it. If job analysis indicates that a software simulation needs to assess for expert knowledge or the ability to complete tasks in a high volume or time sensitive manner, then implementing a speeded test may make sense. In this situation, the time limit on the test puts pressure on the candidate and not all candidates are expected to complete the assessment. The test instructions should tell the candidate that there is a time limit and that many candidates do not complete the assessment. More commonly, organizations want to implement a timer on a software simulation to prevent candidates from spending far too much time making their way through the test. The intent is not to rush the candidate, but rather to ensure the process is completed in a reasonable amount of time. In this situation, the time limit should be set such that the vast majority of candidates have more than enough time to complete the assessment. Organizations or test providers can examine historical data on test time to identify the time point where all but the outliers have completed the assessment. The test instructions should indicate that the test is timed and that most candidates are able to comfortably complete the assessment within the allotted time.

In today's market, most software simulation assessments for employee selection are administered remotely. Software simulations have right and wrong answers that candidates could look up in a remote setting, traditionally an argument for proctored testing. However, software simulations are most typically administered for high volume entry level positions where the cost of high touch proctored assessment is not justified by the relatively low risk of cheating. In addition to the aforementioned benefits of having a test timer, a reasonably set time limit can deter a candidate

from cheating. A reasonably set timer creates enough concern about having enough time that it discourages candidates from spending their finite testing time trying to research each item via the Internet. If an organization was highly concerned about cheating on these types of assessments, re-administering the assessment in its original or shortened form to candidates at the time they are interviewed in person would probably be the most economical course of action.

6.4.3 Setting Qualification Standards

Finally, organizations must decide how to score software simulations in the assessment process. Should the organization set a single pass/fail point? Should the pass/fail point depend on the amount of skill required by the job in question? Some of the complexity in this decision will be influenced by how an organization has decided to go about determining which test is administered. Is that decision centrally made by HR, or is it delegated to the recruiters or hiring managers? If the test is determined centrally, it is more likely that a uniform pass/fail qualification standard can be set and will be adhered to throughout the organization. However, there is some argument for not setting a single pass/fail qualification standard, and it follows much of the same reasoning reviewed earlier in the context of deciding which test to use. If the type and level of software skill needed varies greatly from job to job in the organization, then a single qualification standard may not be useful. It may be more effective to provide decision-making tools for the recruiter and/or hiring manager to help them determine what skill level the role demands and then apply that standard to candidates for the job in question. Test reports that indicate skill level in different areas may provide more useful information than a pass/fail report.

6.5 Summary and Conclusion

Basic computer literacy and familiarity with common office software is necessary for a wide range of jobs, and employers want to assess candidates' skills in those areas prior to hiring them. Software simulations are a natural solution to this business need. Like any skills assessment, a strong content validation approach is important during both development and implementation of these assessments. Because these assessments measure computer skill and are administered via computer, there are many technology considerations that must be accounted for in both development and implementation. Successful implementation of software simulations as part of a selection process requires careful consideration of which skills to measure, at which skill level, and in which version of the software. In addition, setting uniform qualification standards for these assessments is often more difficult when compared to personality or cognitive assessments. However, software simulations have substantial face validity and are generally well received by both candidates and employers.

References

Arghire, I. (2012, July 10). Microsoft's Office has over one billion users. Softpedia. http://www.news.softpedia.com. Accessed 8 Dec 2012.

Kelly, H. (2012, July 16). Can a new Office suite keep Microsoft on top? CNN. http://www.cnn.com. Accessed 2 Dec 2012.

Sawers, P. (2012, July 9). Microsoft: 1.3 billion Windows users, Office is on 1 billion desktops (and more stats). The Next Web. http://thenextweb.com. Accessed 12 Dec 2012.

Team, T. (2013, January 9). An overview why Microsoft's worth $ 42. Forbes. http://www.forbes.com. Accessed 12 Dec 2012.